

OBJECT ORIENTED TECHNOLOGIES AND JAVA PROGRAMMING

MCS-024

For
Masters In Computer Applications [MCA]

**Dinesh Verma
S. Roy**



Useful For

IGNOU, KSOU (Karnataka), Bihar University (Muzaffarpur), Nalanda University, Jamia Millia Islamia, Vardhman Mahaveer Open University (Kota), Uttarakhand Open University, Kurukshetra University, Himachal Pradesh University, Seva Sadan's College of Education (Maharashtra), Lalit Narayan Mithila University, Andhra University, Pt. Sunderlal Sharma (Open) University (Bilaspur), Annamalai University, Bangalore University, Bharathiar University, Bharathidasan University, Centre for distance and open learning, Kakatiya University (Andhra Pradesh), KOU (Rajasthan), MPBOU (MP), MDU (Haryana), Punjab University, Tamilnadu Open University, Sri Padmavati Mahila Visvavidyalayam (Andhra Pradesh), Sri Venkateswara University (Andhra Pradesh), UCSDE (Kerala), University of Jammu, YCMOU, Rajasthan University, UPRTOU, Kalyani University, Banaras Hindu University (BHU) and all other Indian Universities.

Closer to Nature



We use Recycled Paper



GULLYBABA PUBLISHING HOUSE PVT. LTD.

ISO 9001 & ISO 14001 CERTIFIED CO.

Published by:

GullyBaba Publishing House Pvt. Ltd.

Regd. Office:

2525/193, 1st Floor, Onkar Nagar-A,
Tri Nagar, Delhi-110035
(From Kanhaiya Nagar Metro Station Towards
Old Bus Stand)
011-27387998, 27384836, 27385249
+919350849407

Branch Office:

1A/2A, 20, Hari Sadan,
Ansari Road, Daryaganj,
New Delhi-110002
Ph. 011-45794768

E-mail: hello@gullybaba.com, **Website:** GullyBaba.com, GPHbook.com

New Edition

Price: ₹129/-

ISBN: 978-81-89086-74-9

Copyright© with Publisher

All rights are reserved. No part of this publication may be reproduced or stored in a retrieval system or transmitted in any form or by any means; electronic, mechanical, photocopying, recording or otherwise, without the written permission of the copyright holder.

Disclaimer: Although the author and publisher have made every effort to ensure that the information in this book is correct, the author and publisher do not assume and hereby disclaim any liability to any party for any loss, damage, or disruption caused by errors or omissions, whether such errors or omissions result from negligence, accident, or any other cause.

If you find any kind of error, please let us know and get reward and or the new book free of cost.

The book is based on IGNOU syllabus. This is only a sample. The book/author/publisher does not impose any guarantee or claim for full marks or to be passed in exam. You are advised only to understand the contents with the help of this book and answer in your words.

All disputes with respect to this publication shall be subject to the jurisdiction of the Courts, Tribunals and Forums of New Delhi, India only.

Home Delivery of GPH Books

You can get GPH books by VPP/COD/Speed Post/Courier.

You can order books by Email/SMS/WhatsApp/Call.

For more details, visit gullybaba.com/faq-books.html

Our packaging department usually dispatches the books within 2 days after receiving your order and it takes nearly 5-6 days in postal/courier services to reach your destination.

Note: Selling this book on any online platform like Amazon, Flipkart, Shopclues, Rediff, etc. without prior written permission of the publisher is prohibited and hence any sales by the SELLER will be termed as ILLEGAL SALE of GPH Books which will attract strict legal action against the offender.

Preface

This book is mainly targeted for MCA (New Course) exam of Object Oriented Technologies and Java Programming. It has been introduced in market after seeing the huge demand of ready to grasp material for exams with high level of quality, and its un-availability in market. We the GullyBaba Publishing House took a step ahead to publish the quality material focusing on exams at the same time giving you indepth knowledge about the subject.

GPH Book is the pioneer effort that provides a unique methodology so as to perform better in exams. If your goal is to attain higher grade use this powerful study tool independently or along with your text.

On the Web : www.gullybaba.com is the vital resource for your exams acting as catalyst to boost up your preparation. Now you can access us on the net through www.doctorignou.com, www.doeacconline.com, www.gphindia.com, and www.gullybabapublishinghouse.com.

Feedback about the book can be sent at mcs024writers@gullybaba.com.

Acknowledgments :

We appreciate the staff and facility support provided by GullyBaba Publishing House. In particular, we appreciate the encouragement and professional advises received from Mr. S. Roy (Academic Counsellor-St. Xavier's College, Kolkata university), Mr. Ajay Saini, Mr. Tarun Sharma, H. Faheem Ahmed and many more.

Also we would like to thank typesetting and designing team Mrs. Bhawna Verma, Mr. Kamal, Mr. Mukesh.

We gratefully acknowledges the significant contributions of Mr. Mahesh Chand, Mrs. Bimla Devi, Mrs. Bhawna Verma and our experts in bringing out this publication.

New Delhi

**Subscribe to our FREE
Newsletter**

Visit www.Gullybaba.com

Get site news, exam tips, guess paper & other hard to find information & tips.

**Contact Classroom
Teaching**

Visit www.Gullybaba.com

For IGNOU BCA MCA MBA by
**Experienced and Renowned
Faculty**

Earn While You Learn

**Become a Brand Ambassador
for 'Gullybaba'**

Visit www.Gullybaba.com

This is amazing opportunity to earn, while helping your friends to realize their dream. Become a BRAND AMBASSADOR

FREE Downloads

Visit www.Gullybaba.com

Download Firefox software absolutely free. Also many study related material.

IGNOU DOEACC BLOG

**Visit for the information which
we may not have on our website.
www.IgnouOnline.blogspot.com
www.DoeaccOnline.blogspot.com**

Contents

Chapter – 1	Basic Foundation Of OOP.....	1-15
Chapter – 2	Introduction To Java.....	16-27
Chapter – 3	Data Types.....	28-44
Chapter – 4	Java Control Statements.....	45-61
Chapter – 5	Introducing Classes, Constructors & Objects & In Java.....	62-76
Chapter – 6	Inheritance.....	77-89
Chapter – 7	Exception Handling.....	90-104
Chapter – 8	Packages & Interfaces.....	105-113
Chapter – 9	Abstract Window Tool.....	114-127
Chapter – 10	Networking Features.....	128-136
Chapter – 11	Advanced Java.....	137-146
Chapter – 12	Important Questions (Practical).....	147-170
Chapter – 13	Important Questions (Theory).....	171-198

Question Papers

(1) Dec-2005 (Solved).....	200-217
(2) June-2006 (Solved).....	218-234
(3) Dec-2006 (Solved).....	235-262
(4) June-2007 (Solved).....	263-274
(5) Dec-2007 (Solved).....	275-278
(6) June-2008 (Solved).....	279-286
(7) Dec-2008 (Solved).....	287-294
(8) June-2008 (Solved).....	295-302
(9) Dec-2009 (Solved).....	303-313
(10) June-2010 (Solved).....	314-325
(11) Dec-2010 (Solved).....	326-330
(12) June-2011 (Solved).....	331-334
(13) Dec-2011 (Solved).....	335-342
(14) June-2012 (Solved).....	343-349
(15) Dec-2012 (Solved).....	350-366
(16) June-2013	367-368
(17) Dec-2013	369-369
(18) June-2014 (Solved).....	370-382
(19) Dec-2014	383-383
(20) June-2015	384-384

(21) Dec-2015	385-386
(22) June-2016 (Solved).....	387-395
(23) Dec-2016 (Solved).....	396-404
(24) June-2017 (Solved).....	405-423
(25) Dec-2017 (Solved).....	424-439
(26) June-2018 (Solved).....	440-446
(27) Dec-2018	447-447
(28) June-2019 (Solved).....	448-452

CHAPTER 1

BASIC FOUNDATION OF OOP

Programming Paradigm

Paradigm has referred to a thought pattern in any scientific discipline. A programming paradigm is a paradigmatic style of programming (compare with a methodology, which is a paradigmatic style of doing software engineering).

A programming paradigm provides (and determines) the view that the programmer has of the execution of the program. For instance, in object-oriented programming, programmers can think of a program as a collection of interacting objects, while in functional programming a program can be thought of as a sequence of stateless function evaluations.

Just as different groups in software engineering advocate different methodologies, different programming languages advocate different programming paradigms. Some languages are designed to support one particular paradigm (Smalltalk and Java support object-oriented programming while Haskell and Scheme support functional programming), while other programming languages support multiple paradigms (such as Common Lisp, Python, and Oz.)

Many programming paradigms are as well-known for what techniques they eliminate as for what they enable. For instance, pure functional programming disallows the use of side-effects; structured programming disallows the use of goto. However, this avoiding of certain techniques can make it easier to prove theorems about a program's correctness -- or simply to understand its behavior -- without limiting the generality of the programming language.

The relationship between programming paradigms and programming languages can be complex since a programming language can support multiple paradigms. For example, C++ is designed to support elements of procedural programming, object-based programming, object-oriented programming, and generic programming. However, designers and programmers decide how to build a program using those paradigm elements. One can write a purely procedural program in C++, one can write a purely object-oriented program in C++, or one can write a program that contains elements of both paradigms.

Q1. What is an object oriented paradigm? Explain two differences between the object oriented paradigm of programming languages and the structured paradigm of programming languages. [Dec-05, Q1(a)]

The fundamental idea behind object-oriented paradigm is to combine both data and the functions that operate on that data into a single unit called *an object*. An object's functions, called member functions typically provide the only way to access its data. If you want to read a data item in an object, you call only one way to access its data. If you want to read a data item in an object, you call a member function in the object. It will read the item and return the value to you if it can't access the data directly. The data is hidden, so it is safe to accidental alterations. Data and its functions are said to be encapsulated into a single entity, ***data encapsulation and data hiding*** are key terms in the description of object-oriented paradigm. No other functions can access the data. This simplifies writing, debugging, and maintaining the program. We have to think in terms of real object which dealing with object oriented paradigm, this is supported by Object oriented languages like Java, C++ etc.

The two paradigms differ in the **(i)** sequence in which attention is given to the dynamic and static dimensions. Dynamics are emphasized in the structured paradigm and statics are emphasized in the OO paradigm. As a corollary, top-down decomposition is a strength for the Structured Paradigm approach, while the grouping of declarative commonalities via inheritance is a strength for the Object Oriented Paradigm approach.

(ii) The structured paradigm focuses on decomposing behaviors. The OO paradigm focuses on objects, classes, and inheritance. The two paradigms do not mix well. While the OO paradigm tightly integrates the development phases of analysis, design and implementation. OO methods are compatible with prototyping efforts, especially those constructed in order to elucidate otherwise unknown requirement fragments.

Q2. Java is object-oriented. What are the advantages of object-oriented programming compared to procedural oriented programming ?

What is Object Oriented Programming? How is it different from procedural programming?

JAVA IS OBJECT-ORIENTED :

Although influenced by its predecessors, Java was not designed to be source-code compatible with any other language. This allowed the Java team the freedom to design with a blank slate. Once outcome of this was a clean, usable, pragmatic approach to objects. Borrowing liberally from many seminal object-software environment of the last few decades, Java manages to strike a balance between the purist's "everything is an object" paradigm and the pragmatist's "stay out of my way" model. The object model in Java is simple and easy to extend, while simple types, such as integers, are kept as high-performance nonobjects.

Procedural languages are the languages where each statement in the language

tells the computer to do something. Get some input add these numbers, divided by 6, display that output. A program in a procedural language is a list of instructions.

For very small programs no other organizing principle (often called a paradigm) is needed. The programmer creates the list of instructions, and the computer carries them out. Pascal, C, BASIC, Fortran, and similar languages are procedural languages.

► DIVISION INTO FUNCTIONS :

When programs become larger, a single list of instructions becomes unwieldy. Few programmers can comprehend a program of more than a few hundred statements unless it is broken down into smaller units. For this reason the function was adopted as a way to make programs more comprehensible to their human creators. (The term function is used in C++ and C. In other languages the same concept may be referred to as a subroutine, a subprogram, or a procedure)

► PROBLEMS WITH STRUCTURED PROGRAMMING

The principal idea behind structured programming was as simple as the idea of “divide and conquer.” A computer program could be regarded as consisting of a set of tasks. Any task that was too complex to be simply described would be broken into a set of smaller component tasks, until the tasks were sufficiently small and self-contained to be easily understood. As programs grow ever larger and more complex, even the structured programming approach begins to show signs of strain. You may have heard about, or been involved in, horror stories of program development. The project is too complex, the schedule slips, more programmers are added, complexity increases, cost skyrocket, the schedule slips further, and finally disaster.

Analyzing the reasons for these failures reveals that there are weaknesses in the procedural paradigm itself. No matter how well the structured programming approach is implemented, large programs become excessively complex.

What are the reasons for this failure of procedural languages? One of the most crucial is the role played by data.

► DATA IS UNDervalued

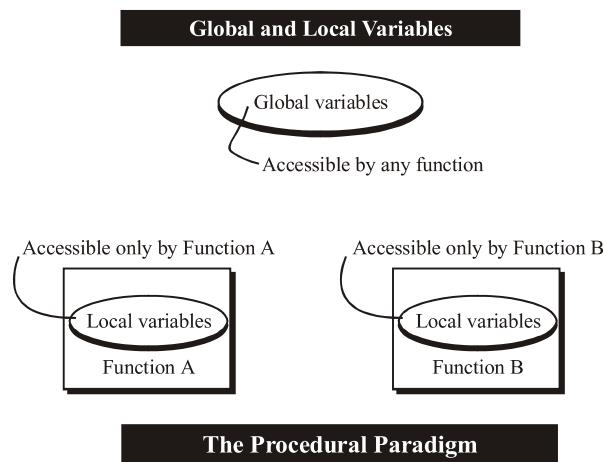
In a procedural language, the emphasis is on doing things--read the keyboard, add the values, check for error, and so on. The subdivision of a program into functions continues this emphasis. Functions do things as single program statements do. What they do may be more complex or abstract, but the emphasis is still on the action.

► WHAT HAPPENS TO THE DATA IN THIS PARADIGM?

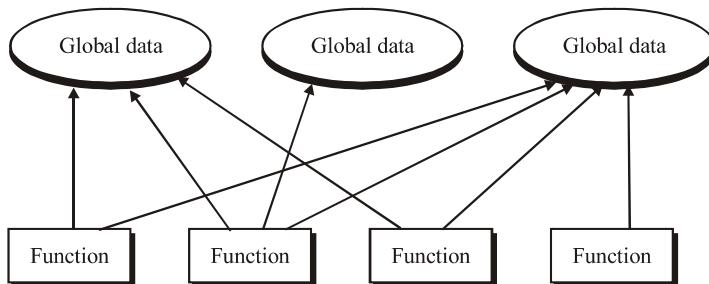
Data is, after all, the reason for a program’s existence. Data is given second-class status in the organization of procedural languages.

For example, in an inventory program, the data that makes up the inventory is probably read from a disk file into memory, where it is treated as a global variable. By global we mean that the variables that constitute the data are declared outside of any function, so they are accessible to all functions. These functions perform various operations on the data. They read it, analyze it, update it, rearrange it, display it, write it back to the disk, and so on.

We should note that most languages, such as pascal and C, also support local variables, which are hidden within a single function. But local variables are not useful for important data that must be accessed by many different functions. Figure below shows the relationship between global and local variables.



The Procedural Paradigm



Now suppose a new programmer is hired to write a function to analyze this inventory data in a certain way. Unfamiliar with the subtleties of the program, the programmer creates a function which has complete access to the data. It's like leaving your personal papers in the lobby of your apartment building: Anyone can change or destroy them. In the same way, Global data can be

corrupted by functions that have no authorization to change it. What is needed is a way to restrict access to the data, to hide it from all but a few critical functions. This will Protect the data, simplify maintenance, and offer other benefits as well.

► **TRADITIONAL LANGUAGES DOES NOT PROVIDE EXTENSIBILITY.**

One is the difficulty of creating new data types. Computer languages typically have several built-in data types: your own data type? Perhaps you want to work with complex numbers, or two-dimensional coordinates or dates-quantities the built-in-data types don't handle easily. Being able to create your own types is called extensibility; you can extend the capabilities of the language. Traditional languages are not usually extensible. Without unnatural convolutions, you can't bundle together both X and Y coordinates into a single variable called point, and then add subtract values of this type. The result is that traditional programs are more complex to write and maintain.

► **The object-oriented approach : Solution to all these problems.**

The fundamental idea behind object-oriented languages is to combine both data and the functions that operate on that data into a single unit called *an object*. An object's functions, called member functions in Java typically provide the only way to access its data. If you want to read a data item in an object, you call only way to access its data. If you want to read a data item in an object, you call a member function in the object. It will read the item and return the value to you it can't access the data directly. The data is hidden, so it is safe to accidental alterations. Data and its functions are said to be encapsulated into a single entity, ***data encapsulation and data hiding*** are key terms in the description of object-oriented languages.

No other functions can access the data. This simplifies writing, debugging, and maintaining the program.

A Java program typically consist of a number of objects, which communicates with each other by calling one another's member functions.

Data items are referred to as ***instance variables***. Calling an object's member function is referred to as ***sending a message to the object***.

Q3. Difference between class and Object.

#: **What is an object? How are objects and classes associated with each other? Also give two advantages of messages passing between objects.**

[June-06, Q1(b)]

#: **Explain the concept of Class and Object with example of each.**

[Dec-05, Q3(a, i)]

Classes and objects are separate but related concepts. Every object belongs to a class and every class contains one or more related objects.

So what exactly are classes and objects and what is the association or differ-

ence between them? A **Class** is static. All of the attributes of a class are fixed before, during, and after the execution of a program. The attributes of a class don't change. The class to which an object belongs is also (usually) static. If a particular object belongs to a certain class at the time that it is created then it almost certainly will still belong to that class right up until the time that it is destroyed.

An **Object** on the other hand has a limited lifespan. Objects are created and eventually destroyed. Also during that lifetime, the attributes of the object may undergo significant change. So let's now use an example to clarify how they are associated or different with each other. Let us consider the class **car**. Cars have a body, wheels, an engine, seats, are used to transport people between locations, and require at least one person in the car for it to move by its own power. These are some of the attributes of the class - car - and all members that this class has ever or will ever have share these attributes. The members of the class - car - are objects and the objects are **individual and specific** cars. Each individual car has a creation date (an example of an object having an attribute that is static), an owner, a registered address (examples of attributes that may or may not change), a current location, current occupants, current fuel level (examples of attributes that change quickly), and it may be covered by insurance (an example of an attribute that may or may not exist). Or A class can be thought as an architectural drawing of a building, by using that drawing we can make as many as buildings to live in. Architectural Design is of no use until the buildings are made on the basis of this drawing. And building can differ in color, size, number of rooms, etc. Here, Architectural design is *class* and Building is an *object*.

So basically the difference between a class and an object is that a class is a general concept while objects are the specific and real instances that embody that concept. When creating an object oriented program we define the classes and the relationships between the classes. We then execute the program to create, update, and destroy the objects which are the specific realization of these classes.

Two advantages of Message Passing : (i) Objects don't need to be in the same process or even on the same machine to send and receive messages back and forth to each other.

(ii) An object's behaviour is expressed through its methods, so(aside from direct variable access) message passing supports all possible interactions between objects.

Q4. What is message passing? Explain the need of message passing in object oriented programming with an example. [Dec-05, Q1(b)]

Message Passing : Object communicate with each other by message passing

when object A want to get some information from object B then A pass a message to object B, and object B in turn give the information.

Its Need, example : when student want to get certain information from office file. In this situation three objects student, clerk, and office file come in picture. Here, student object passes message to clerk that I need certain information from File. Clerk passes message to the File object which inturn gives back the information requested by Clerk object. Then this information is sent to the Student object this is called Message Passing.

Q5. What are the Characteristics of object-oriented languages ?

Let's briefly examine a few of the major elements of object-oriented languages in general and Java in particular.

1. OBJECTS

When you approach a programming problem in an object-oriented language you no longer ask how the problem will be divided into functions, but how it will be divided into object. Thinking in term of object, rather than functions has a surprisingly helpful effect on how easily programs can be designed. This results from the close match between objects in the programming sense and objects in the real world.

What kinds of things become objects in object-oriented programs? The answer to this is limited only by your imagination, but here are typical categories to start you thinking:

2. PHYSICAL OBJECTS

Automobiles in a traffic-flow simulation

Electrical components in a circuit-design program

Countries in an economics model

Aircraft in an-traffic-control system

3. ELEMENTS OF THE COMPUTER-USER ENVIRONMENT

Windows

Menus

Graphics object (lines, rectangles, circles)

The mouse and the keyboard

4. PROGRAMMING CONSTRUCTS

Customized arrays

Stacks

Linked lists

Binary trees

5. COLLECTIONS OF DATA

An inventory

- A personnel file
- A dictionary
- A table of the latitudes and longitudes of world cities

6. USER-DEFINED DATA TYPES

- Time Angles
- Complex numbers
- Points on the plane
- Components in computer games
- Positions in a board game (chess, checkers)
- Animals in an ecological simulation

The match between programming objects and real-world object is the happy results of combining data and functions: The resulting objects offer a revolution in program design. No such close match between programing constructs and the items being modeled exists in a procedural language.

Q6. Define Class variable.

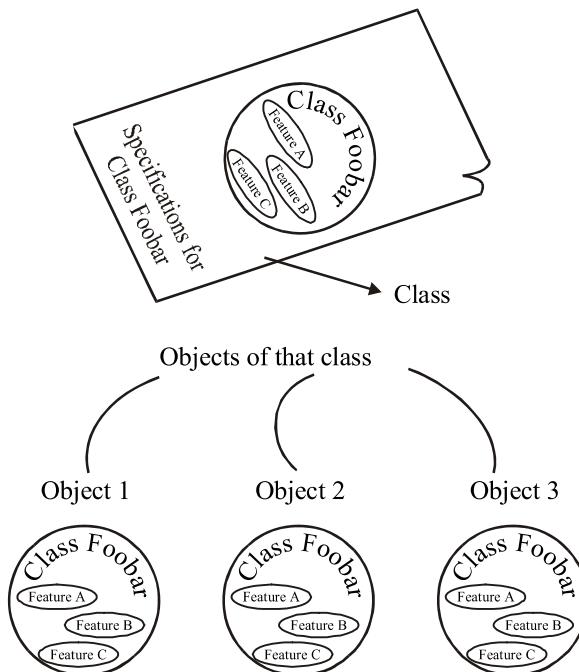
In OOP we say that objects are members of classes. What does this mean? Let's look at an analogy. Almost all computer languages have built-in data types. For instance a data type int. Meaning integer, is predefined in Java. You can declare as many variables of types int as you need in your program:

```
int day;  
int count;  
int divisor;  
int answer;
```

In a similar way you can make as many as objects of the same class, as shown in figure below. A class serves as a plan, or template. It specifies what data and what functions

A Class and Its Objects

Sales Department



will be included in object of that class. Defining the class doesn't create any abject, just as the mere existence of a type int doesn't create any variables. A class is thus a collection of similar objects. This fits our nontechnical understanding of the class of rock musicians. There is no one person called "rock musician" but specific people with specific names are members of this class if they posses certain characteristics.

Q7. Define Data Encapsulation.

The fundamental idea behind object-oriented languages is to combine both data and the functions that operate on that data into a single unit called ***an object***. An object's functions, called member functions in Java typically provide the only way to access its data. If you want to read a data item in an object, you call only way to access its data. If you want to read a data item in an object, you call a member function in the object. It will read the item and return the value to you it can't access the data directly. The data is hidden, so it is safe to accidental alterations. Data and its functions are said to be encapsulated into a single entity, ***data encapsulation and data hiding*** are key terms

in the description of object-oriented languages.

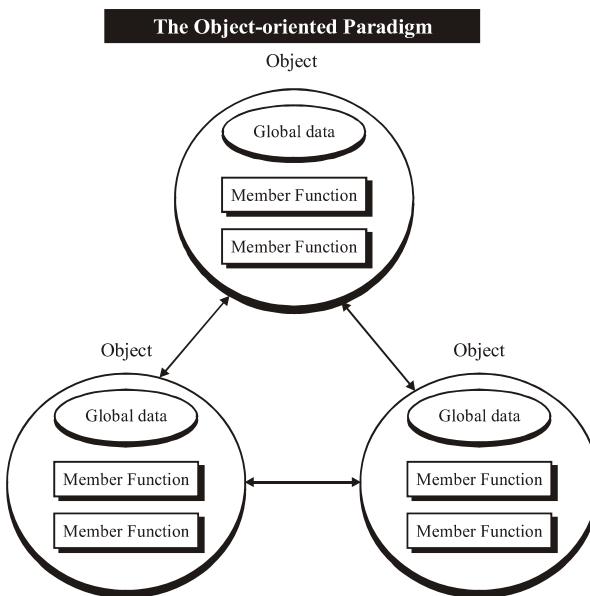
No other functions can access the data. This simplifies writing, debugging, and maintaining the program.

A class can be thought as an architectural drawing of a building, by using that drawing we can make as many as buildings to live in. Architectural Design is of no use until the buildings are made on the basis of this drawing. And building can differ in color, size, number of rooms, etc. Here, Architectural design is *class* and Building is an *object*.

Q8. Write short note on instance variable and sending message to the object.

A Java program typically consist of a number of objects, which communicates with each other by calling one another's member functions.

Data items are referred to as *instance variables*. Calling an object's member function is referred to as *sending a message to the object*. Refer the following diagram to understand the concept of data encapsulation.



INHERITANCE

Q9. What is inheritance ? Explain diagrammatically.

#: Explain the concept of Inheritance with example. [Dec-05, Q3(a,ii)]

The idea of classes leads to the idea of inheritance. In our daily lives, we use the concept of classes being divided into subclasses. We know that the class

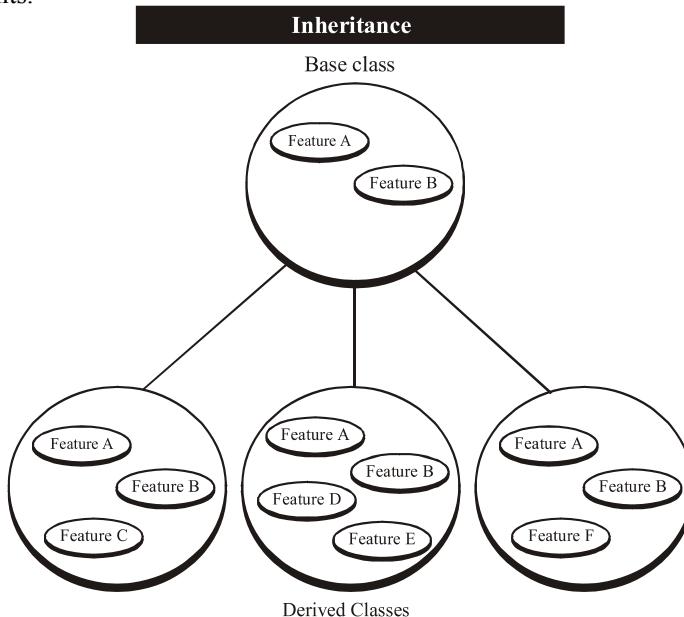
of animals is divided into mammals, amphibians, insects, bird, and so on. The class of vehicles is divided into cars, truck, buses, and motorcycles.

The principle in this sort of division is that each subclass shares common characteristics with the class from which it's derived. Cars, truck, buses, and motorcycles all have wheels and a motor; these are the defining characteristics of vehicles. In addition to the characteristics: buses, for instance, have seats for many people, while trucks have space for hauling loads.

This ideas shown in the next figure notice in the figure that features of A and B, which are part of the base class, are common to all derived classes, but that each deriver class also has features of its own.

In a similar way, an OOP class can be divided into subclass. In Java the original class is called the **base class**: other classes can be defined that share characteristics, but add their own as well. These are called **derived classes**. Don't confuse the relation of objects to classes, on the one hand with the relation of a base class to deriver classes, on the other. Objects, which exit in the computers memory, embody the exact characteristics from their base class, but add new ones of their own.

Inheritance is somewhat analogous to using functions to simplify a traditional procedural program. If we find that three different sections of a procedural program do almost exactly the same thing, we recognize an opportunity to extract the common elements of these three sections and put them into a single function . As functions do in a procedural program, inheritance shortens an object-oriented program and clarifies the relationship among program elements.



REUSABILITY

Q10. Explain the concept of reusability.

Once a class has been written, created, and debugged, it can be distributed to other programmers for use in their own programs. This is called reusability. It is similar to the way a library of functions in a procedural language can be incorporate into different programs.

In OOP the concept of inheritance provides an important extension to the idea of reusability. A programmer can take an existing class, and without modifying it. This is done by deriving a new class from the existing one. The new class will inherit the capabilities of the old one but is free to add new features of its own.

For example, you might have written (or purchased from someone else) a class that creates a menu system, such as that used in the Java's Integrated Development Environment (IDE). This class works fine, and you don't want to change it but you want to add the capability to make some menu entries flash on and off to do this you simply create a new class that inherits all the capability of the existing one but adds flashing menu entries.

ABSTRACTION

Q11. Define the term abstraction.

#: What is ‘abstraction’? Explain two advantages of abstraction with an example. [June-06, Q3(c)]

The most important term is ‘**Abstraction**’. Abstraction means ignoring those aspects of a subject that are not revealed to the current purpose in order to concentrate more fully on those that are relevant. To an analyst who is as good as a subject expert it means choosing certain things or aspects of the system over others at a time. Most of the things in the real world are intrinsically complex, far more complex than one can comprehend at a time. By using abstraction one selects only a part of the whole complex instead of working with the whole thing. It does not mean ignoring the details altogether but only temporarily, the details are embedded inside and are dealt with at a later stage. Abstraction is used by OOP as a primary method of managing complexity.

For example, here is a sample Java fragment to represent some common farm “animals” to a level of abstraction suitable to model simple aspects of their hunger and feeding. It defines an Animal class to represent both the state of the animal and its functions:

```
class Animal extends LivingThing {
```

```

Location loc;
double energyReserves;

boolean isHungry() {
    if (energy_reserves < 2.5) { return true; }
    else { return false; }
}
void eats(Food f) {
    // Consume food
    energyReserves += f.getCalories();
}
void movesTo(Location l) {
    // Move to new location
    loc = l;
}
}

```

With the above definition, one could create objects of type Animal and call their methods like this:

```

thePig = new Animal();
theCow = new Animal();
if (thePig.isHungry()) { thePig.eats(tableScraps); }
if (theCow.isHungry()) { theCow.eats(grass); }
theCow.movesTo(theBarn);

```

In the above example, the class *Animal* is an abstraction used in place of an actual animal, *LivingThing* is a further abstraction (in this case a generalisation) of Animal.

If a more differentiated hierarchy of animals is required to differentiate, say, those who provide milk from those who provide nothing except meat at the end of their lives, that is an intermediary level of abstraction, probably DairyAnimal (cows, goats) who would eat foods suitable to giving good milk, and Animal (pigs, steers) who would eat foods to give the best meat quality. Such an abstraction could remove the need for the application coder to specify the type of food, so s/he could concentrate instead on the feeding schedule. The two classes could be related using inheritance or stand alone, and varying degrees of polymorphism between the two types could be defined. These facilities tend to vary drastically between languages, but in general each can achieve anything that is possible with any of the others. A great many operation overloads, data type by data type, can have the same effect at compile-time as any degree of inheritance or other means to achieve polymorphism.

Q12. Define Association.

Association is the principle by which ideas are connected together or united. Association is used to tie together certain things that happen at some point in time or under similar conditions.

Communicating with messages is the principle for managing complexity, which is used when interfaces are involved to communicate between the entities. OOP uses classes and objects have attributes, which can be exclusively accessed by services of the objects. To change the attributes of the object (a data member of the object), a message has to be sent to the object to invoke the service. This mechanism helps in implementing encapsulation and **data abstraction**.

CREATING NEW DATA TYPES

Q13. Discuss new data types in java.

One of the benefits of objects is that give the programmer a convenient way to construct new data types. Suppose you work with two-dimensional positions (such as X and Y coordinates, or latitude and longitude) in your program. You would like to express operations on these positional values with normal arithmetic operations, such as

Position1 = position2 + origin;

Where the variables position1, position2 and origin each represent a pair of independent numerical quantities. By creating a class that incorporates these two values, and declaring position1, position2 and origin to be objects of this class, we can, in effect, create a new data type. Many features of Java are intended to facilitate the creation of new data types in this manner.

Q14. Define Operator Overloading.

POLYMORPHISM & OVERLOADING : Note that the = (equal) and + (plus) operators, used in the position arithmetic don't act the same way they do in operations on built-in types like int. The object position1(in previous question) and so on are not predefined in Java, but are programmer-defined objects of class position. How do the = and + operations know how to operate on objects ? The answer is that we can define new operations for these operators. These operators will be member functions of the position class.

Using operators or functions in different ways, depending on what they are operating on , is called **polymorphism**. One thing with several distinct forms. When an existing operator, such as + or = is given the capability to operate on a new data type, it is said to be overloaded. **Overloading** is a kind of polymorphism; it is also an important feature of OOP.

Q15. Differentiate between Data abstraction and data hiding.

In data abstraction, data structures are used without having to be concerned about the exact details of implementation.

Q16. Does procedure oriented language support the concept of class?

Yes procedural languages also support the concept of class, for example, type (data type of the language) is a class and is supported by procedural languages. You know C language support several data types. But procedural languages don't support the user-defined class that has data and functions together.

Q17. Differentiate between object based and object oriented programming.

Object based languages support the notion of objects. Object Oriented languages support the concept of class and permit inheritance between classes.

Advertise with US**Visit www.Gullybaba.com**

Advertise with us to get advertised in 31 countries. Low cost and highly effective advertisement will never be so easy

IGNOU DOEACC BLOG

**Visit for the information which we may not have on our website.
www.IgnouOnline.blogspot.com
www.DoeaccOnline.blogspot.com**

CHAPTER2

INTRODUCTION TO JAVA

The Creation of Java

Java was conceived by James Gosing, Patrick Naughton, Chris Warth, EdFrank, and Mike Sheridan at Sun Microsystems, Inc. in 1991. It took 18 months to develop the first working version. This language was finally called “Oak” but was renamed “Java” in 1995.

Java derives much of its character form C and C++. This is by intent. The Java designers knew that using the familiar syntax of C and echoing the object-oriented features of C++ would make their language appealing to the community of experienced C/C++ programmers.

The environment change there was the need for platform-independent programs destined for distribution on the Internet. Then Java fulfilled this need. However, Java also embodies changes in the way that people approach the writing of program.

“Java is to Internet programming what C was to systems programming: a revolutionary force that will change the world.”

Q1. What is the difference between an Applet and an Application ?

Java can be used to create two types of programs: applications and applets. An *application* is a program that runs on your computer, under the operating system of that computer. When used to create applications, Java is not much different from any other computer language. Rather, it is Java’s ability to create applets that makes it important. An *applet* is an application designed to be transmitted over the Internet and executed by a Java-compatible Web browser. An applet is actually a tiny Java program, dynamically downloaded across the network, just like an image, sound file, or video clip. The important difference is that an applet is an *intelligent program*, not just an animation or media file. In other words, an applet is a program that can react to user input and dynamically change-not just run the same animation or sound over and over.

As exciting as applets are, they would be nothing more than wishful thinking if Java were not able to address the two fundamental problems associated with them: security and portability. Let us define difference between them.

Java Application	Java Applet
1. GUI (Graphical User Interface)	Optional Inherently graphical
2. Memory Requirements	Minimal Java Application requirements Java Application requirements plus web browser requirements
3. Distribution	Loaded from the file system or by a Linked via HTML and transported via custom class loading process. HTTP.
4. Environmental Input	Command-Line Parameters Browser client location and size; parameters embedded in the host HTML document.
5. Method Expected by Virtual Machine	“main()” - startup method. int() - initialize method. start() - startup method stop() - pause/deactivate method destroy() - termination method paint() - drawing method.
6. Typical Applications	Network server; multimedia kiosks, public-access order-entry system for the developer tools; appliance and consumer web; online multimedia presentations; electronics control. web based animations and navigations.

Q2. Difference between Interpreter and Compiler.

A *compiler* is a program that translates a source program written in some high-level programming language (such as Java) into machine code for some computer architecture (such as the Intel Pentium architecture). The generated machine code can be later executed many times against different data each time.

An *interpreter* reads an executable source program written in a high-level programming language as well as data for this program, and it runs the program against the data to produce some results. One example is the Unix shell interpreter, which runs operating system commands interactively.

Note that both interpreters and compilers (like any other program) are written in some high-level programming language (which may be different from the language they accept) and they are translated into machine code. For example, a Java interpreter can be completely written in Pascal, or even Java. The interpreter source program is machine independent since it does not generate machine code. (Note the difference between *generate* and *translated into* machine code.) An interpreter is generally slower than a compiler because it processes and interprets each statement in a program as many times as the number of the evaluations of this statement. For example, when a for-loop is interpreted, the statements inside the for-loop body will be analyzed and evaluated on every loop step. Some languages, such as Java and LISP, come with both an interpreter and a compiler. Java source programs (Java classes with .java extension) are translated by the javac compiler into byte-code files (with .class extension). The Java interpreter, java, called the Java Virtual Machine (JVM), may actually interpret byte codes directly or may internally compile them to machine code and then execute that code.

Typically, when used in that generic manner, the term **Java compiler** refers to a program which translates Java language source code into the Java Virtual Machine (JVM) *bytecodes*. The term **Java interpreter** refers to a program which implements the JVM specification and actually executes the bytecodes (and thereby running your program).

Q3. What is Java Virtual Machine(JVM) ? Explain it.

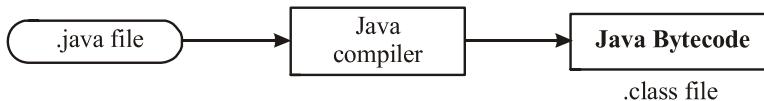
The core of Java technology, the Java Virtual Machine(JVM) is an abstract computing machine that enables the Java platform to host applications on any computer or operating system without rewriting or recompiling. Anyone interested in designing a language or writing a compiler for the Java Virtual Machine must have an in-depth understanding of its binary class format and instruction set. If you are programming with the Java programming language, knowledge of the Java virtual machine will give you valuable insight into the Java platform's security capabilities and cross-platform portability. It will increase your understanding of the Java programming language, enabling you to improve the security and performance of your programs.

Q4. How Java Works ?

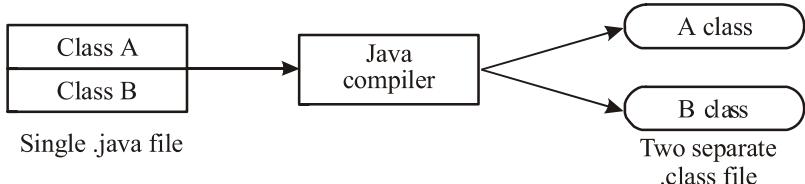
What is bytecode in Java ?

As with many other programming languages, Java uses a compiler to convert human-readable source code (.java) files into executable code (.class) files. Traditional compilers produce code that can be executed by specific hardware; for example, a Windows 98 C++ compiler creates executable program that works with Intel x⁸⁶ compatible processors. In contrast the Java compiler (javac.exe) generates architecture - independent “bytecodes”.

This bytecode can be executed by only a Java Virtual Machine (JVM), which is an idealized Java processor chip usually implemented in software rather than hardware.

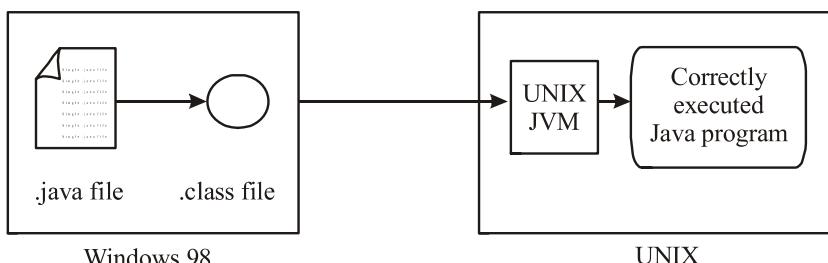


A Java source code file that can contain more than one class, on compilation will produce different “.class” file for each class in the source code.



Remember, although the details of the JVM will differ from platform to platform, all interpret the same Java bytecode.

The execution unit of the VM carries out the instructions specified in the bytecodes. The simplex execution unit is an “interpreter”, which is a program that reads the bytecodes, interprets their meaning, and then performs the associated function.



Q5. Describe main features of Java Programming Language.

#: Explain the eight basic features of Java. [June-06, Q2(a)]

: Although the fundamental forces that necessitated the invention of Java are portability and security, other factors also played an important role in molding the final form of the language.

- ▶ Simple
- ▶ Secure
- ▶ Portable

- ▶ Object-oriented
- ▶ Robust
- ▶ Multithreaded
- ▶ Architecture-natural
- ▶ Interpreted
- ▶ High performance
- ▶ Distributed
- ▶ Dynamic

Simple

Because Java inherits the C/C++ syntax and many of the object-oriented features of C++, most programmers have little trouble learning Java. Also, some of the more confusing concepts from C++ are either left out of Java or implemented in a cleaner, more approachable manner.

Beyond its similarities with C/C++, Java has another attribute that makes it easy to learn: It makes an effort not to have *surprising* features. In Java, there are a small number of clearly defined ways to accomplish a given task.

Secure

As you are likely aware, every time that you download a “normal” program, you are risking a viral infection. Prior to Java, most users did not download executable programs frequently, and those who did scanned them for viruses prior to execution. Even so, most users still worried about the possibility of infecting their systems with a virus. In addition to viruses, another type of malicious program exists that must be guarded against. This type of program can gather private information, such as credit card numbers, bank account balances, and passwords, by searching the contents of your computer’s local file system.

When you use a Java-compatible Web browser, you can safely download Java applets without fear of viral infection or malicious intent. Java achieves this protection by confining a Java program to the Java execution environment and not allowing it access to other parts of the computer. The ability to download applets with confidence that no harm will be done and that no security will be breached is considered by many to be the single most important aspect of Java.

Portable

There are many types of computers and operating systems in use throughout the world—and many are connected to the Internet. For programs to be dynamically downloaded to all the various types of platforms connected to the

Internet, some means of generating portable executable code is needed. Translating a Java program into bytecode (Binary coded classes) help makes it much easier to run a program in a wide variety of environment. The reason is straightforward: only the JVM needs to be implemented for each platform. Once the run-time package exists for a given system, any Java program can run on it. Remember, although the details of the JVM will differ from platform to platform, all interpret the same Java bytecode. If a Java program were compiled to native code, then different versions of the same program would have to exist for each type of CPU connected to the Internet. This is, of course, not a feasible solution. Thus, the interpretation of bytecode is the easiest way to create truly portable programs.

Object-oriented

Although influenced by its predecessors, Java was not designed to be source-code compatible with any other language. This allowed the Java team the freedom to design with a blank state. Once outcome of this was a clean, usable, pragmatic approach to objects. Borrowing liberally from many seminal object-software environment of the last few decades, Java manages to strike a balance between the purist's "everything is an object" paradigm and the pragmatist's "stay out of my way" model. The object model in Java is simple and easy to extend, while simple types, such as integers, are kept as high-performance nonobjects.

Robust

The multiplatformed environment of the Web places extraordinary demands on a program, because the program must execute reliably in a variety of systems. Thus, the ability to create robust programs was given a high priority in the design of Java. Because Java is a strictly typed language, it checks your code at compile time.

To understand how Java is robust, consider two of the main reasons for program failure: memory management mistakes and mishandled exceptional conditions (that is, run-time errors). Memory management can be a difficult, tedious task in traditional programming environments. For example, in C/C++, the programmer must manually allocate and free all dynamic memory. This sometimes leads to problems, because programmer will either forget to free memory that has been previously allocated or, worse, try to free some memory that another part of their code is still using. Java virtually eliminates these problems by managing memory allocation and deallocation for you. (In fact, deallocation is completely automatic, because Java provides garbage collection for unused objects.) Exceptional conditions in traditional environment often arise in situation such a division by zero or "file not found," and they must be

managed with clumsy and hard-to-read constructs. Java helps in this area by providing object-oriented exception handling. In a well-written Java program, all run-time errors can-and should-be managed by your program.

Multithreaded

Java was designed to meet the real-world requirement of creating interactive, networked programs. To accomplish this, Java supports multithreaded programming, which allows you to write programs that do many things simultaneously. The Java run-time system comes with an elegant yet sophisticated solution for multiprocess synchronization that enables you to construct smoothly running interactive systems. Java's easy-to-use approach to multithreading allows you to think about the specific behavior of your program, not the multitasking subsystem

Architecture-neutral

A central issue for the Java designers was that of code longevity and portability. One of the main problems faced by programmers is that no guarantee exists that if you write a program today, it will run tomorrow—even on the same machine. Operating system upgrades, processor upgrades, and changes in core system resources can all combine to make a program malfunction. The Java designers made several hard decisions in the Java language and the Java Virtual Machine in an attempt to alter this situation. Their goal was “write once; run anywhere, any time, forever.” To a great extent, this goal was accomplished.

Interpreted

As described earlier, Java enables the creation of cross-platform programs by compiling into an intermediate representation called Java bytecode. This code can be interpreted on any system that provides a Java Virtual Machine.

High performance

Java was designed to perform well on very low-power CPUs. The Java bytecode was carefully designed so that it would be easy to translate directly into native machine code. It is of very high performance by using a just-in-time compiler. Java run-time systems that provide this feature lose none of the benefits of the platform-independent code.

Distributed

Java is designed for the distributed environment of the Internet, because it handles TCP/IP protocols. In fact, accessing a resource using a URL is not much different from accessing a file. The original version of Java (Oak) included

features for intra-address-spaces messaging. This allowed objects on two different computers to execute procedure remotely. Java has recently revived these interfaces in a package called *Remote Method Invocation (RMI)*. This feature brings an unparalleled level of abstraction to client/server programming.

Dynamic

Java programs carry with them substantial amounts of run-time types information that is used to verify and resolve accesses to objects at run time. This makes it possible to dynamically link code in a safe and expedient manner. This is crucial to the robustness of the applet environment, in which small fragments of bytecode may be dynamically updated on a running system.

Q6. What is the relationship between Java and HTML ?

HTML(Hypertext Markup Language) is, in essence, a means of defining the logical organization of information and providing links, called hypertext links, to related information. Although HTML allows a user to read documents in a dynamic manner, HTML is markup language, and never has been, a programming language. As to be a programming language the language must have support for “conditional”, “looping” and “branching” statements which all are not supported by HTML. The only connection that HTML has to Java is that it provides the applet tag, which executes a Java applet. Thus, it is possible to embed instructions in a hypertext document that cause a Java applet to execute.

Q7. Write a simple program in Java. Explain all the keywords used.

“Hi Students, Welcome to GullyBaba” Program

```
/* This is a simple Java Program*/
class HiStudents
{
    public static void main(String args[ ])
    {
        System.out.println("Hi Students, Welcome to GullyBaba");
    }
}
```

Save this text file as “hi.java” in JDK’s |bin| folder.

Go to Command prompt and follow these steps.

1. Compile Java file.

bin > javac hi.java

2. Run your Class file.

bin > java hi

All Java applications begin execution by calling main(). (This is just like C/C++). The public keyword is an access specifier, which allows the programmer to control the visibility of class members. When a class member is preceded by public, then that member may be accessed by code outside the class in which it is declared. (The opposite of public is private, which prevents a member from being used by code defined outside of its class). In this case, main() must be declared as public, since it must be called by code outside of its class when the program is started. The keyword static allows main() to be called without having to instantiate a particular instance of the class. This is necessary since main() is called by the Java interpreter before any object is made. The keyword void simply tells the compiler that main() does not return a value. As you will see, methods may also return values.

In main(), there is only one parameter, albeit a complicated one. String args[] declares a parameter named args, which is an array of instances of the class String. (Arrays are collections of similar objects.) Objects of type String store character strings. In this case, args receives any command-line arguments present when the program is executed.

Q8. Discuss all Atomic Elements used in JAVA.

Java programs are a collection of whitespace, identifiers, comments, literals, separators and keywords.

Whitespace → Java is a free-form language. This means that you do not need to follow any special indentation rules. For example, the Example program could have been written all on one line or in any other strange way you felt like typing it, as long as there was at least one whitespace character between each token that was not already delineated by an operator or separator. In Java, whitespace is a space, tab or newline.

Identifiers →

- (a) Identifiers are used for class names, method names and variable names.
- (b) An identifier may be any descriptive sequence of uppercase and lowercase letters, numbers, or the underscore and dollar-sign characters.
- (c) They must not begin with a number, lest they be confused with a numeric literal.
- (d) Java is case-sensitive, so VALUE is a different identifier than Value.

Some examples of valid identifiers are:

AvgTemp count a4 \$test
this_is_ok

Invalid variable names include:

2count high-temp Not/ok

Literals → A literal is an actual value such as 35 or “Hello”. A constant value in Java is created by using a literal representation of it. For example, here are some literals:

100 98.6 ‘X’ “This is a test”

Left to right, the first literal specifies an integer, the next is a floating-point value, the third is a character constant, and the last is a string. A literal can be used anywhere a value of its type is allowed.

Comment → There are three types of comments defined by Java. You have already seen two: single-line and multiline. The third type is called a documentation comment. This type of comment is used to produce an HTML file that documents your program. The documentation comment begins with a `/**` and ends with a `*/`.

Separators → In Java, there are a few characters that are used as separators. The most commonly used separator in Java is the semicolon. As you have seen, it is used to terminate statements. The separators are shown in the following table.

Symbol Name	Purpose
()	Parentheses Used to contain lists of parameters in method definition and invocation. Also used for defining precedence in expressions, in control statements, and surrounding cast types.
{ }	Braces Used to contain the values of automatically initialized arrays. Also used to define a block of code, for classes, methods, and local scopes.
[]	Brackets Used to declare array types. Also used when dereferencing array values.; Semicolon Terminates statements., Comma Separates consecutive identifiers in a variable declaration. Also used the chain statements together inside a for statement.
.	Period Used to separate package names from subpackages and classes. Also used from subpackages and classes. Also used to separate a variable or method from a reference variable.

The Java Keywords

There are 48 reserved keywords currently defined in the Java languages (see following table). These keywords, combined with the syntax of the operators and separators, form the definition of the Java language. These keywords cannot be used as names for a variable, class, or method.

abstract	const	finally	int	public	this
boolean	continue	float	interface	return	throw
break	default	for	long	short	throws
byte	do	goto	native	static	tansient
case	double	if	new	strictfp	try
catch	else	implements	package	super	void
char	extends	import	private	switch	volatile
class	final	instanceof	protected	synchronized	while

Table : Java Reserved Keywords

The keywords const and goto are reserved but not used. In the early days of Java, several other keywords were reserved for possible future use. However, the current specification for Java only defines the keywords shown in table. In addition to the keywords, Java reserves the following: true, false, and null. These are values defined by Java. You may not use these words for the names of variables, classes and so on.

Q9. What is instance variable? Two different objects of same class can have same value for an instance variable or not?

Every object in the world has its state and behavior. For example, a student object may have name, programme of a study, semester are known as instance variables. The value of instance variables at a particular time to decide the state of an object.

Instance variables of different objects of the same class may have same value for example two different student may have same name, but its obvious that they are not same.

Q10. Why an object can be used in different programs without modification?

Objects are instance of a class are combination of variables and methods used to represent state and behaviour of objects. When the class is defined its object can be used anywhere with the properties which are defined for it in class without modification.

Q11. Why is abstract keyword but Abstract in not?

Because Java is case-sensitive, so even though abstract is a keyword but Abstract is not a keyword at all.

Q12. What is synchronization and why is it important?

With respect to multithreading, synchronization is the capability to control the access of multiple threads to shared resources. Without synchronization, it is possible for one thread to modify a shared object while another thread is in the process of using or updating that object's value. This often leads to significant errors.

Q13. What are two major advantages of automatic de-allocation of memory?

There are two major advantages of automatic garbage collection

- i) Increase programmer productivity.
- (ii) Ensure program integrity.

FREE Downloads

Visit www.Gullybaba.com

Download Firefox software absolutely free. Also many study related material.

**Earn While You Learn
Become a Brand Ambassador
for 'Gullybaba'**

Visit www.Gullybaba.com

This is amazing opportunity to earn, while helping your friends to realize their dream. Become a BRAND AMBASSADOR

CHAPTER 3

DATA TYPES

Q1. List all primitive data types, size and range of values they accept.

List all primitive data types with their sizes as defined in Java.

Java defines eight simple (or elemental) types of data: byte, short, int, long, char, float, double and boolean. These can be put in four groups :

- ✓ **Integers** : This group includes byte, short, int and long, which are for whole-valued signed number.
- ✓ **Floating-point numbers** : This group includes float and double, which represent numbers with fractional precision.
- ✓ **Characters** : This group includes char, which represents symbols in a character set, like letters and numbers.
- ✓ **Boolean** : This group includes boolean, which is a special type for representing true/false values.

Integers : Java defines four integer types: byte, short, int and long. All of these are signed, positive and negative values. The width and ranges of these integer types vary widely, as shown in this table:

Name	Width	Range
long	64	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
int	2	-2,147,483,648 to 2,147,483,647
short	16	-32,768 to 32,767
byte	8	-128 to 127

Floating-Point Types : Floating-point numbers, also known as real numbers, are used when evaluating expressions that require fractional precision.

Name	Width in Bits	Range
double	64	1.7e-308 to 1.7e+308
float	32	3.4e-038 to 3.4e+038

Q2. Discuss characters in Java.

#. What is the difference between ASCII code and Unicode ?

For character data, each character is assigned a unique code. We can have a

character set of 26 capital letters, 10 digits, 0 through 9, and a few punctuation marks such as space, period, minus sign, comma and carriage return. More sophisticated character sets include capital letters and small letters, the 10 digits, a wide set of punctuation characters, a collection of special characters useful in mathematics and even greek letters. Once such widely used code is ASCII (American Standard Code for Information Interchange). Most of the present day computers use this code for storing data in memory. The ASCII character set consists of 128 characters and 7 bits are used to code each character. e.g. a=1100001 (141 octal). Now even more sophisticated character representation supported by Unicode. Java uses Unicode to represent characters. Unicode define a fully international character set that can represent all of the characters found in all human languages. It is a unification of dozens of character sets, such as Latin, Greek, Arabic, Cyrillic, Hebrew, Katakana, Hangul and many more. For this purpose, it requires 16 bits. Thus, in Java char is a 16-bit type. The range of a char is 0 to 65,536. There are no negative chras. The standard set of characters known as ASCII still ranges from 0 to 255.

Booleans

Q3. What is Boolean?

ANS : Java has a simple type, called boolean, for logical values. It can have only one of two possible values, true or false. This is the type returned by all relational operators, such as a < b. Boolean is also the type required by the conditional expressions that govern the control statements such as if and for.

Q4. Write the syntax, and explain, any four constructors of the String class.

[June-06, Q2(c)]

The seven String class's constructors look like this:

```
public String()
public String(String value)
public String(char value[])
public String(char value[], int offset, int count)
```

These constructors create, respectively, the following:

- Null string
- String object from another String object (including from a string literal)
- String from an array of characters
- String from a subarray of characters

Note : There's a big difference between a null String object and a null string. When you declare a String object with a line like String str;, you are declaring an object of the String class that has not yet been instantiated. That is, there is not yet a String object associated with str, meaning that the String object is

null. When you create a String object with a line like String str = “”;, you are creating a fully instantiated String object whose string contains no characters (has a string length of zero). This is called a *null string*.

Q5. List some Derived Data Types.

#. Discuss the difference between String and StringBuffer class as defined in Java.

String & String Buffer : A string is a sequence of character. The **String** datatype is actually a class of the core API (java.lang.string), rather than a built-in type, but it is used so frequently that it is appropriate to cover here. Java compiler recognizes characters within double quotation marks as “String” literals. String can contain as many characters as one wish, there is no maximum string length restriction.

Strings are immutable in Java; that is, you cannot change the contents of a string, although you can redefine a **String** object variable.

For instance, as the string variable *message* is initially defined as

```
string message = “Hello Java”;
```

There is no way to change the contents of the object pointed to by *message* by, say, passing it to another method: *change(S)*;

```
//Impossible to change object pointed to by S
```

However, we can make the string variable *message* refer to a new string object;

```
message = message + “Language”;
```

Some useful methods provided by “String” class

Method	Description
boolean equals (String S)	Text two strings for equality
int Length()	Obtain the length of the String
char charAt (int index)	Obtain the character at a specified index within a String

Examples:

```
//Demonstrating some string class methods
```

```
Class StringDemo
```

```
{
```

```
    public static void main (String args[ ])
```

```
{
```

```
    String S1 = “First”;
```

```
    String S2 = “Second”;
```

```
String S3 = S1;
System.out.println("Length of S1: " + S1.Length());
System.out.println("char at index 2 in S2: " + S2.charAt(3));
System.out.print("Is S1 same as S2 : " + S1.equals(S2));
System.out.print("Is S1 equals to S3 : " + S1.equals(S3));
}
}
```

StringBuffer is a peer class of String that provides much of the functionality of strings. String represents fixed-length, immutable character sequences. In contrast, StringBuffer may have characters and substrings inserted in the middle or appended to the end. StringBuffer will automatically grow to make room for such additions and often has more characters preallocated than are actually needed, to allow room for growth.

StringBuffer defines the below three constructors :

```
StringBuffer()
StringBuffer(int size)
StringBuffer(String str)
```

Methods in this class :

```
length()
capacity()
setLength()
insert()
reverse()
replace() etc.
```

Q6. Discuss various operators available in Java.

Arithmetic Operators

Java's arithmetic operators are summarized in following table. These operators accept integer or floating-point operands and produce integer or floating-point results. The auto-increment and auto-decrement operators are included as arithmetic operators.

Arithmetic Operators

Operator	Purpose
<code>++, -</code>	Auto-increment, auto-decrement
<code>+, -</code>	Unary plus, unary minus
<code>*</code>	Multiplication
<code>/</code>	Division
<code>%</code>	Remainder (modulo division)
<code>+, -</code>	Addition, subtraction

The remainder operator (%) returns the remainder of dividing the first operand by the second, so 24 % 10 is the remainder left over after dividing 24 by 10, namely 4. This operator also works with floating-point operands.

Relational Operators

Relational Operators compare two quantities to determine if they are equal or if one is greater than the other. Java has kept C's question/colon, or conditional, operator that takes three operands (it is ternary operator). The first operand is boolean, and the two other operands may be of any type. If the boolean operand is true, the result is the second operand; if it is false, the result is the third operand:

```
boolean b;
int c;
b = true;
c = (b ? 1 : 2); //1 will be assigned to c because b is true
b = false;
c = (b ? 1 : 2); //2 will be assigned to c because b is false.
```

These comparison operators are usually used in conjunction with conditional statements. The relational operators are summarized in following table.

Relational Operators

Operator	Purpose
<code>>, <, >=, <=</code>	Tests relative magnitude
<code>==</code>	Tests equality
<code>!=</code>	Tests inequality
<code>?:</code>	Conditional-returns one of two operands based on a third

Boolean Operators or Logical Operators

Boolean operators act on boolean operands and return a boolean result. They implement the standard boolean algebraic operations:
AND, OR, NOT and XOR (eXclusive OR).

Boolean operators are shown in following table:

Boolean Operators

Operator	Purpose
!	NOT
&	Boolean AND
^	XOR
	boolean OR
&&	Conditional AND
	Conditional OR

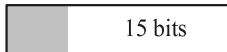
Bitwise Operators

The integral types (byte, short, int, and long) are represented in the computer's memory as a sequence of bits (binary digits). Just like decimal numbers, binary numbers have their most significant digits to the left. In decimal, the most significant digit in the number 325 is the 3 because it represents 300, and the least significant digit is the 5. Written as a binary number, 325 is 101000101. The leftmost digit represents 100000000 binary (256 decimal) and is the most significant digit. Java's integers are signed numbers, so Java must use the leftmost bit of storage to represent the sign of the integer. For Java's integer datatypes, the high bit is used to represent the sign of the number, as shown in the next figure.

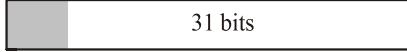
byte



short



int



long



Figure : Java's integral datatypes use the high bit to indicate the sign of the number.
If high bit is 1, the number is negative.

Java uses the same operators as the C language to manipulate the bits of integers. Because all of Java's integral datatypes are signed, Java supplements the C operators with an additional operator. Bit-manipulation operators are referred to as bitwise operators. The bitwise operators perform the same sort of functions as the boolean operators, as well as bit shifts.

The bitwise AND operator applies the AND operator to the corresponding bits of each operand. The bitwise OR, XOR and NOT operators work in a similar fashion. The bitwise operators are illustrated in following figure.

<table border="1" style="border-collapse: collapse; width: 100px; height: 20px;"> <tr><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr> </table> AND &	1	0	0	1	0	0	1	1	<table border="1" style="border-collapse: collapse; width: 100px; height: 20px;"> <tr><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr> </table> OR	1	0	0	1	0	0	1	1
1	0	0	1	0	0	1	1										
1	0	0	1	0	0	1	1										
<table border="1" style="border-collapse: collapse; width: 100px; height: 20px;"> <tr><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td></tr> </table> =	1	0	1	0	1	0	1	0	<table border="1" style="border-collapse: collapse; width: 100px; height: 20px;"> <tr><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td></tr> </table> =	1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	0										
1	0	1	0	1	0	1	0										
<table border="1" style="border-collapse: collapse; width: 100px; height: 20px;"> <tr><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr> </table> NOT ~	1	0	0	0	0	0	1	0	<table border="1" style="border-collapse: collapse; width: 100px; height: 20px;"> <tr><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr> </table> XOR ^	1	0	0	1	0	0	1	1
1	0	0	0	0	0	1	0										
1	0	0	1	0	0	1	1										
<table border="1" style="border-collapse: collapse; width: 100px; height: 20px;"> <tr><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td></tr> </table> =	0	1	1	0	1	1	0	0	<table border="1" style="border-collapse: collapse; width: 100px; height: 20px;"> <tr><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td></tr> </table> =	1	0	1	0	1	0	1	0
0	1	1	0	1	1	0	0										
1	0	1	0	1	0	1	0										
<table border="1" style="border-collapse: collapse; width: 100px; height: 20px;"> <tr><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td></tr> </table>	0	0	1	1	1	0	0	1	<table border="1" style="border-collapse: collapse; width: 100px; height: 20px;"> <tr><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td></tr> </table>	0	0	1	1	1	0	0	1
0	0	1	1	1	0	0	1										
0	0	1	1	1	0	0	1										

Figure : Boolean bitwise operators

The shift operators shift all the bits in an integral type to the left or the right, as shown in figure. The shift operators are binary operators. The second operand is an integer that determines the number of bits to shift. The standard C shift operators act slightly differently in Java. In Java, all numbers are signed, and the sign bit is preserved through all shifts. Even though this sign bit is shifted, it is also copied, so that the resulting number will have the same sign as the original. Java adds the >>> operator, which shifts all bits to the right as if the integer were unsigned.

<table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr> </table> <p style="text-align: center;">Right shift >>> 2 =</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td></tr> </table>	1	0	0	1	0	0	1	1	0	0	1	0	0	1	0	0	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr> </table> <p style="text-align: center;">Signed right-shift >> 1 =</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td></tr> </table>	1	0	0	1	0	0	1	1	1	1	0	0	1	0	0	1
1	0	0	1	0	0	1	1																										
0	0	1	0	0	1	0	0																										
1	0	0	1	0	0	1	1																										
1	1	0	0	1	0	0	1																										
<table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr> </table> <p style="text-align: center;">Signed left-shift << 2 =</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td></tr> </table>	1	0	0	1	0	0	1	1	1	1	0	0	1	1	0	0																	
1	0	0	1	0	0	1	1																										
1	1	0	0	1	1	0	0																										

Figure : Examples of Java's bit-shifting operators

Bitwise Operators

Operator	Purpose
~	NOT (bitwise complement)
<<, >>	Left shift, Right shift
>>>	Right shift as if unsigned
&	Bitwise AND
^	Bitwise XOR
	Bitwise OR

Assignment Operators

The assignment operator is an operator that assigns the second operand to the first and returns the second operand as a result.

Other assignment operators serve as shorthand for combined operation and assignment. If you wanted to add 5 to a number, you could write:

```
int i;
i = i + 5;
```

You could also use the assignment operator +=:

```
int i;
i += 5; //this is the same as i = i + 5
```

Assignment operators exist for most of the operators already mentioned. They are summarized in following table.

Assignment Operators

Operator	Purpose
=	Assignment
*=	Assignment with operation
/=	Assignment with operation
%=	Assignment with operation
+=	Assignment with operation
-=	Assignment with operation
>>=	Assignment with operation
<<=	Assignment with operation
>>>=	Assignment with operation
=	Assignment with operation
&=	Assignment with operation

Q7. Define Variables. How to declare variable ?

Variable : The variable is the base unit of storage in a Java program. A variable is defined by the combination of an identifier, a type, and an optional initializer. In addition, all variables have a scope, which defines their visibility, and a lifetime.

Declaring a Variable :

In Java, all variables must be declared before they can be used. The basic form of a variable declaration is shown here:

type identifier [= value][, identifier [= value] ...];

The type is one of Java's atomic types, or the name of a class or interface. The identifier is the name of the variable. You can initialize the variable by specifying an equal sign and a value. Keep in mind that the initialization expression must result in a value of the same (or compatible) type as that specified for the variable. To declare more than one variable of the specified type, use a comma-separated list.

Here are several examples of variable declarations of various types. Note that some include an initialization.

```
int a, b, c;           // declares three ints, a, b, and c.
int d = 3, e, f = 5;   // declares three more ints, initializing
                      // d and f.
byte z = 22;          //initializes z.
double pi = 3.14159;  // declares an approximation of pi.
```

```
char x = 'x';           // the variable x has the value 'x'.
```

The identifiers that you choose have nothing intrinsic in their names that indicates their type.

Dynamic Initialization

Java allows variables to be initialized dynamically, using any expression valid at the time the variable is declared.

For example, here is a short program that computes the length of the hypotenuse of a right triangle give the lengths of its two opposing sides:

```
// Demonstrate dynamic initialization.
```

```
class DynInit {
    public static void main(String args []) {
        double a = 3.0, b = 4.0;
        // c is dynamically initialized
        double c = Math.sqrt(a * a + b * b);
        System.out.println("Hypotenuse is " + c);
    }
}
```

Here, three local variables-a, b, and c are declared. The first two, a and b, are initialized by constants. However, c is initialized dynamically to the length of the hypotenuse (using the Pythagorean theorem). The program uses another of Java's built-in methods, sqrt(), which is a member of the Math class, to compute the square root of its argument. The key point here is that the initialization expression may use any element valid at the time of the initialization calls to methods, other variables, or literals.

Q8. Discuss in detail “type casting”.

What is typecasting ? Give two examples of typecasting in Java.

Type Casting Rules :

When one type of data is assigned to another type of variable, an automatic type conversion will take place if the following two conditions are met:

- The two types are compatible.
- The destination type is larger than the source type.

When these two conditions are met, a widening conversion takes place. For example, the int type is always large enough to hold all valid byte values, so an explicit cast statement is required. For widening conversions, the numeric types, including integer and floating-point types, are compatible with each other. However, the numeric types are not compatible with char or boolean. Also, char and boolean are not compatible with each other.

As mentioned earlier, Java also performs an automatic type conversion when storing a literal integer constant into variables of byte, short, or long.

Casting Incompatible Types :

Although the automatic type conversions are helpful, they will not fulfill all needs. For example, what if you want to assign an int value to a byte variable? This conversion will not be performed automatically, because a byte is smaller than an int. This kind of conversion is sometimes called a narrowing conversion, since you are explicitly making the value narrower so that it will fit into the target type.

To create a conversion between two incompatible types, you must use a cast. A cast is simply an explicit type conversion. It has this general form:

(target-type) value

Here, target-type specifies the desired type to convert the specified value to. For example, the following fragment casts an int to a byte. If the integer's value is larger than the range of a byte, it will be reduced modulo (the remainder of an integer division by) the byte's range.

```
int a;
byte b;
// ...
b = (byte)a;
```

A different type of conversion will occur when a floating-point value is assigned to an integer type: truncation. As you know, integers do not have fractional component. For example, if the value 1.23 is assigned to an integer, the resulting value will simply be 1. The 0.23 will have been truncated. Of course, if the size of the whole number component is large to fit in the target integer type, then that value will be reduced modulo to the target type's range.

The following program demonstrates some type conversions that require casts:

```
// Demonstrate casts.
```

```
class Conversion {
    public static void main(String args[] ) {
        byte b;
        int i = 257;
        double d = 323.142;
        System.out.println("\nConversion of int to byte.");
        b = (byte)i;
        System.out.println("i and b " + i + " " + b);
        System.out.println("\nConversion of double to int.");
        i = (int)d;
        System.out.println("d and i " + d + " " + i);
        System.out.println("\nConversion of double to byte.");
        b = (byte)d;
        System.out.println("d and b " + d + " " + b);
    }
}
```

```
}
```

Conversion of int to byte.

i and b 257	1
-------------	---

Conversion of double to int.

d and i 323.142	323
-----------------	-----

Conversion of double to byte.

d and b 323.142	67
-----------------	----

Let's look at each conversion. When the value 257 is cast into a byte variable, the result is the remainder of the division of 257 by 256 (the range of a byte), which is 1 in this case. When the d is converted to an int, its fractional component is lost. When d is converted to a byte, its fractional component is lost, and the value is reduced modulo 256, which in this case is 67.

Automatic Type Promotion

In addition to assignments, there is another place where certain type of conversions may occur: To see why, consider the following. In an expression, the precision required of an intermediate value will sometimes exceed the range of either operand. For example, examine the following expression:

```
byte a = 40;
byte b = 50;
byte c = 100;
int d = a * b / c;
```

The result of the intermediate term a * b easily exceeds the range of either of its byte operands. To handle this kind of problem, Java automatically promotes each byte or short operand to int when evaluating an expression. This means that the subexpression a * b is performed using integers-not bytes. Thus, 2,000, the result of the intermediate expression, 50 * 40, is legal even though a and b are both specified as type byte.

As useful as the automatic promotions are, they can cause confusing compile-time errors. For example, this seemingly correct code causes a problem:

```
byte b = 50;
b = b*2; // Error! Cannot assign an int to a byte!
```

The code is attempting to store 50 * 2, a perfectly valid byte value, back into a byte variable. However, because the operands were automatically promoted to int when the expression was evaluated, the result has also been promoted to int. Thus, the result of the expression is now of type int, which cannot be assigned to a byte without the use of a cast. This is true even if, as in this particular case, the value being assigned would still fit in the target type.

In cases where you understand the consequences of overflow, you should use an explicit cast, such as

```
byte b = 50;
```

`b = (byte) (b*2);`
 which yields the correct value of 100.

Q9. Write some programs showing Java's Variable Declaration and use of literals.

```
class testVar
{
    public static void main(String args[])
    {
        int a;      //Declaring an int type variable that store
                    //int type of data
        char b;    //Declaring an char type variable that store
                    //char type of data
        float c;   //Declaring an float type variable that store
                    //float type of data

        a=10;
        b='a';
        c=10.2f;
        System.out.println(a + " , " + b + " , " + c);
        a=20;
        b='b';
        c=13.2f;
        System.out.println(a + " , " + b + " , " + c);
        ++a;
        ++b;      //Integer value of char 'b' is incremented by 1
        ++c;
        System.out.println(a + " , " + b + " , " + c);
    }
}

/*****class chardemo
{
    public static void main(String args[])
    {
        char ch1,ch2;
        ch1=88;
        ch2='Y';
        System.out.println(ch1 +" "+ ch2);
    }
}
```

```
*****  
class testfloatVar  
{  
    public static void main(String args[])  
    {  
        float pi=3.14f;  
        double area;  
        float radius=9;  
        area=pi*radius*radius;  
        System.out.println("Area of the circle : " + area);  
    }  
}
```

In the given example to initialize a float variable ‘f’ is added at end of float literal, also, we achieve the same by adding ‘(float)’ before the literal.

```
*****  
class booltest  
{  
    public static void main(String args[])  
    {  
        boolean b;  
        b=false;  
        if(b)      // same as if(b==true)  
            System.out.println("b is true");  
        else  
            System.out.println("b is false");  
    }  
}
```

Q10. Define array. Write short note on One Dimensional Arrays.

An array is a group of variables of same type that can be referred to by a common name. The type can be either primitive datatype (int, short, char, boolean), or an object type like “String.”

One Dimensional Array :

To create an array, we must create an array of desired type. The general form of declaring an one dimensional array is

<data type> <variable name>[];

For example consider these arrays declared differently and valid in Java.

```
int month_days[ ];
int[ ] month_days;
```

Declaring an array variable creates only a place holder for the reference to the array, not the memory holding the array of elements. Also, the actual memory used by the array elements is dynamically allocated either by a `|new|` statement or an array initializer. The memory referenced by the array variable will be automatically garbage - collected (discussed later) when it is no longer referred to. The general form of `|new|` as it applies to one-dimensional arrays appears as follows:

```
<variable name>= new <datatype> [size];
```

For example consider these arrays

```
month_days = new int[ ];
month_days = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
int month_days [] = new int [12];
int month_days [] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

Java strictly checks to make sure that one can not accidentally try to store or reference values outside of the range of the array.

Q11. What happens when your program attempts to access an array element with an invalid index ?

Java follows normal C - style indexing for accessing an array element that is, you attach an integer - valued expression between square brackets after the name of the array. The array index starts with zero.

If the index value is invalid (less than zero or greater than array length), the Java Run Time throws the exception `ArrayIndexOutOfBoundsException`. Which can be handled through the exception handling facility provided in Java for these kind of situations.

Q12. Define Multi-dimensional Arrays.

In Java, multidimensional arrays are actually arrays of arrays. These, as you might expect, look and act like regular multidimensional arrays. However, as you will see, there are a couple of subtle differences. To declare a multidimensional array variable, specify each additional index using another set of square brackets. For example, the following declares a two-dimensional array variable called `twoD`.

```
int twoD[ ][ ] = new int [4] [5];
```

This allocates a 4 by 5 array and assigns it to `twoD`. Internally this matrix is implemented as an array of arrays of `int`. Conceptually, this array will look like the one shown figure.

a ₀₀	a ₀₁	a ₀₂	a ₀₃	a ₀₄
a ₁₀	a ₁₁	a ₁₂	a ₁₃	a ₁₄
a ₂₀	a ₂₁	a ₂₂	a ₂₃	a ₂₄
a ₃₀	a ₃₁	a ₃₂	a ₃₃	a ₃₄

To initialize a multi dimensional array without using *new*.

```
int matrix [4][2] = {{1,2}, {3,4}, {5,6}, {7,8}};
```

Q13. Write a Program for Arithmetic Operations IN JAVA.

```
class testArith_Operator
```

```
{
```

```
    public static void main(String args[])
```

```
{
```

```
    int x=6;
```

```
    int y=2;
```

```
    System.out.println("The value of x + y is "+ (x + y));
```

```
    System.out.println("The value of x - y is "+ (x - y));
```

/* The parenthesis is necessary so that compiler treats the + sign as an arithmetic operator not string concatenate as in the following statement.

```
System.out.println("the value of x + y is "+ x + y);
```

This will print "the value of x + y is 62"

(NOTE: This is Polymorphism) */

```
    System.out.println("The remainder of x / y is "+ x%y);
```

```
    System.out.println("The value of x / y is "+ x/y );
```

```
    System.out.println("The value of x * y is "+ x*y );
```

```
}
```

```
}
```

Q14. Write a program for Java supporting two types of data conversion.

```
class conv
```

```
{
```

```
    public static void main(String a[])
```

```
{
```

```
    int x=128;
```

```
    byte b=(byte) x;      /*Explicit Conversion*/
```

```
    long l=x;            /*Implicit Conversion*/
```

```
        System.out.println("b = " + b);
    }
}
```

Q15. Write a Program showing use of two dimensional array to store a matrix of 2×2 .

```
class d2array
{
    public static void main (String args[])
    {
        int a[][];
        a=new int[2][2];
        a[0][0]=0;
        a[0][1]=1;
        a[1][0]=2;
        a[1][1]=3;
        System.out.println(a[0][0]);
        System.out.println(a[0][1]);
        System.out.println(a[1][0]);
        System.out.println(a[1][1]);
    }
}
```

**Subscribe to our FREE
Newsletter**

Visit www.Gullybaba.com

Get site news, exam tips, guess paper & other hard to find information & tips.

CHAPTER 4

JAVA CONTROL STATEMENTS

Condition

- a. Use of if condition
- b. If
- c. Nested if
- d. Multiple if
- e. Switch

Condition control statement

The If statement

The syntax of the if statement is as follows:

if(expression)

 statement;

The expression must be enclosed in parenthesis. The statement may be a compound statement.

for example:

if ($y > 5$)

 x=0;

The statement will be executed only if the expression is true. The syntax of the if else statement is as follows:

if(expression)

 statement-1;

else

 statement-2;

If expression is true statement –1 is executed. If expression is false, statement –2 is executed. So depending on the truth value of expression, either statement-1 or statement 2 is executed,

Nested if statements

The statement sequence of if or else may contain another statement i.e. the if else statement can be nested within one another as shown below:

Syntax of nested if statement

if (expression-1)

```

{
    if(expression-2)
{
    statement;
}
else
    if(expression-3)
    {
    statement;
    }
    else
{
    statement;
}
}

```

There may be one problem raised with nested if that is to decide “which if does the else match”.

F Each else matches to its nearest unmatched preceding if.

For example

```

if(a<10)
{
    if(b>5)
        Sum=Sum + a + b;
    else
        Sum=Sum - a - b ;

```

Multiple if statement

Syntax of multiple if statement.

```

if(expression-1)
    statement -1;
else if (expression -2)
    statement -2;
else if(expression -3)
    statement-3;
..
..
else if(expression -n)
    statement-n;
else
    default statement;

```

the expressions are evaluated in order and if any condition is true then the

statement associated with it is executed and this terminates the whole chain. The last else part is executed as default statement.

The switch statement

If you have a long series of conditions to check, a large block of if...else statements can get tiresome and confusing to write (and read, later on). For this reason, Java provides the switch statement. The general syntax is:

```
switch ( expression )
```

```
{
```

```
    case value1:
```

```
        statements
```

```
        [break;]
```

```
    case value 2:
```

```
        statements
```

```
        [break:]
```

```
....
```

```
default:
```

```
    Statements
```

```
}
```

An expression is evaluated in the first line of a switch statement, and its value is used to determine the action or actions to take. The value of the expression is compared in turn with each value listed next to a case statement. If a match occurs, the statements following that case statement are executed. The break statement (shown in square brackets to indicate that it is optional; the square brackets are not actually part of the code), if present, causes the switch statement to be terminated after the statements in the current case block are executed, that is, control will pass to the statement immediately after the final brace ending the switch statement without any further case statements being tested, if the break statement is not present, the next case statement will be checked as well. In other words, a switch statement offers greater flexibility than an if.. Else statement.

Iteration Statements

Java's iteration statements are For, while, and do-while. These statements create what we commonly call loops. As you probably know, a loop repeatedly executes the same set of instructions until a termination condition is met.

While

The **while** loop is Java's most fundamental looping statement. It repeats a statement or block while its controlling expression is true. Here is its general form:

```
While(condition)           // body of loop
}
```

The condition can be any Boolean expression. The body of the loop will be executed as long as the conditional expression is true. When condition becomes false, control passes to the next line of code immediately following the loop. The curly braces are unnecessary if only a single statement is being repeated. Since the **while** loop evaluates its conditional expression at the top of the loop, the body of the loop will not execute even once if the condition is false to begin with. For example, in the following fragment, the call to **println()** is never executed:

```
int a = 10, b = 20;
while (a > b)
    System.out.println("This will not be displayed");
```

do-while

As you just saw, if the conditional expression controlling a **while** loop is initially false, then the body of the loop will not be executed at all. Sometimes it is desirable to execute the body of a **while** loop at least once, even if the conditional expression is false to begin with. The **do-while** loop always executes its body at least once, because its conditional expression is at the bottom of the loop.

Its general form is

```
do {
    // body of loop
}
```

} while (condition);

Each iteration of the **do-while** loop first executes the body of the loop and then evaluates the conditional expression. If this expression is true, the loop will repeat. Otherwise, the loop terminates. As with all of Java's loops, condition must be a Boolean expression.

for

The general form of the for statement:

```
for (initialization; condition; iteration) {
    //body
}
```

If only one statement is being repeated, there is no need for the curly braces. The **for** loop operates as follows. When the loop first starts, the initialization portion of the loop is executed. Generally, this is an expression that sets the value of the loop control variable, which acts as a counter that controls the loop. It is important to understand that the initialization expression is only executed once. Next, condition is evaluated. This must be a Boolean expression. It usually tests the loop control variable against a target value. If this expression is true, then the body of the loop is executed. If it is false, the loop terminates. Next, the iteration portion of the loop is executed. This is usually an expression that increments or decrements the loop control variable. The loop then iterates,

first evaluating the conditional expression, then executing the body of the loop, and then executing the iteration expression with each pass. This process repeats until the controlling expression is false.

Nested Loops

Like all other programming languages, Java allows loops to be nested. That is, one loop may be inside another. For example, here is a program that nests for loops:

//Loops may be nested.

```
class Nested {  
    public static void main(String args[ ] ) {  
        int i, j;  
        for(i=0; i<10; i++ ) {  
            for(j=i; j<10; j++)  
                System.out.print(“.”);  
            System.out.println( );  
        }  
    }  
}
```

The **output** produced by this program is shown here:

.....

Break and Labelled Break commands.

By using **break**, you can force immediate termination of a loop, bypassing the conditional expression and any remaining code in the body of the loop. When a **break** statement is encountered inside a loop, the loop is terminated and program control resumes at the next statement following the loop.

Here are two points to remember about **break**. First, more than one **break** statement may appear in a loop. However, be careful. Too many **break** statements have the tendency to destructure your code. Second, the **break** that terminates a switch statement affects only that **switch** statement and not any enclosing loops.

Labelled Break :

Sometimes you'd like more flexibility, for example the ability to break out two levels.

In Java, there's a simpler approach to solving this problem as :

```
outer_block:
    for (i = 1; i <= 10; i++) {
        for (j = 1; j <= 10; j++) {
            if (some condition)
                break outer_block;
            // other processing in loop
        }
    }
    // control comes here when break is executed
```

The break statement has a label. When the statement is executed, control transfers out of the enclosed labelled statement. Control is NOT transferred to the label, but to a point past the end of the labelled statement. A labelled continue statement works in a similar way; control is transferred to the end of a labelled loop.

Note that with both break and continue, any finally block will be executed before the labelled statement is exited or the next iteration of a loop takes place. For example:

```
loop:
    while (some condition) {
        try {
            if (some condition)
                continue loop;
            // other processing
        }
        finally {
            // some processing
        }
    }
```

The labelled continue statement will cause the finally block to be executed before going to the bottom of the while loop in preparation for the next iteration of the loop.

Continue

Sometimes it is useful to force an early iteration of a loop. That is, you might want to continue running the loop, but stop processing the remainder of the code in its body for this particular iteration. This is, in effect, a goto pass the body of the loop, to the loop's end. In **while** and **do-while** loops, a **continue**

statement causes control to be transferred directly to the conditional expression that controls the loop. In a **for** loop, control goes first to the iteration portion of the for statement and then to the conditional expression. For all three loops, any intermediate code is bypassed.

Here is an example program that uses continue to cause two numbers to be printed on each line:

```
// Demonstrate continue.
Class continue {
    public static void main(String args[]) {
        for(int i=0; i<10; i++) {
            System.out.print (i + " ");
            if(i % 2 == 0) continue;
            System.out.print In (" ");
        }
    }
}
```

This code uses the **%** operator to check if *i* is even. If it is, the loop continues without printing a newline.

IF Programs

SIMPLE IF ELSE (Single Statement)

```
class simple_if
{
    public static void main(String args[])
    {
        int x=5,y=7;
        if(x<y)
            System.out.println("y is max.");
        else
            System.out.println("x is max.");
    }
} ****
```

IF ELSE BLOCKS (Multiple Statements in a if block)

```
class EvenOdd
{
    public static void main(String args[])
    {
        int x=5,y=8;
        if(y%2==0)
        {
            System.out.println("y is even");
        }
    }
}
```

```

        x=y;
    }
else
{
    System.out.println("y is odd");
    x=y;
}
}

/*****

```

Nested If Statement

The Statement part of “if” contains another if. This example finds the greatest valued integer in a, b and c.

```

class nested_if
{
    public static void main(String args[])
    {
        int a=34, b=23, c=332;
        if(a > b)
        {
            if (a>c)
                System.out.println("a is greatest");
            else
                System.out.println("c is greatest");
        }
        else
        {
            if (b>c)
                System.out.println("b is greatest");
            else
                System.out.println("c is greatest");
        }
    }
}

```

Program showing use of switch for welcoming a user.

```

class gender
{
    public static void main(String args[])
    {
        char gender='f';

```

```
String msg;
switch(gender)
{
    case 'm': msg="Hello Mister";
                break;
    case 'f': msg="Hello Miss";
                break;
    default:
        msg="Hello Alien";
}
System.out.print(msg);
}
```

A seasons program (case grouping).

```
class Switch
{
    public static void main(String args[])
    {
        int month=4;
        String season;

        switch(month)
        {
            case 12:
            case 1:
            case 2:   season="winter";
                        break;
            case 3:
            case 4:
            case 5:   season="spring";
                        break;
            case 6:
            case 7:
            case 8:   season="summer";
                        break;
            case 9:
            case 10:
            case 11:  season="autumn";
                        break;
            default:
        }
    }
}
```

```

        season="bogus month";
    }
    System.out.println("april is in the " +season+ ".");
}
}

Find my religious book.
class holybook
{
    public static void main(String args[])
    {
        String holybook;
        char religion='h';

        switch (religion)
        {
            case 'h': holybook="Srimad Bhagwat Gita";
                        break;
            case 'p': holybook="Zenda Avista";
                        break;
            case 's': holybook="Sri Guru Granth Sahib Ji";
                        break;
            case 'm': holybook="Quran";
                        break;
            case 'c': holybook="Bible";

            default: holybook="God Shiva Lord Krishna";
        }
        System.out.print("The holybook chosen is " + holybook);
    }
}

```

A simple for loop demo.

```

class Simple_For
{
    public static void main(String a[])
    {
        int i;
        //Print the table of 2
        for (i=1;i<=20;i=2*i)
            System.out.print(i);
    }
}
```

```
}
```

Program to print first 6 numbers of a fabonacci series.

```
class fabonacii
{
    public static void main(String args[])
    {
        int first=1,second=2;
        int temp;
        //Print 1 and 2 as they are the starting values
        System.out.println(first);
        System.out.println(second);
        for(int i=0;i<=6;i++)
        {
            temp=second;
            second=first+second;
            first=temp;
            System.out.println(second);
        }
        System.out.print("\n");
    }
}
```

Program to find binary equivalent of a decimal value.

```
class dec
{
    public static void main(String args[])
    {
        int index=0;
        int dec=132;
        int d[]={};
        for(int i=dec;i>=1;)
        {
            //Store each bit in array
            d[index]=i%2;
            index++;
            i=i/2;
        }
        System.out.print("the binary of " + dec + " is ");
        //Print the array in reverse fashion
        for(int i=index-1;i>=0;i--)
    }
```

```

    {
        System.out.print(d[i]);
    }
    System.out.print("\n");
}
}

```

Program to find maximum value and its index position in array.

class findMAX

```

{
    public static void main(String args[])
    {
        int a[]={12,32,75,8,182,946,234,234,123,234,76};
        int max=a[0], pos=0;

        for (int i=1;i<=a.length-1;i++)
        {
            if (max > a[i])
            {
                continue;
            }
            else
            {
                max=a[i];
                pos=i;
            }
        }
        System.out.print("Max Value is " +max+ " at position "+ pos);
    }
}

```

Program to find Max and Min in an Array.

class findMAX_MIN

```

{
    public static void main(String args[])
    {
        int
a[]={40,52,75,65,72,75,24,64,73,75,71,52,75,65,72,75,24,64,73,75,71};
        int max=a[0];
        int min=a[0];
        int pmax=0;
    }
}

```

```

int pmin=0;
for(int i=1;i<=a.length-1;i++)
{
    if(max<a[i])
    {
        max=a[i];
        pmax=i;
    }
    if(min>a[i])
    {
        min=a[i];
        pmin=i;
    }
}
System.out.println("MAX=( " + max + ", "+pmax+") , MIN=( " + min + ", " +
pmin +")");
}
}

```

Program to search an array for a given value.

```

class search_array
{
    public static void main(String gf[])
    {
        int a[] = new int[10];
        int s=24,i;
        boolean found=false;
                                            //fill the array with a table of 3
        for(i=0;i<=9;i++)
            a[i]=i*3;
                                            //Searching the array for s=24
        for(i=0;i<=9;i++)
        {
            if (a[i]==s)
            {
                found=true;
                break;
            }
        }
        if (found)

```

```
        {
            System.out.println("The search value " + s + " found at position " +
(i+1));
        }
    else
    {
        System.out.println("The search value not found");
    }
}
```

Program to sort an array using Insertion Sort.

```
class INSERTION_SORT
{
    public static void main(String args[])
    {
        int a[]={12,32,75,8,182};           //,946,234,234,123,234,76};
        int i,j,temp;
        for(i=0;i<=a.length-1;i++)
        {
            for(j=i;j<=a.length-1;j++)
            {
                if(a[i]>a[j])
                {
                    temp=a[i];
                    a[i]=a[j];
                    a[j]=temp;
                }
            }
        }
        System.out.println("Array sorted in descending order");
        for(i=0;i<=a.length-1;i++)
        System.out.println(a[i]);
    }
}
```

Q1. Write a Program demonstrating the use of 2-D arrays in matrices calculations.

ADDITION or SUBTRACTION

class mat add

{

```
public static void main(String args[])
{
    int a[][]=new int[3][3];
    int b[][]=new int[3][3];
                                //Fill Matrix a
    for(int i=0;i<3;i++)
    {
        for(int j=0;j<3;j++)
        {
            a[i][j]=3*(j+1);
            System.out.println(a[i][j]);
        }
    }
                                //Fill Matrix b
    for(int i=0;i<3;i++)
    {
        for(int j=0;j<3;j++)
        {
            b[i][j]=4*(j+1);
        }
    }
                                //Calculate c=a+b or c=a-b
    for(int i=0;i<3;i++)
    {
        for(int j=0;j<3;j++)
        {
            a[i][j]=a[i][j]+b[i][j];
        }
    }
                                //Print the resultant matrix
    for(int i=0;i<3;i++)
    {
        for(int j=0;j<3;j++)
        {
            System.out.print("\t" + a[i][j]);
        }
        System.out.print("\n");
    }
}
```

MATRIX MULTIPLICATION

```
class mat_mult  
{
```

```
    public static void main(String args[])
```

```
    {
```

```
        int m=2,n=4,p=3;
```

```
        int a[][]=new int[m][n];
```

```
        int b[][]=new int[n][p];
```

```
        int c[][]=new int[m][p];
```

```
        a[0][0]=1;
```

```
        a[0][1]=1;
```

```
        a[0][2]=1;
```

```
        a[0][3]=1;
```

```
        a[1][0]=1;
```

```
        a[1][1]=1;
```

```
        a[1][2]=1;
```

```
        a[1][3]=1;
```

```
        b[0][0]=2;
```

```
        b[0][1]=2;
```

```
        b[0][2]=2;
```

```
        b[1][0]=2;
```

```
        b[1][1]=2;
```

```
        b[1][2]=2;
```

```
        b[2][0]=2;
```

```
        b[2][1]=2;
```

```
        b[2][2]=2;
```

```
        b[3][0]=2;
```

```
        b[3][1]=2;
```

```
        b[3][2]=2;
```

// Matrix Multiplication

```
    for(int i=0;i<=m-1;i++)
```

```
    {
```

```
        for(int j=0;j<=n-1;j++)
```

```
        {
```

```
            for(int k=0;k<=p-1;k++)
```

```
            {
```

```
                c[i][k]+=a[i][j]*b[j][k];
```

```
            }
```

```
        }
```

```
}
```

// Printing Resultant Matrix

```
for(int i=0;i<=m-1;i++)  
{  
    for(int j=0;j<=p-1;j++)  
    {  
        System.out.print("\t"+ c[i][j]);  
    }  
    System.out.print("\n");  
}  
}  
}
```

Q2. Why should switch statement be avoided?

Sometimes it is easy to fall through accidentally to an unwanted case while using switch statement. It is advisable to use if/else instead.

**Earn While You Learn
Become a Brand Ambassador
for 'Gullybaba'
Visit www.Gullybaba.com**

This is amazing opportunity to earn, while helping your friends to realize their dream. Become a BRAND AMBASSADOR

FREE Downloads

Visit www.Gullybaba.com

Download Firefox software absolutely free. Also many study related material.

CHAPTER 5

INTRODUCING CLASSES, CONSTRUCTORS & OBJECTS IN JAVA

Class Fundamentals

A class is a way to bind the data and its associated methods (functions) together. It allows the data & methods to be hidden, if necessary, from external use. When we define a class, we are creating a new data type that can be treated like any other data type of Java. Once defined, this new type can be used to create object of that type. Thus, a class is a *template* for an object, and an object is an instance of a class.

The general form of a *class* definition:

```
class <classname>
{
    type <instance_variable1>;
    type <instance_variable2>;
    type methodname1(parameter_list)
    {
        // body of Method
    }
    type methodname2 (parameter_list)
    {
        // body of Method
    }
}
```

A class can have any number of instance variable or methods. A typical class declaration would look like:

```
class Item
{
    int number;
    float cost;
    void getData(int a, float b)
    {
        number = a;
        cost = b;
    }
}
```

```

}
void printData()
{
    System.out.println("Total Items: " + number);
    System.out.println("cost per Item: " + cost);
}
}

```

We usually give a class some meaningful name, such as item. This “item” becomes a new type identifier that can be used to declare *instances* of that class type:

The method “getData()” can be used to assign values to the instance variables number and cost, and printData() for displaying their values.

The figure shows notations used by the OOP analysts to represent a class.

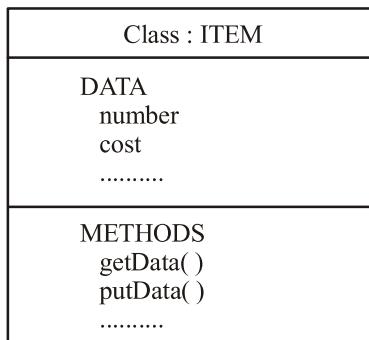


Figure : Representation of a class

Creating objects

As a class defines a new type of data, we use the class name to declare objects of that class. It is important to note that a class declaration only creates a template; it does not create an object.

To actually create an object, we use the statement like the following.

Item myItem=newItem();

After this statement executes, myItem will be an instance of Item. Thus, it will have physical reality.

We can create as many objects of a class as we like, and each object contains its own copy of instance variables defined by the class.

To access an instance variable, use dot (.) operator. The dot operator links the name of the object with the name of an instance variable.

myItem.Number = 100;

Similarly, we can also call the methods of an object

myItem.GetData(5, 13.5);

Since, the getData() has a void return type, the above statement works, but if

suppose it has an integer return type than we must use an integer type acceptor for method `getData()` and the statement looks like:

```
int X;  
N=myItem.GetData(5,13.5);
```

Q1. Difference between Constructor and Method.

The new operator dynamically allocates memory for an object. It has this general form:

```
Class-var = new classname();
```

Here, class-var is a variable of the class type being created. The classname is the name of the class that is being instantiated.

The class name followed by parentheses specifies the **constructor** for the class. A constructor defines what occurs when an object of a class is created. Constructors are an important part of all classes and have many significant attributes. Most real-world classes explicitly define their own constructors within their class definition. However, if no explicit constructor is specified, then Java will automatically supply a default constructor.

Methods

This is the general form of a method:

```
Type name(parameter-list) {  
    //body of method  
}
```

Here, type specifies the type of data returned by the method. This can be any valid type, including class types that you create. If the method does not return a value, its return type must be void. The name of the method is specified by name. This can be any legal identifier other than those already used by other items within the current scope. The parameter-list is a sequence of type and identifier pairs separated by commas. Parameters are essentially variables that receive the value of the arguments passed to the method when it is called. If the method has no parameters, then the parameter list will be empty.

Methods that have a return type other than **void** return a value to the calling routine using the following form of the **return** statement:

```
return value;
```

Here, value is the value returned.

Understanding Return

The implementation of `putData()` simply displays the total number of items and cost of each item. How about having another method `itemValue()`, which provides the total value of Item object.

Now, Item class looks like:

```
class Item
```

```
{  
int number;  
float cost;  
void getData(int a, float b)  
{  
number = a;  
cost = b;  
}  
void putData( )  
{  
System.out.println("Total item: " + number + "cost: " + cost);  
}  
float itemvalue()  
{  
return number cost;  
}  
}  
}  
class ItemDemo {  
public static void main ( String args [ ] )  
{  
float value;  
Item myItem = newItem();  
myItem.getData(10,12.0);  
myItem.putData();  
Value = myItem.itemvalue ( ) ;  
System.out.println( " Total stock held is " + value + " Rs");  
}  
}
```

Methods that take Parameters

We already have method `getData()` in class `Item` which has two parameters `a` and `b` of type integer & float respectively. Note that whenever in our previous programs we call `getData()` we always supply the value of matching type as parameters declared.

The values passed in the `getData()` in `main()` method are called the arguments.

Q2. What is constructor ? How can you overload constructor ? Describe with an example.

Describe Constructors.

It can be tedious to initialize all of the variables in a class each time an instance is created. Even when you add convenience function like `getdata()`, it would be simplest and more concise to have all of the setup done at the time the object is first created. Because the requirement for initialization is so common, Java allows objects to initialize themselves when they are created. This automatic initialization is performed through the use of a constructor.

A constructor initializes an object immediately upon creation. It has the same name as the class, in which it resides and is syntactically similar to a method. Once defined, the constructor is automatically called immediately after the object is created, before the `new` operator completes. Constructors look a little strange because they have not return type, not even `void`. This is because the implicit return type of a class's constructor is the class type itself. It is the constructor's job to initialize the internal state of an object so that the code creating an instance will have a fully initialized, usable object immediately.

Let's how add a constructor in Item template so the values of variable automatically initialized when an object is constructed.

```
Item( )
{
    Number = 10;
    Cost = 120;
}
```

The constructor defined above in class Item. Set the instance variable number and cost for each object to the same values. However, there are cases when we want to start an object with different set of values. For this we need to add parameters to the constructor Item

```
Item (int num, cost cost_item)
{
    number = num;
    cost = cost_item;
}
```

Since constructors are also a type of method with some special authorities, so Java allows overloaded constructors.

The program shows three overloaded constructors for class shape.

```
//Overloaded Constructor
```

```
class Shape
{
    int x;
    int y;
    int z;
```

```

Shape(int i,int j,int k)
{
    x=i;
    y=j;
    z=k;
}
Shape(int i)
{
    x=y=z=i;
}

Shape()
{
    x=y=z=-1;
}
}

class mainClass
{
    public static void main(String pa[])
    {
        //Unknown shape:: Default Constructor
        Shape b1=new Shape();
        System.out.println(b1.x + " , " + b1.y + " , " + b1.z);

        //Shape is a Cube or Sphere::Constructor with One Parameter
        Shape b2=new Shape(12);
        System.out.println(b2.x + " , " + b2.y + " , " + b2.z);

        //Shape is a Cuboid::Constructor with three Parameter
        Shape b3=new Shape(12,23,2);
        System.out.println(b3.x + " , " + b3.y + " , " + b3.z);
    }
}

```

Q3. Define the Methods Overloading and Overriding a method.

#: What is method overloading ? Explain how a method is overloaded in Java, with an example. [June-06, Q5(d)]

#: What is method overloading? What are the important points which should be taken care of while overloading methods? Write a Java program to explain the working of overloaded methods. [Dec-05, Q2(b)]

Methods Overloading : When an overloaded method is invoked, Java uses the type and /or number of arguments as its guide to determine which version of the overloaded method to actually call. Thus, overloaded methods must differ in the type and / or number of their parameters. While overloaded methods may have different return types, the return type alone is insufficient to distinguish two versions of a method. When Java encounters a call to an overloaded method, it simply executes the version of the method whose parameters match the arguments used in the call.

Since constructors are also a type of method with some special authorities, so Java allows overloaded constructors.

The program shows three overloaded constructors for class shape.

```
//Overloaded Constructor
```

```
class Shape
```

```
{
```

```
    int x;
```

```
    int y;
```

```
    int z;
```

```
Shape(int i,int j,int k)
```

```
{
```

```
    x=i;
```

```
    y=j;
```

```
    z=k;
```

```
}
```

```
Shape(int i)
```

```
{
```

```
    x=y=z=i;
```

```
}
```

```
Shape()
```

```
{
```

```
    x=y=z=-1;
```

```
}
```

```
}
```

```
class mainClass
```

```
{
```

```
    public static void main(String pa[])
```

```
{  
//Unknown shape:: Default Constructor  
Shape b1=new Shape();  
System.out.println(b1.x + " , " + b1.y + " , " + b1.z);  
  
//Shape is a Cube or Sphere::Constructor with One Parameter  
Shape b2=new Shape(12);  
System.out.println(b2.x + " , " + b2.y + " , " + b2.z);  
  
//Shape is a Cuboid::Constructor with three Parameter  
Shape b3=new Shape(12,23,2);  
System.out.println(b3.x + " , " + b3.y + " , " + b3.z);  
}  
}  
Another program to show overloaded methods are called is given.  
//Overloaded Constructor  
class CalcVolume  
{  
    int volume()  
    {  
        return -1;  
    }  
    int volume(int i)  
    {  
        return i*i*i;  
    }  
    long volume(int i,int j,int k)  
    {  
        return i*j*k;  
    }  
}  
class mainClass  
{  
    public static void main(String pa[])  
    {  
        int temp;
```

```

CalcVolume myObject=new CalcVolume();
    temp=myObject.volume();
System.out.println("Volume of Unknown Object is always:" + temp);

temp=myObject.volume(4);
System.out.println("Volume of a Cube:" + temp);
}
}

```

Consider, there are three methods with same name but with different parameters. In the main(), depending on number and type of parameters passed the respective method is called.

This is quite a powerful feature of OOP, because in most of situations one want to give the user or programmers, flexibility to choose the type of shape they want.

Q4. Difference between Method Overloading and Overriding a method.

Method Overloading	Method Overriding
<ul style="list-style-type: none"> (i) Overloaded methods supplement each other. (ii) Overloading methods may have different argument lists of different types. (iii) The return type may be different for overloaded methods (iv) Since overloading methods are essentially different methods, there is no restriction on exceptions they can throw. 	<ul style="list-style-type: none"> (i) An overriding method replaces the method it overrides. (ii) An overriding method must have argument lists of identical type and order. (iii) The return type must be same as that of overridden method. (iv) The overriding method must not throw checked exceptions which cannot be thrown by the original method.

Q5. Define Polymorphism.

What is polymorphism ? How is it implemented ? Show, through an example, the implementation of a polymorphism.

#Explain the concept of Polymorphism. Also, give an example of a Polymorphism. [June-06, Q1(a)]

In Java concept of polymorphism is implemented through Overloading the operators, Overriding a method. Below are the detailed description of both implementations :

OVERLOADING : One of the benefits of objects is that give the programmer a convenient way to construct new data types. Suppose you work with two-dimensional positions (such as X and Y coordinates, or latitude and longitude) in your program. You would like to express operations on these positional values with normal arithmetic operations, such as

Position1 = position2 + origin;

Where the variables position1, position2 and origin each represent a pair of independent numerical quantities. By creating a class that incorporates these two values, and declaring position1, position2 and origin to be objects of this class, we can, in effect, create a new data type. Many features of Java are intended to facilitate the creation of new data types in this manner.

Note that the = (equal) and + (plus) operators, used in the position arithmetic don't act the same way they do in operations on built-in types like int. The object position1 and so on are not predefined in Java, but are programmer-defined objects of class position. How do the = and + operations know how to operate on objects ? The answer is that we can define new operations for these operators. These operators will be member functions of the position class.

Using operators or functions in different ways, depending on what they are operating on , is called **polymorphism**. One thing with several distinct forms. When an existing operator, such as + or = is given the capability to operate on a new data type, it is said to be overloaded. **Overloading** is a kind of polymorphism; it is also an important feature of OOP.

Overriding a method : Java Runtime Polymorphism : Another way objects work together is to define methods that take other objects as parameters. You get even more cooperation and efficiency when the objects are united by a common superclass. All classes in the Java programming language have an inheritance relationship.

For example, if you define a method that takes a java.lang.Object as a parameter, it can accept any object in the entire Java platform. If you define a method that takes a java.awt.Component as a parameter, it can accept any component object. This form of cooperation is called *polymorphism*.

Overridden methods allow Java to support runtime polymorphism. Sometimes, we need to create a new method in a derived class with same signature and name as of its superclass member method. For example, consider a superclass **AutoMobile** having a method brake() and extended by class **MotorBike**. Although MotorBike has access of brake() of its superclass, but wants to add some new system like disc brakes in front wheel while applying brakes, so MotorBikes declare a method brake() with same return type and parameter list. Now whenever an instance of MotorBike calls the brake() method it calls

the one implemented in MotorBike, because it overrides the brake() of AutoMobile.

```
class AutoMobile
{
    String color;
    int weight;
    double bhp;

    void start()
    {
        System.out.println("Start the engine");
    }

    void brake()
    {
        System.out.println("Apply brakes");
    }
}

class MotorBike extends AutoMobile
{
    //start() is not an Overridden method, b'cos
    //signature's of start() in SuperClass is different
    String start(int no_of_kicks)
    {
        return "engine started in " + no_of_kicks + " kicks";
    }
    //Overridden brake()
    void brake()
    {
        System.out.println("Apply hydrolic brakes in rear and
                           disc brakes in front wheel");
    }
}

class mainClass
{
    public static void main(String args[])
    {
        MotorBike myBike=new MotorBike();

        myBike.color="BLUE";
        myBike.weight=120;
```

```

myBike.bhp=9.5;
String status=myBike.start(2);
System.out.println(status);
myBike.brake();
}
/*
The output of the program is:
engine started in 2 kicks
Apply hydraulic brakes in rear and disc brakes in front wheel
*/

```

Notes

- ▶ Always use private, public and protected access modifiers, for member declaration.
- ▶ Attributes of an objects which holds some value at a time should be declared as instance variable, and attributes which has some processing part should be declared as a method.
For example, in class **Automobile**, attributes such as color, no-of-wheels, weight etc. should be declared as instance variable and attributes such as ignition, acceleration, brake etc. should be declared as methods.
- ▶ Use final variables for attributes which are constant values and used frequently.
For example, value of Pi is fixed to 3.14.
- ▶ The **println** statement in the body part of the methods such as brake, ignition etc. where you do not know what to do in a non-real application.

Recursion

Java supports recursion. Recursion is the process of defining something in terms of itself. As it relates to Java programming, recursion is the attribute that allows a method to call itself. A method that calls itself is said to be recursive.

When a method calls itself, new local variables and parameters are allocated storage on the stack, and the method code is executed with these new variables from the start. A recursive call does not make a new copy of the method. Only the arguments are new. As each recursive call returns, the old local variables and parameters are removed from the stack, and execution resumes the point of the call inside the method.

Recursive versions of many routines may execute a bit more slowly than the iterative equivalent because of the added overhead of the additional function calls.

The main advantage to recursive methods is that they can be used to create clearer and simpler versions of several algorithms than can their iterative

relatives. For example, the QuickSort sorting algorithm is quite difficult to implement in an iterative way. Some problems, especially AI-related ones, seem to lend themselves to recursive solutions..

Note : When writing recursive methods, you must have an if statement somewhere to force the method to return without the recursive call being executed. If you don't do this, once you call the method, it will never return. This is a very common error in working with recursion. Use `Println()` statements liberally during development so that you can watch what is going on and abort execution if you see that you have made a mistake.

Understanding static

There will be times when you will want to define a class member that will be used independently of any object of that class. Normally a class member must be accessed only in conjunction with an object of its class. However, it is possible to create a member that can be used by its class. However, it is possible to create a member that can be used by itself, without reference to a specific instance. To create such a member, precede its declaration with keyword **static**. When a member is declared static, it can be accessed before any objects of its class are created, and without reference to any object. You can declare both methods and variables to be static. The most common example of a **static** member is a **main ()**. **Main ()** is declared as static because it must be called before any objects exist.

Instance variables declared as **static** are, essentially, global variables. When objects of its class are declared, no copy of a **static** variable is made. Instead, all instances of this class share the same **static** variable.

Methods declared as **static** have several restrictions:

- ▶ They can only call other **static** methods.
- ▶ They must only access **static** data.
- ▶ They cannot refer to **this** or **super** in anyway.

If you need to do computation in order to initialize your **static** variables, you can declare a **static** block which gets executed exactly once, when the class is first loaded the following example shows a class that has a static method, some **static** variables, and a **static** initialization block:

```
class Stat_Class
{
    static int counter;
    Stat_Class()
    {
        counter++;
    }
    //This Block initialized only once during entire life of Stat_Class
```

```

static
{
    System.out.println("Static block initialized");
    counter=1;
}

class Static_Demo
{
    public static void main(String args[])
    {
        Stat_Class stat1=new Stat_Class();
        System.out.println(stat1.counter);
        // We can also access the static variable counter with
        //out using Object name, for example Stat_Class.counter
        //which we done next

        Stat_Class stat2=new Stat_Class();
        System.out.println(Stat_Class.counter);
    }
}

```

Outside of the class in which they are defined, **static** methods and variables can be used independently of any object. To do so, you need only to specify the name of their class followed by the dot operator. For example, if you wish to call a **static** method from outside its class, you can do so by using the following, general form:

Classname.Method();

Here, classname is the name of the class in which the **static** method is declared. As you can see, this format is similar to that used to call non-**static** methods through object-reference variables. A **Static** variable can be accessed in the same way-by use of the dot operator on the name of the class. This is how Java implements a controlled version of global function and global variables.

Garbage Collection

Since objects are dynamically allocated by using the **new** operator, you might be wondering how such objects are destroyed and their memory is released for later reallocation. In some languages, such as C++, dynamically allocated objects must be manually released by use of a **delete** operator. Java takes a different approach; it handles deallocation for you automatically. The technique that accomplishes this is called garbage collection. It works like this: when no references to an object exist, that object is assumed to be no longer needed, and the memory occupied by the object can be reclaimed. There is no explicit need to destroy objects as in C++. Garbage collection only occurs sporadically

(if at all) during the execution of your program. It will not occur simply because one or more objects exist that are no longer used. Furthermore, different Java run-time implementation will take varying approaches to garbage collection, but for the most part, you should not have to think about it while writing your programs.

The finalize() Method

Sometimes an object will need to perform some action when it is destroyed. For example, if an object is holding some non-Java resource such as a file handle or window character font, then you might want to make sure these resources are freed before an object is destroyed. To handle such situations, Java provides a mechanism called Finalization. By using finalization you can define specific actions that will occur when an object is just about to be reclaimed by the garbage collector.

To add a finalizer to a class, you simply define the **finalize()** method. The Java run time calls that method whenever it is about to recycle an object of that class. Inside the **finalize()** method you will specify those actions that must be performed before the object is destroyed. The garbage collector runs periodically, checking for objects that are no longer referenced by any running state or indirectly through other reference objects. Right before an asset is freed, the Java run time calls the **finalize()** method on the object.

The **finalize()** method has this general form:

```
Protected void finalize( )
{
    //finalization code here
}
```

Here, the keyword **protected** is a specifier that prevents access to **finalize()** by code defined outside its class.

It is important to understand that **finalize()** is only called just prior to garbage collection. It is not called when an object goes out-of-scope, for example. This means that you cannot know when- or even if-**finalize()** will be executed. Therefore, your program should provide other means of releasing system resources, etc., used by the object. It must not rely on **finalize()** for normal program operation.

Q6. How does multithreading takes place on a computer with a single CPU?

In single CPU system, the task scheduler of operating system allocates execution time to multiple tasks. By quickly switching between executing tasks, it creates the impression that tasks executes concurrently.

CHAPTER 6

INHERITANCE

Introduction

Reusability is an important feature of OOP. It is always nice if we could reuse something that already exists rather than trying to create the same all over again i.e. “*why to reinvent the wheel.*”

The Java classes can be reused in several ways. Once a class has been written and tested, it can be adapted by other programmers to suit their requirements. This is basically done by creating new classes, reusing the properties of the existing ones. The mechanism of deriving a new class from an old one is called **Inheritance (or Derivation)**. The old class is referred to as the *base class* and the new one is called the *derived class* or *subclass*. Therefore, a subclass is a specialized version of a super class or base class and adds its own, unique elements.

Why Use Inheritance

Inheritance is a large part of the reason that Object Oriented Programming is a powerful tool. It has the following advantages:

- you can customize and enhance working classes
- it is easier to reuse code
- you can take a more general class and modify to suit a particular situation

Q1. What is multilevel inheritance? How does it differ from multiple inheritance? Also give an example of multiple inheritance.

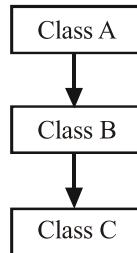
[June-06, Q1(d)]

Inheritance is a mechanism through which a class is shared by other class or classes. The class, which is shared, by other class is called BASE CLASS or SUPER CLASS and the class which share base class is called DERIVED CLASS or CHILD CLASS.

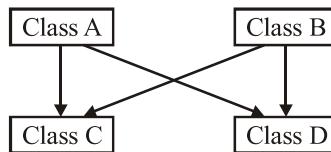
Types of Inheritance

1. Multi-level Inheritance
2. Multiple Inheritance

Multi-level Inheritance



Multiple Inheritance



Java doesn't support the multiple inheritance, so a derived class can only inherit one class, but Java fully support multilevel inheritance e.g., given three classes called A, B, and C, C can be a subclass of B, which is a subclass of A. Here C inherits all the traits of B, and A classes.

The example shows a short program, here class B inherits class A and all members of class A included in class B. That is, object of the class B can access any member of class A as it access its own member.

```

class A
{
    int i;
    void get_A_Val()
    {
        System.out.println("Value of i is : " + i);
    }
}
class B extends A
{
    int j;
    void get_B_Val()
    {
        get_A_Val(); // get_A_val() is now member of B
        System.out.println("Value of j is : " + j);
    }
}
  
```

```

        }
    }
class mainClass
{
    public static void main(String args[])
    {
        B objB;           //Object of B declared
        objB=new B(); //Object is instantiated

        objB.j=10;

        objB.i=5;      //objB accessing i as its own member

        objB.get_B_Val();
    }
}

```

Even though A is a superclass for B, it is also a completely independent, stand-alone class. Being a superclass for a subclass does not mean that the superclass cannot be used by itself. Further, a subclass can be a superclass for another subclass.

Q2. Define Extends keyword.

To inherit a class, Java uses **extends** keyword. To inherit a class, syntax for the derived class declaration is

```

class<derived class name> extends <base class>
{
    //body of the class
}

```

Java doesn't support the multiple inheritance, so a derived class can only inherit one class, but Java fully support multilevel inheritance e.g., given three classes called A, B, and C, C can be a subclass of B, which is a subclass of A. Here C inherits all the traits of B, and A classes.

The example shows a short program, here class B inherits class A and all members of class A included in class B. That is, object of the class B can access any member of class A as it access its own member.

```

class A
{
    int i;
    void get_A_Val()
    {
        System.out.println("Value of i is : " + i);
    }
}

```

```

        }
    }
class B extends A
{
    int j;
    void get_B_Val()
    {
        get_A_Val(); // get_A_val() is now member of B
        System.out.println("Value of j is : " + j);
    }
}
class mainClass
{
    public static void main(String args[])
    {
        B objB;           //Object of B declared
        objB=new B(); //Object is instantiated

        objB.j=10;

        objB.i=5;      //objB accessing i as its own member

        objB.get_B_Val();
    }
}

```

Even though A is a superclass for B, it is also a completely independent, stand-alone class. Being a superclass for a subclass does not mean that the superclass cannot be used by itself. Further, a subclass can be a superclass for another subclass.

Note : that not class can be a superclass of itself.

Note : that to include a previously compiled class in a new program, simply put the .class file of that class in the same directory where the new .java file is saved and compiled.

Member Access and Inheritance

For now, we focus on three types of access specifiers default, private and public.

default : When no specifier is used, default members are accessible within the

same class and to other class, provided both are in the same directory, in both ways through inheritance and objects.

private : Accessible only within the class.

public : Accessible in both conditions either inherited or by objects.

Q3. Differentiate between the “This” and “Super”.

#. What is ‘this’ keyword? How is it different from a super keyword ?

#. What is this pointer ? Explain its use with the help of an example.

[June-06, Q2(e)]

The “this” keyword : Sometimes a method will need to refer to the object that invoked it. To allow this, Java defines the this keyword. This can be used inside any method to refer to the current object. That is, this is always a reference to the object on which the method was invoked. You can use this anywhere a reference to an object of the current class’ type is permitted.

Instance Variable Hiding

As you know, it is illegal in Java to declare two local variables with the same name inside the same or enclosing scopes. Interestingly, you can have local variables, including formal parameters to methods, which overlap with the names of the class’ instance variables. However, when a local variable has the same name as an instance variable, the local variable hides the instance variable.

class Shape

```

{
    int x;
    int y;
    int z;
    Shape(int x,int y,int z)
    {
        this.x=i;
        this.y=j;
        this.z=k;
    }
}
class mainClass
{
    public static void main(String pa[])
    {
        Shape myShape=new Shape(12,23,2);
        System.out.println(myShape.x + " , " + myShape.y + " , " + myShape.z);
    }
}
```

Q4. What is ‘super’ in Java ? Explain at least two different uses of ‘super in the Java programs, with an example. [Dec-05, Q1(e)]

super keyword in Java, used to access the methods or data members defined in the superclass. *super* has two general forms.

(i) First Use : *super* for Constructor Reference

A subclass can call a constructor method defined by its superclass by use of the following form of *super*:

super (parameter_list);

In a subclass constructor, there is an implied first statement; the superclass constructor with no parameters is automatically called.

class A

{

 int i;

 A()

 {

 System.out.println("Default Constructor of A");

 }

 A(int a)

 {

 i=a;

 System.out.println("Parameterised Constructor of A");

 }

}

class B extends A

{

 int i;

 B()

 {

 //Default Constructor of A is automatically called

 System.out.println("Default Constructor of B");

 }

 B(int a,int b)

 {

 super(a); //Calling the Parameterised constructor of SuperClass A

 i=b;

```

        System.out.println("Parameterised Constructor of B");
    }
}
class mainClass
{
    public static void main(String args[])
    {
        B objB1=new B();
        B objB2=new B(2,4);
    }
}
/*

```

The **output** of the program is:

```

Default Constructor of A
Default Constructor of B
Parameterised Constructor of A
Parameterised Constructor of B
*/

```

In the preceding example, super() was called with three arguments. Since, constructors can be overloaded, super() can be called using any form defined by the superclass. The constructor executed will be the one that matches the arguments.

(ii) Second Use : Super for Member Access

super finds its use to access a member of the superclass that has been hidden (overloaded) by a member of a subclass. This usage has the following general form:

- (i) super. <member variable of superclass>;
- (ii) super. <member method of superclass> (parameter_list);

The example shows the second use of super.

class A

{

 int i;

 void show_A_val()

 {

 System.out.println("Value of i = " + i);

 }

}

class B extends A

```

{
    int j;
    void set_B_val(int a,int b)
    {
        super.i=a;           // access instance variable of Super Class
        j=b;
    }
    void show_B_val()
    {
        super.show_A_val(); //access method of Super Class
        System.out.println("Value of j = " + j);
    }
}
class mainClass
{
    public static void main(String args[])
    {
        B objB=new B();
        objB.set_B_val(12,23);
        objB.show_B_val();
    }
}
/*
The output of the program is:
Value of i = 12
Value of j = 23
*/

```

Q5. What are the five issues that are to be taken care of while overriding a method ? [June-06, Q1(e)]

Following are the five issues :

- (i) An overriding method (largely) replaces the method it overrides.
- (ii) Each method in a parent class can be overridden at most once in any one of the subclass.
- (iii) Overriding methods must have exactly the same argument lists, both in type and in order.
- (iv) An overriding method must have exactly the same return type as the method it overrides.
- (v) Overriding is associated with inheritance.

Q6. Difference between Final and Abstract.

What is final keyword? What happens if we apply keyword in class access specifier?

Describe the modifier abstract.

Final

A variable can be declared as **final**. Doing so prevents its content from being modified. This means that you must initialize a **final** variable when it is declared. (In this usage, **final** is similar to **const** in C/C++.) For example:

```
Final int FILE_NEW = 1;
Final int FILE_OPEN = 2;
Final int FILE_SAVE = 3;
Final int FILE_SAVEAS = 4;
Final int FILE_QUIT = 5;
```

Subsequent parts of your program can now use **FILE_OPEN**, etc., as if they were constants, without fear that a value has been changed.

It is a common coding convention to choose all uppercase identifiers for final variables. Variable declared as **final** do not occupy memory on a per-instance basis. Thus, a **final** variable is essentially a constant.

The keyword **final** can also be applied to methods but its meaning is substantially different than when it is applied to variables. Keyword **final** has its two special uses when applied in inheritance.

(i) final methods, these methods cannot be overridden, for example:

```
class A
{
    final void xyz( )
    {
        //body of Method
    }
}
class B extends A
{
    /* void xyz( ) overridden xyz( )
    {
    }
ERROR
*/
}
```

(ii) final classes, these classes cannot be inherited by other class, that is, they are just opposite to abstract classes.

A final class can only be used to create objects. For example:

```
final class A
```

```
{
    //class implementation
}
class B extends A
{
    //ERROR CAN'T SUBCLASS A
}
```

Abstract

When applied to a class, the ***abstract*** modifier indicates that the class has not been fully implemented and that it should not be instantiated like this:

```
A objA=new A(); //ERROR
```

but they can be used to create object references, because Java's approach to runtime polymorphism is implemented through the use of superclass reference.

abstract class A

```
{
    int i;
    void show()
    {
        System.out.println("Hello! This is A");
    }
}
```

class B extends A

```
{
    int j;
    void show()
    {
        System.out.println("Value of j = " + j);
    }
}
```

class mainClass

```
{
    public static void main(String args[])
    {
        B objB=new B();
    }
}
```

```
/* ERROR: A is abstract,can not be instantiated
   A objA=new A();
*/
```

```
A objA;
objA=objB; // objA references to objB
```

```

objA.i=6;           // objA can now access A's member
objA.show();         // show() of B:Run-Time Polymorphism

/* ERROR : j is not accessible through objA
   objA.j=5;
*/
   objB.j=5;
   objA.show();           //show() of B:Run-Time Polymorphism
}
}

```

/* Note that when objA calls show(), then Java run_time decides to call show() of B.

The **output** of the program is:

```

Value of j = 0
Value of j = 5
*/
```

If applied to a member method declaration, the *abstract* modifier means that the function will be implemented in a subclass. An abstract method has no implementation in class where it is declared. Instead it must be implemented in the subclass.

Note that any class having an abstract method must be declared abstract.

An abstract class can have normal method with abstract method.

abstract class A

```

{
    abstract void xyz( );           // No implementation here.
    void abc( )                    // A normal method.
    {
        //body of the method
    }
}

class B extends A
{
    void xyz( )                  //This is must
    {
        //body of the method
    }
}
```

Q7. What is inner class? Give its use. Describe with an example.

Introducing Nested and Inner Classes : It is possible to define a class within another class; such classes are known as nested classes. The scope of a nested class is bounded by the scope of its enclosing class. Thus, if class B is defined within class A, then B is known to A, but not outside of A. A nested class has access to the members, including private members, of the class in which it is nested. However, the enclosing class does not have access to the members of the nested class.

The most important type of nested class is the inner class. It has access to all of the variables and methods of its outer class and may refer to them directly in the same way that other non-static members of the outer class do. Thus, an inner class is fully within the scope of its enclosing class.

The following program illustrates how to define and use an inner class. The class named Outer has one instance variable named outer_x, one instance method named test(), and defines one inner class called Inner.

//Demonstrate an inner class.

```
class Outer {
    int outer_x = 100;
```

```
void test() {
    Inner inner = new Inner();
    inner.display();
}
```

```
//this is an inner class
class Inner {
    void display() {
        System.out.println("display: outer_x = " + outer_x);
    }
}
```

```
class InnerClassDemo {
    public static void main(String args[ ]) {
        Outer outer = new Outer();
        outer.test();
    }
}
```

Output from this application is shown here:

display: outer_x = 100

It is important to realize that class Inner is known only within the scope of class Outer. The Java compiler generates an error message if any code outside of class Outer attempts to instantiate class Inner. An inner class has access to all of the members of its enclosing class, but the reverse is not true. Members of the inner class are known only within the scope of the inner class and may not be used by the outer class.

CHAPTER 7

EXCEPTION HANDLING

An *exception* is an abnormal condition that arises in a sequence at run time. In other words, an exception is a run-time error. In computer languages that do not support exception handling, errors must be checked and handled manually—typically through the use of error codes, and so no. This approach is as cumbersome as it is troublesome. Java's exception handling avoids these problems and, in the process, brings run-time error management into the object-oriented world.

Q1. Define Exception handling features in Java.

A Java exception is an object that describes an exceptional (that is, error) condition that has occurred in a piece of code. When an exceptional condition arises, an object representing that exception is created in the method that caused the error. That method may choose to handle the exception itself, or pass it on. Either way, at some point, the exception is *caught* and processed. Exceptions can be generated by the Java run-time system, or they can be manually generated by your code. Exceptions thrown by Java relate to fundamental errors that violate the rules of the Java language or the constraints of the Java execution environment. Manually generated exceptions are typically, used to report some error condition to the caller of a method.

Java exception handling is managed via five keywords: **try**, **catch**, **throw**, **throws**, and **finally**.

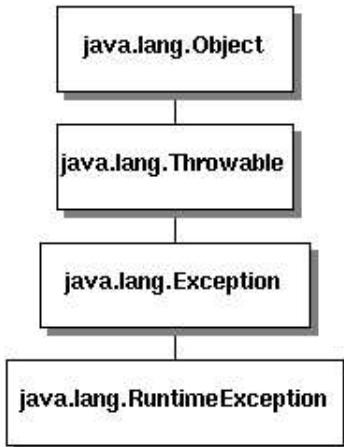
This is the general form of an exception-handling block:

```
try {  
    // block of code to monitor for errors  
}  
catch (Exception Type 2 exOb) {  
// exception handler for Exception Type 1  
}  
catch (Exception Type2 exOb) {  
// exception handler for Exception Type2  
}  
// ...  
finally {  
// block of code to be executed before try block ends  
}
```

Here, Exception type is the type of exception that has occurred.

Q2. Describe the following classes :

- * **Exception Class**
- * **Runtime Exception class**
- * **I/O exception class**



All exception types are subclasses of the built-in class **Throwable**. Thus, **Throwable** is at the top of the exception class hierarchy. Immediately below **Throwable** are two subclasses that partition exceptions into two distinct branches. One branch is headed by **Exception**. This class is used for exceptional conditions that user programs could catch. This is also the class that you will subclass to create your own custom exception types. There is an important subclass of **Exception**, called **Runtime Exception**. Exceptions of this type are automatically defined for the programs that you write and include things such as “division by zero” and “invalid array indexing”.

Q3. Describe in detail Error.

The other branch is topped by **Error**, which defines exceptions that are not expected to be caught under normal circumstances by your program. Exceptions of type **Error** are used by the Java run-time system to indicate errors having to do with the run-time environment, itself. “Stack overflow” is an **example** of such an error.

To guard against and handle a run-time error, simply enclose the code that you want to monitor inside a **try** block. Immediately following the **try** block, include a **catch** clause that specifies the exception type that you wish to catch. To illustrate how easily this can be done, the following program includes a **try**

block and a **catch** clause which processes the Arithmetic Exception generated by the division-by-zero error:

```
class Exc2 {
    public static void main (String args[ ]) {
        int d, a;
        try {                                     // monitor a block of code.
            d = 0;
            a = 42 / d ;
            System.out.println ("This will not be printed. ");
        } catch (ArithmaticException e) {           //catch divide-by-zero error
            System.out.println ("Division by Zero.");
        }
        System.out.println ("After catch statement.");
    }
}
```

This program generates the following output:

Division by zero.

After catch statement.

Notice that the call to **println()** inside the **try** block is never executed. Once an exception is thrown, program control transfers out of the **try** block into the **catch** block. Put differently, **catch** is not “called,” so execution never “returns” the **try** block from a **catch**. Thus, the line “This will not be printed.” is not displayed. Once the **catch** statement has executed, program control continues with the next line in the program following the entire **try/catch** mechanism.

A **try** and its **catch** statement form a unit. The scope of the **catch** clause is restricted to those statements specified by the immediately preceding **try** statement.

Multiple catch Clauses

In some cases, more than one exception could be raised by a single piece of code. To handle this type of exception, when an exception is thrown, each **catch** statement is inspected in order, and the first one whose type matches that of the exception is executed. After one **catch** statement executes, the others are bypassed, and execution continues after the **try/catch** block.

```
class multiple_Catch
{
    public static void main(String args[])
    {
        int n=1;
        try
        {
```

```
if(n==1)
{
    //Expression forces a divide by zero exception
    n=n/(n-n);
}
if(n==2)
{
    int a[]={1};
    //Accessing a non-existing index of Array
    a[42]=99;
}
}
catch(ArrayIndexOutOfBoundsException e)
{
    // When n=2
    System.out.println("Array Index Out Of Bound Exception" + e);
}
catch(ArithmaticException e)
{
    // When n=1
    System.out.println("Arithmatic Exception" + e);
}
```

Note: when you use multiple **catch** statements, it is important to remember that exception subclasses must come before any of their superclasses. This is because a **catch** statement that uses a superclass will catch exceptions of that type plus any of its subclasses. Thus, a subclass would never be reached if it comes after its superclass.

Q4. What is the difference between keyword throw and throws?

#. What is the keyword “throw” used for ?

So far, you have only been catching exceptions that are thrown by the Java run-time system. However, it is possible for your program to throw an exception explicitly, using the **throw** statement. The general form of throw is shown here;

Throw *Throwableinstance*;

Here, *Throwableinstance* must be an object of type **Throwable** or a subclass, of **Throwable**.

The flow of execution stops immediately after the throw statement; any

subsequent statements are not executed. The nearest **try** block is inspected to see if it has a **catch** statement that matches the type of the exception. If it does find a match, control is transferred to that statement. If not, then the next enclosing **try** statement is inspected, and so on. If no matching **catch** is found, then the default exception handler halts the program and prints the stack trace.

```
class exc_throw
{
    public static void main(String args[])
    {
        try
        {
            throw new ArithmeticException("hello");
        }
        catch(ArithmaticException e)
        {
            System.out.println("The exception catched is : " + e);
        }
    }
}

*****
```

Output is:

```
The exception catched is : java.lang.ArithmaticException: hello
*****
```

Here, **new** is used to construct an instance of **NullPointerException**. All of Java's built-in run-time exceptions have two constructors: one with no parameter and one that takes a string parameter.

Throws

If a method is capable of causing an exception that it does not handle, it must specify this behavior so that callers of the method can guard themselves against that exception. You do this by including a **throws** clause in the method's declaration. A **throws** clause lists the types of exceptions that method might throw.

This is the general (*parameter-list*) **throws** *exception-list*

```
{
    // body of method
}
```

Here, *exception-list* is a comma-separated list of the exceptions that a method can throw.

Following is an example of an incorrect program that tries to throw an exception that it does not catch. Because the program does not specify a **throws** clause to declare this fact, the program will not compile.

```
// This program contains an error and will not compile.
class Throws Demo {
    Static void throwOne ( ) {
        System.out.println ("Inside throwOne");
        Throw new IllegalAccessException ("demo");
    }
    public static void main (String args[ ]) {
        throwOne ( );
    }
}
```

To make this example compile, you need to make two changes. First you need to declare that **throwOne()** throws **IllegalAccessException**. Second, **main()** must define a **try/catch** statement that catches this exception.

The corrected example is shown here:

```
//This is now correct.
class ThrowsDemo {
    Static void throwOne( ) throws IllegalAccessException {
        System.out.println("Inside throwOne.");
        Throw new IllegalAccessException("demo");
    }
    public static void main(String args[ ]) {
        try {
            throwOne();
        } catch(IllegalAccessException e) {
            System.out.println("Caught " + e);
        }
    }
}
```

Here is the **output** generated by running this example program;

Inside throwOne

Caught Java.Lang.IllegalAccessException: demo

Q5. Define keyword ‘Finally’.

Finally creates a block of code that will be executed after a **try/catch** block has completed and before the code following the **try/catch** block. The **finally block** will execute whether or not an exception is thrown. If an exception is thrown, the **finally** block will execute even if no **catch** statement matches the exception. Any time a method is about to return to the caller from inside a **try/catch** block, via an uncaught exception or an explicit return statement, the **finally** clause is also executed just before the method returns. This can be useful for closing file handles and freeing up any other resources that might

have been allocated at the beginning of a method with the intent of disposing of them before returning. The **finally** clause is optional. However, each **try** statement requires at least one **catch** or a **finally** clause.

```
class exc_finally
{
    static void methodA()
    {
        try
        {
            throw new ArithmeticException("hello");
        }
        catch(ArithmeticException e)
        {
            System.out.println("Exception " + e + " is caught");
            throw new NullPointerException("Good Day");
        }
        finally
        {
            System.out.println("methodA's finally");
        }
    }

    public static void main(String args[])
    {
        try
        {
            methodA();
        }
        catch(NullPointerException e)
        {
            System.out.println("Exception " + e + " is caught");
        }
        finally
        {
            System.out.println("main's finally");
        }
    }
}
```

Output is:

```

Exception java.lang.ArithmaticException: hello is caught
methodA's finally
Exception java.lang.NullPointerException: Good Bye is caught main's fi-
nally
*****

```

Q6. Write an example of user defined exception.

#. Define an exception and discuss how user defined exceptions can be implemented in Java.

#. Describe the process of Creating Your Own Exception Subclasses.

An *exception* is an abnormal condition that arises in a sequence at run time. In other words, an exception is a run-time error. In computer languages that do not support exception handling, errors must be checked and handled manually—typically through the use of error codes, and so no. This approach is as cumbersome as it is troublesome. Java’s exception handling avoids these problems and, in the process, brings run-time error management into the object-oriented world.

Although Java’s built-in exceptions handle most common errors, you will probably want to create your own exception types to handle situations of **Exception** (which is, of course, a subclass of **Throwable**). Your subclasses don’t need to actually implement anything – it is their existence in the type system that allows you to use them as exceptions.

Method	Description
Throwable fillInStackTrace()	Returns a Throwable object that contains a completed stack trace. This object can be rethrown.
String getLocalizedMessage()	Returns a localized description of the exception.
String getMessage()	Returns a description of the exception.
void printStackTrace()	Displays the stack trace.
void printStackTrace(PrintStream stream)	Sends the stack trace to the specified stream.
void printStackTrace(PrintWriter stream)	Sends the stack trace to the specified stream.
String toString()	Returns a String object containing a description of the exception. This method is called by println() when outputting a Throwable object.

Table : The Methods defined by Throwable

The **Exception** class does not define any methods of its own. It does, of course, inherit those methods provided by **Throwable**. Thus, all exceptions, including those that you create, have the methods defined by throwable available to them. They are shown in above Table. You may also wish to override one or more of these methods in exception classes that you create.

The following example declares a new subclass of **Exception** and then uses that subclass to signal an error condition in a method. It overrides the **toString()** method, allowing the description of the exception to be displayed using **println()**.

//This program creates a custom exception types.

```
Class MyException extends Exception {
```

```
    Private int detail;
```

```
    MyException(int a) {
```

```
        Detail = a;
```

```
}
```

```
    public String toString( ) {
```

```
        return "MyException [" + detail + "] ";
```

```
}
```

```
}
```

```
class ExceptionDemo {
```

```
    static void compute (int a) throws MyException {
```

```
        System.out.println(" Called compute (" + a + ") ");
```

```
        If (a > 10)
```

```
            Throw new MyException (a);
```

```
            System.out.println(" Normal exit") ;
```

```
}
```

```
    public static void main(String args [ ] ) {
```

```
        try {
```

```
            compute(1) ;
```

```
            compute(20) ;
```

```
        } catch(MyException e) {
```

```
            System.out.println(" Caught " + e) ;
```

```
        }
```

```
}
```

```
}
```

This example defines a subclass of **Exception** called **MyException**. This subclass is quite simple; it has only a constructor plus an overloaded **toString()** method that displays the value of the exception. The **ExceptionDemo** class defines a method named **compute()** that throws a **MyException** object. The exception is thrown when **compute()**'s integer parameter is greater than 10. The **main()** method sets up an exception handler for **MyException**, then called **compute()** with a legal value (less than 10) and an illegal one to show

both paths through the code.

Here is the **result**:

Called compute (1)

Normal exit

Called compute (20)

Caught MyException [20]

Q7. The following program contains a bug. Find it and fix it.

```
// This program compiles but won't run successfully.
public class WhatHappens {
    public static void main(String[] args) {
        StringBuffer[] stringBuffer = new StringBuffer[10];
        for (int i = 0; i < stringBuffers.length; i++) {
            stringBuffers[i].append("StringBuffer at index " + i);
        }
    }
}
```

The program generates a Null Pointer Exception on line 6. The program creates the array, but does not create the string buffers, so it cannot append any text to them. The solution is to create the 10 string buffers in the loop with new StringBuffer() as follows:

```
public class ThisMustHappens
{
    public static void main(String[] args)
    {
        StringBuffer[] stringBuffers = new StringBuffer[10];

        for (int i = 0; i < stringBuffers.length; i++)
        {
            stringBuffer[i] = new StringBuffer();
            stringBuffers[i].append("StringBuffer at index " + i);
        }
    }
}
```

One object can be used as reference to another object provided both are of the same class **type**. Objects should be used as reference very carefully because if there is any change in values of instance variables of one object, values of instance variable of the second object also gets changed.

For Example, consider the following program:

```
// Objects are passed by reference
```

```
Class Test {
```

```
int a, b;
```

```
Test (int i, int j) {
```

```
a=i;  
b=j;  
}  
// pass an object  
void meth(Test o) {  
    o.a *= 2;  
    o.b /= 2;  
}  
}  
class CallByRef {  
    public static void main(String args[]) {  
        Test ob = new Test (15,20);  
        System.out.println("ob.a and ob.b before call: " + ob.a + " " + ob.b);  
        ob.meth(ob);  
        System.out.println("ob.a and ob.b after call: " + ob.a + " " + ob.b);  
    }  
}
```

This program generate the following output:

ob.a and ob.b before call: 15 20

ob.a and ob.b after call: 30 10

Here, inside meth() have affected the object used as an argument. This implies, when a simple type is passed to a method, it is done by call-by-value. Objects are passed by use of call-by-reference because the reference itself is passed by use of call-by-value. However, since the value being passed refers to an object, the copy of that value will still refer to the same object that its corresponding argument does.

Q8. Is there anything wrong with this exception handing code? Will this code compile?

```
Try  
{  
    //operation code...  
}  
catch (Exception e)  
{  
    //exception handing  
}  
catch (ArithmaticException ae)  
{  
    //ArithmaticException handing  
}
```

This first handler catches exception of type Exception; therefore, it catches any exception, including ArithmeticException. In this situation the second handler could never be reached.

Q9. Differentiate between checked and unchecked exceptions.

An unchecked exception is a type of exception that doesn't force you to handle it. It is optional to handle it, or ignore it. Exceptions instantiated from **RuntimeException** and its subclasses are considered unchecked exceptions. Checked exception are those exceptions that cannot be ignored during you write the code in your methods. The conceptual difference between checked and unchecked exceptions is that checked exceptions signal abnormal conditions that you have to deal with, and it is not the case with unchecked exceptions.

Q10. What is stream? Differentiate between stream source and stream destination.

A stream is a sequence of bytes of undetermined length that travel from one place to another over a communication path.

Places from where the streams are picked-up are known as stream source. A source may be a file, input device or a network place-generating stream.

Places where streams are received or stored are known as stream destination. A stream destination may be a file, output device or network place ready to receive stream.

Q11. Write a program to read the output of a file and display it on console.

This program Reads the content from Intro1.txt file and print it on console.

```
import java.io.*;
public class PrintConsol
{
    public static void main(String[] args)
    {
        try
        {
            FileInputStream FIS = new FileInputStream("Intro1.txt");
            Int n;
            While (( n = FIS.available()) > 0)
            {
                byte[] b = new byte[n];
                int result = FIS.read(b);
```

```

if (result == -1) break;
String s = new String(b);
System.out.print(s);
} // end while
FIS.close();
} // end try
catch (IOException e)
{
System.err.println(e);
}
System.out.println();
}
}

```

Q12. Differentiate between Transient & Volatile keyword.

Volatile indicates that concurrent running threads can modify the variable asynchronously. Volatile variables are used when multiple threads are doing the work. Transient keyword is used to declare those variables whose value need not persist when an object is stored.

Q13. Explain the use of equal() method with the help of code statements.

This instance method is used to compare the value held by the current object with the value held by another object. For example let CH1 and CH2 be two objects of Character class then Ch1.equals(CH2); method returns true if the values held by both objects are equal, otherwise false.

Q14. Write a program to find the length of string “Practice in programming is always good”. Find the difference between first and last occurrence of ‘r’ in this string.

```

class StringLen
{
public static void main(String[] args)
{
String MyStr = new String("Perfection in programming is always good");
System.out.println("The length of MyStr is :"+MyStr.length());
int i = MyStr.lastIndexOf("r")-MyStr.indexOf("r");
System.out.println("Difference between first and last occurrence of 'r' in
MyStr is :"+i);
}

```

```
}
```

Output:

The length of MyStr is:41

Difference between first and last occurrence of 'r' in MyStr is: 14

**Q15. Write a program, which takes full path (directory path and file name) of a file and display Extension, Filename and Path separately for example, for input"/home/mem/index.html",output is Extension = html
Filename = index Path = /home/mem.**

//It is assumed that fullPath has a directory path, filename, and extension.

```
class Filename
{
    private String fullPath;
    private char pathSeparator, extension Separator;
    public Filename(String str, char separ, char ext)
    {
        fullPath = str;
        pathSeparator = separ;
        extensionSeparator = ext;
    }
    public String extension()
    {
        int dot = fullPath.fastIndexOf(extensionSeparator);
        return fullPath.substring(dot + 1);
    }
    public String filename()
    {
        int dot = fullPath.lastIndexOf(extensionSeparator);
        int separ = fullPath.lastIndexOf(pathSeparator);
        return fullPath.substring(separ + 1, dot);
    }
    public String path()
    {
        int separ = fullPath.lastIndexOf(pathSeparator);
        return fullPath.substring(0, separ);
    }
}
// Main method is in FilenameDemo class
public class FilenameDemo1
{
    public static void main(String[] args)
    {
```

```

Filename myHomePage=new Filename("/HomeDir/MyDir/MyFile.txt",'/' );
System.out.println("Extension = "+ myHomePage.extension());
System.out.println("Filename = " + myHomePage.filename());
System.out.println("Path = " + myHomePage.path());
}
}

```

Output:

Extension = txt
 Filename = MyFile
 Path = /HomeDir/MyDir

Note: In this program you can notice that extension uses dot + 1 as the argument to substring. If the period character is the last character of the string, then dot + 1 is equal to the length of the string which is one larger than the largest index into the string (because indices start at 0). However, substring accepts an index equal to but not greater than the length of the string and interprets it mean “the end of the string.”

Q16. Write a program that finds initials from your full name and displays them.

```

public class NameInitials
{
    public static void main(String[] args)
    {
        String myNamels = "Mahesh Chand Verma";
        StringBuffer myNameInitials = new StringBuffer();
        System.out.println("The name is : "+myNamels);
        //Find length of name given
        int len = myNamels.length();
        for (int i = 0; i < len; i++)
        if (Character.isUpperCase(myNamels.charAt(i)))
            {
                myNameInitials.append(myNamels.charAt(i));
            }
        }
        System.out.println("Initials of the name: "+ myNameInitials);
    }
}

```

Output:

The name is: Mahesh Chand Verma
 Initials of the name: MCV

CHAPTER 8

PACKAGES & INTERFACES

Q1. What is a package in Java ? How is it created ? How can a package be used in another file ? List two important packages in Java, giving 2 important characteristics of each.

#: What are Packages in Java?

#: What is a package ? Explain the different access controls for packages in Java. [Dec-05, Q5(b)]

#: What is a Package ? Explain the process of defining our own package, with the help of an example. [June-06, Q2(b)]

In the preceding chapters, the name of each example class was taken from the same name space. This means that a unique name had to be used for each class to avoid name collision. After a while, without some way to manage the name space, you could run out of convenient, descriptive names for individual classes. You also need some way to be assured that the name you choose for a class will be reasonably unique and not collide with class names chosen by other programmers.

Packages are containers for classes that are used to keep the class name space compartmentalized. Packages are stored in a hierarchical manner and are explicitly imported into new class definitions. You can define classes inside a package that are not accessible by code outside that package. You can also define class members that are only exposed to other members of the same package.

To create a package, Java source file (.java) need to include a **package** statement. The **package** statement has the following syntax:

```
package <packagename>;
```

How to create packages?

Java uses file system directories to store packages. The following statement creates a package called MyPackage.

```
package MyPackage;
```

One must remember, while creating package:

- Each Java source file in folder MyPackage must include the statement above.
- package statement is the first statement in Java file.

- Classes you want to be accessible by code outside that package must be declared public.
- package names are case sensitive, so the folder name and name of package should be same.
- You can create a hierarchy of packages, that is, package within a package. The example of a multileveled package statement is package book.chapter.topic;
Here **book** is the top level package having another package **chapter** inside and **topic** package inside **chapter**.
When we create a package class, we should declare each class member with its access modifier. (private, public, protected).

Access Protection

In the preceding chapters, you learned about various aspects of Java's access control mechanism and its access specifies. For example, you already know that access to a **private** member of a class is granted only to other members of that class. Packages add another dimension to access control. As you will see, Java provides many levels of protection to allow fine-grained control over the visibility of variables and methods within classes, subclasses, and packages.

Classes and packages are both means of encapsulating and containing the name space and scope of variables and methods. Packages act as containers for classes and other subordinate packages. Classes act as containers for data and code. The class is Java's smallest unit of abstraction. Because of the interplay between classes and packages, Java addresses four categories of visibility for class members:

- Subclasses in the same package
- Non-subclasses in the same package
- Subclasses in different packages
- Classes that are neither in the same package nor subclasses

The three access specifies **private**, **public**, and **protected** provide a variety of ways to produce the many levels of access required by these categories. Following table sums up the interactions.

While Java's access control mechanism may seem complicated, we can simplify it as follows. Anything declared **public** can be accessed from anywhere. Anything declared **private** cannot be seen outside of its class. When a member does not have an explicit access specification, it is visible to subs classes as well as to other classes in the same package. This is the default access. If you want to allow an element to be seen outside your current package, but only to classes that subclass you directly, then declare that element **protected**.

	Private	No modifier	Protected	Public
Same class	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Same package subclass	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Same package non-subclass	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Different package subclass	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Different package non-subclass	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Table : Class Member Access

Table applies only to members of classes. A has only two possible access levels: default and public. When a class is declared as **public**, it is accessible by any other code. If a class has default access, then it can only be accessed by other code within its same package.

The following example shows all combinations of the access control modifiers. This example has two packages and four classes.

```

package p1;
public class X
{
    int def_X=1;
    private int pri_X=2;
    protected int pro_X=3;
    public int pub_X=4;
}
package p1;
// Y is a sub-class of X within the same package
class Y extends X
{
    Y()
    {
        System.out.println("Default member of X = " + def_X);
        // ERROR: Private of X is not accessible
        // System.out.println("Private member of X = " + pri_X);
        System.out.println("Protected member of X = " + pro_X);
        System.out.println("Public member of X = " + pub_X);
    }
}

```

```

package p2;
// sub_X is a sub-class of X in a different package
class sub_X extends p1.X
{
    sub_X()
    {
        // ERROR: Default & Private member of X are not accessible
        // System.out.println("Default member of X = " + def_X);
        // System.out.println("Private member of X = " + pri_X);
        System.out.println("Protected member of X = " + pro_X);
        System.out.println("Public member of X = " + pub_X);
    }
}
package p2;
// non_sub_X is a non sub-class of X in a different package
class non_sub_X
{
    non_sub_X()
    {
        p1.X objX=new p1.X();
        //ERROR: Default, Private and Protected member of X are not
accessible
        //System.out.println("Default member of X = " + objX.def_X);
        // System.out.println("Private member of X = " + objX.pri_X);
        // System.out.println("Protected member of X = " + objX.pro_X);
        System.out.println("Public member of X = " + objX.pub_X);
    }
}

```

public class X
 {
 //body of class
 }
 class Y extends X
 {
 //body of class
 }

P1

class sub_X extends P1.X
 {
 //body of class
 }
 class non_sub_X
 {
 //body of class
 }

P2

Class X has four instance variables, with default, private, protected & public access.

Class Y is a subclass of X within the same package. So Y can access default,
www.IgnouOnline.com

protected & public variables but not private.

Another package P2 has two classes, one is a subclass of X (subX), and other is a non-subclass of X (non_subX).

The subX can access only public and protected member of X.

The non-subX can access only public member of X.

Importing Packages

Given that packages exist and are a good mechanism for compartmentalizing diverse classes from each other, it is easy to see why all of the built-in Java classes are stored in packages. Since classes within packages must be fully qualified with their package name or names, it could become tedious to type in the long dot-separated package path name for every class you want to use. For this reason, Java includes the **import** statement to bring certain classes, or entire packages, into visibility. Once imported, a class can be referred to directly, using only its name. The **import** statement is a convenience to the programmer and is not technically needed to write a complete Java program. In a Java source file, **import** statements occur immediately following the **package** statement (if it exists) and before any class definitions. This is the general form of the **import** statement:

```
import pkg 1 [ .pkg 2] (classname | *);
```

Here, pkg1 is the name of a top-level package, and pkg 2 is the name of a subordinate package inside the other package separated by a dot (.). There is no practical limit on the depth of a package hierarchy, except that imposed by the file system. Finally, you specify either an explicit *classname* or a star (*), which indicates that the Java compiler should import the entire package.

The example demonstrate, how we can import classes into our program. Here, import the previously created class X of package P1 and see how its member access is possible.

```
import p1.*;
// importing a package p1
class mainClass
{
    public static void main(String args[])
    {
        X objX=new X();
        // ERROR: Default, Private and Protected member of X are not // accessible
        // System.out.println("Default member of X = " + objX.def_X);
        // System.out.println("Private member of X = " + objX.pri_X);
        // System.out.println("Protected member of X = " + objX.pro_X);
        System.out.println("Public member of X = " + objX.pub_X);
    }
}
```

{}

The 7 most commonly used packages in Java :

- package | java.util | : contains languages support classes of a more utilitarian nature. These include Linked List, Stack, Arrays and Hash Table Classes, as well as some useful abstract designs codified by the interfaces **Enumeration** and **Observer**.
- package | java.io | : It provides device_independent file and stream input and output services.
- package | java.awt | : It contains Java's Abstract Windows Toolkil (AWT), the package should really be considered as the heart of entire hierarchy.
- package | java.awt.image | : contains class related to image processing.
- package | java.net | : It combines the classes supporting low-level Internet programming plus World Wide Web/HTML support.
- package | java.applet | : contains a single class with support for HTML embedded Java applets.

Q2. What is a final variable? Can you define a final variable without supplying its value?

A variable can be declared as **final**. Doing so prevents its content form being modified. This means that you must initialize a **final** variable when it is declared. (In this usage, **final** is similar to **const** in C/C++.) For example:

```
Final int FILE_NEW = 1;
Final int FILE_OPEN = 2;
Final int FILE_SAVE = 3;
Final int FILE_SAVEAS = 4;
Final int FILE_QUIT = 5;
```

Subsequent parts of your program can now use **FILE_OPEN**, etc., as if they were constants, without fear that a value has been changed.

It is a common coding convention to choose all uppercase identifiers for final variables. Variable declared as **final** do not occupy memory on a per-instance basis. Thus, a **final** variable is essentially a constant.

Yes, we can define a final variable without supplying its value. We can supply value later on. For example, we can define:

```
Final int quantity;
quantity=50000;
```

Q3. Describe interface.

Interface : Through the use of the interface keyword, Java allows you to fully abstract the interface from its implementation. Using interface, you can specify a set of methods which can be implemented by one or more classes. The interface, itself, does not actually define any implementation. Although they are similar to abstract classes, interfaces have an additional capability: A

class can implement more than one interface. By contrast, a class can only inherit a single super class (abstract or otherwise).

Packages and interfaces are two of the basic components of a Java program. An interface is defined much like a class. This is the general form of an interface.

```
access interface name {
    return-type method-name1(parameter-list);
    return-type method-name2(parameter-list);
    type final-varname1 = value;
    type final-varname2 = value;
    // ...
    return-type method-nameN(parameter-list);
    type final-varnameN = value;
}
```

Q4. Write five differences between an interface and an abstract class.

[June-06, Q5(c)]

INTERFACES

(1) Using interface you can specify, what a class must do, but not how it does. Interface is a standalone structure.

(2) Method overridden in the class that implements interface must be proceeded by keyword public.

(3) They are designed to support dynamic method resolution at runtime without complexity of extensibility

ABSTRACT CLASSES

(1) Abstract classes also specify what must be done but not how it does but it is used with inheritance i.e. Abstract class is super class of the subclass in which abstract methods will be overridden. It is not a standalone structure.

(2) Method overridden does n't require public to be proceeded.

(3) They are also for dynamic method resolution but complexion of extensibility is there.

eg. Interface

```
interface Callback {
    void callback (int param);
}

class client implements Callback {
    public void callback (int p)
}
www.DoeaccOnline.com
```

```

    {
        System.out.println ("Callback called with " + p);
    }
}

```

Abstract class

```
abstract class A {
    abstract void callme ();
}
```

```
class B extends A {
    void call me () {
        {
            System.out.pinrln ("B's implementation of callme");
        }
    }
}
```

```
class AbstractDemo {
    public static void main (String args [ ])
    {
        B b = new B();
        b.callme ();
    }
}
```

Q5. Write difference between abstract classes and interface.

Abstract classes and interfaces carry similarity between them that methods of both should be implemented but there are many differences between them. These are:

A class can implement more than one interface, but an abstract class can only subclass one class.

An abstract class can have non-abstract methods.

All methods of an interface are implicitly (or explicitly) abstract.

An abstract class can have non-public methods.

Every method of an interface is implicitly (or explicitly) public.

An abstract class can declare instance variables; an interface cannot.

An abstract class can have a user-defined constructor; an interface has no constructors.

Q6. Show through a program that fields in an interface are implicitly static and final and methods are automatically public?

//program

Interface MyInterface

```
{  
void MyMethod();  
int MyValue1 = 500;  
}  
Class MyinterfaceImp implements MyInterface  
{  
public void MyMethod ()  
{  
System.out.println("Method of interface MyInterface is implemented with  
Value:"+MyValue1);  
//MyValue1 += 50;  
}  
}  
class Interface Test  
{  
public static void main(String args[])  
{  
MyinterfaceImp object = new MyinterfaceImp() ;  
Object.MyMethod(); //myMethod is by default public.  
{  
MyInterfaceImp object = new MyinterfaceImp() ;  
Object.MyMethod(); //myMethod is by default public.  
}  
}  
}
```

Output:

Method of interface MyInterface is implemented with value:500

If you remove comment (//) from statement “//My Value1 += 50;” and compile this program you will get:

Compile

InterfaceTest.java:12: cannot assign a value to final variable MyValue1
MyValue1 += 50;
^

1 error

This error is because of attempt to modify final variable My Value1.

CHAPTER 9

ABSTRACT WINDOW TOOLKIT

Introduction

The Abstract Windowing Toolkit (AWT) is an API that is responsible for building the Graphical User Interface (GUI). It is a part of the Java Foundation Classes (JFC). AWT includes a rich set of user interface components, a powerful event handling model, graphics and image tools, layout managers and support for data transfer using cut and paste through clipboards. AWT also supports javaBeans architecture. Every AWT component is a simple bean. The `java.awt` package contains all classes for creating user interfaces and for painting graphics and images.

The AWT has several subsystems that support the development of Graphical User Interface (GUI) programs. The subsystems include :

- ▶ Graphics primitives that allow the drawing and rendering of lines and images.
- ▶ Components such as Labels, Buttons and TextFields.
- ▶ Containers that include Frames, panels and Dialogs.
- ▶ Layout managers that control the display in a portable manner.
- ▶ Event system, which allows the user to respond to interactions between the user to respond to interactions between the components and containers in the application

The original AWT version 1.0 of the Java Development Kit (JDK) did its work by instantiating peer objects from the native operating system. Starting with JDK 1.1, Sun has made substantial changes to the user interface components. The AWT that came with 1.0 is no longer in use, it has been replaced by the AWT in 1.1. The interface in Java 1.1 still does its work through peer objects and these are now known as heavyweight components. These are also lightweight components that do not tie the user to the native operating system.

These instead support the pluggable look and feel. Java makes many details of the graphical user interface (GUI) programming invisible but they are still present. This is possible because of the platform independence of Java.

User Interface Components

A place in which the various drawing needs to be done must be provided and this is called the container. The container is derived from `java.awt.Container` class. The elements of the `java.awt.Component` class.

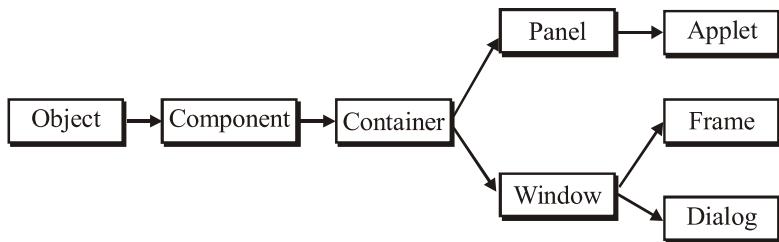
The `Component` class is the abstract superclass of the nonmenu-related Abstract Window Toolkit components. All the user interface components and container classes are derived from this class. This class is responsible for a variety of activities like display event handling and font management.

There are two major sets of classes derived from the component class:

- **Container** classes - They are generic AWT components that can contain other components.
- **User Interface component** classes - these include components like Button, label etc.

Containers

A generic Abstract Window Toolkit (AWT) container object is a component that can contain other AWT components. User interface components added to a container are tracked in a list. The order of the list will define the components' front-to-back stacking order within the container. If no index is specified when adding a component to a container, it will be added to the end of the list. Components have to be necessarily added to a container if they have to be displayed on the screen. Fig. 8.1 shows the class hierarchy of the classes that can act as containers.



The basic functionalities of containers are encapsulated in an abstract class, `Container`. This class has methods that allow other components to be nested within it. The panel is a simple non-abstract container that can contain other components. The `add()` method of the `Container` class can be used to add components to a `Panel`.

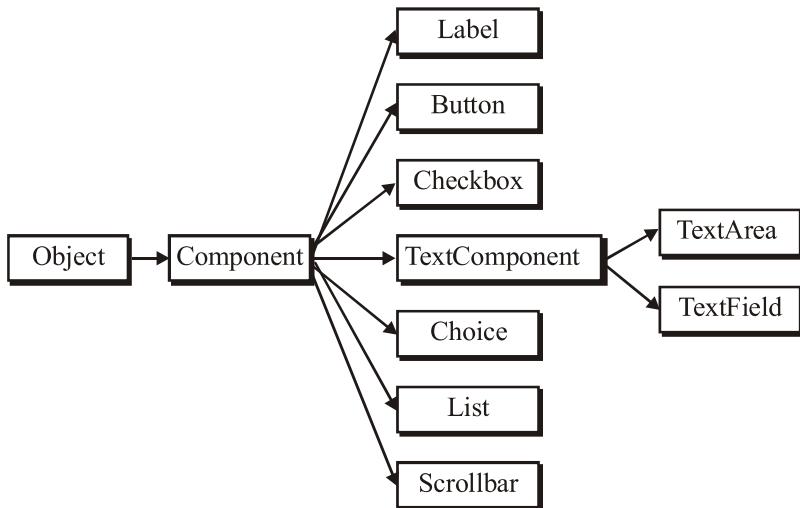
The `Applet` class is derived from the `panel` class and, hence, can act as a container. This property of the applet shall be exploited by adding user interface components and other containers directly to the applet.

Note

The Applet class, through derived from panel, does not belong to the AWT package.

User Interface Component Classes

The AWT package contains a number of component classes that are typical elements of any interactive user interface. These classes, collectively called UI component classes, are derived from the abstract Component class. Fig. below shows the class hierarchy of these classes.



The component class defines a number of methods that can be used on any of the classes that are derived from it. The methods below in table can be used in all GUI components as well as containers

Method	Description
setSize (Dimension d)	Resizes the corresponding component so that it has width d.width and height d.height.
setSize(int width, int height)	Resizes the corresponding component so that it has width and height.
setFont (font f)	Sets the font of corresponding component.
setEnabled(boolean b)	Enables or disables the corresponding component, depending on the value of the parameter b.
setVisible(boolean b)	Shows or hides the corresponding component depending on the value of parameter b.
setForeground(Color c)	Sets the foreground color of the corresponding component.
setBounds(int x, int y, int width, int height)	Moves and resizes the corresponding component.
setBounds(Rectangle r)	Moves and resizes the corresponding component to conform to the new bounding rectangle r.
setBackground(Color c)	Sets the background color of the corresponding component.
getBackground()	Gets the background color of the corresponding component.
getBounds()	Gets the bounds of the corresponding component in the form of a Rectangle object.
getFont()	Gets the font of the corresponding component.
getForeground()	Gets the foreground color of the corresponding component.
getSize()	Returns the size of the corresponding component in the form of a Dimension object.

The various user interface components have been shown in figure above the following sections will discuss each of these components in detail.

To effectively use these components, the following steps have to be carried out.

- First, the user interface component has to be created using the constructor of the corresponding class. For example, to create a Button object, the following code fragment can be used:

Button mybutton = new Button ("OK");

► Next, the component created is added to a container using the add() method of the Container class. For example, add(button); adds the button created to the container. Since, the Applet itself is derived from the Container class, the add() method can be used on an applet.

► Depending on the user interface component, it is necessary to handle the event generated by it. There are two ways of handling events as mentioned in the last chapter. But it is always wise to use listeners rather than letting the component handle its events by itself.

► To determine the component in which an event has occurred, it is necessary to use inner classes which was dealt in the fourth chapter. Each object has an inner class defined for it that implements their respective event listeners. The object of this inner class forms a parameter to the XXXListener() function in its outer class.

The user interface component classes also have a number of methods that can be used to change the attributes of the component like, the label of a button, text of a text field etc. the user interface component classes, including their events and methods, are discussed next.

Button

Buttons are components that can contain a label. It has an outline. The button is similar to a push button in any other GUI. Pushing a button causes the run time system to generate an event. This event is sent to the program. The program in turn can detect the event and respond to the event. Clicking a button generates an ActionEvent.

However, before a button can be used it has to be created using the new keyboard in association with the constructors that are defined for it. Buttons must be added to the containers before they can be used. This is done using the keyword add. Once the buttons have been created and added they can be used.

When a button is clicked the AWT sends an instance of the ActionEvent to the button, by calling the processEvent on the button. The processEvent method of the button receives all the events of the button. This then passes an action event along by calling its processActionEvent. The processActionEvent then passes the action event to any action listeners that have registered an interest in active events generated by the button in question.

Any application that has to perform an action based on the button has to implement the ActionListener and register the listener to receive the events from the button, by calling its addActionListener method.

Constructor	Description
Button()	Constructs a button with no label
Button (String label)	Constructs a button with the label specified.

Table lists some of the methods that can be used in relation to a Button component.

Method	Description
addActionListener (ActionListener1)	Adds the specified action listener to receive action events from the corresponding button.
getActionCommand()	Returns the command name of the action event fired by the corresponding button.
getLabel()	Returns the label of the corresponding button.
paramString()	Returns the parameter string representing the state of the corresponding button.
setLabel(Stringlabel)	Sets the label of the button of the value specified.

Example 1 : Write an applet that contains two labels, and two buttons, “OK” and “Cancel”. Two labels are used to display the text of the buttons namely - Button1 and Button2. the status bar should indicate about which button was clicked.

Enter the following code in a file and save it as Button Test.java.

```

1 : import java.awt.*;
2 : import java.applet.*;
3 : import java.awt.event.*;
4 : public class ButtonTest extends Applet {
5 : Label b1=new Label ("Button1");
6 : Button b1 =new Button ("ok");
7 : Label b2 =new Label ("Button2");
8 : Button b2 = new Button ("cancel");
9 : public void init () {
10 : setLayout (new FlowLayout ( ) );

```

```

11 : b1 . addActionListener (new B1 ( ) );
12 : b2 . addActionListener (new B2 ( ) );
13 : add (b1);
14 : add (b1);
15 : add (b2);
16 : add (b2);
17 : }
18 : class B1 implements ActionListener {
19 : public void actionPerformed (ActionEvent e)
20 : getAppletContext ( ).showstatus ("Button1: ok clicked");
21 : }
22 : }
23 : class B2 implements ActionListener {
24 : public void actionPerformed (ActionEvent e) {
25 : getAppletContext ( ).showStatus ("Button2: cancel clicked");
26 :         }
27 :     }
28 : }
```

- Enter the following script in a file and save it as ButtonTest.html.

<APPLET CODE= “ButtonTest .class” WIDTH=400 HEIGHT= 100> </APPLET>

Two buttons and two labels are added to the applet to the applet in the init () method in lines 13 to 16. Two action listeners are created in the inner classes B1 and B2 and are added to the applet using the addActionlistener () method. Note that these inner classes implement the ActionListener interface. The getAppletContext () method determines this applet’s context and uses the showstatus () method to display appropriate messages on the status line when the buttons are clicked. The setLayout() method is dealt in the latter half of this chapter.

CheckBox

Checkbox are user interface components that have state: checked and unchecked.

Clicking on it can change the state of the checkbox.

Java supports two types of checkboxes: exclusive and non-exclusive.

► In case of exclusive checkbox, only one group of items can be selected at a time. If an item from the group is selected, the checkbox currently checked is deselected and the new selection is highlighted. The exclusive checkboxes are also called **radio buttons**.

- The non-exclusive checkboxes are not grouped together and, each checkbox

can be selected independent of the other.

The various constructors of checkboxes include:

Constructor	Description
Checkbox()	Creates a checkbox with no label
Checkbox(String label)	Creates a checkbox with the specified label
Checkbox (String label), boolean state)	Creates a checkbox with the specified label and sets the specified state
Checkbox(String label, boolean state, CheckboxGroup group)	Creates a checkbox with the specified label and sets the specified state and places it in the specified group. The position of the checkbox group and state can be interchanged.

The various method that are supported by checkboxes are listed below :

Method	Description
getCheckboxGroup()	Determines the group of the corresponding check box
getLabel()	Gets the name of the corresponding check box
getSelectedObjects()	Returns an array (length 1) containing the checkbox label or null if the checkbox is not selected.
getState()	Determines if the checkbox is in the 'on' or 'off' state
setCheckboxGroup(CheckboxGroup g)	Sets the corresponding checkbox's group to the specified one.
setLabel(String label)	Sets the label of the corresponding checkbox to the value specified.
setState(boolean state)	Sets the state of the corresponding checkbox to the value specified.

Choice

The choice class implements a pop-up menu that allows the user to select an item from that menu. This UI component displays the currently selected item with an arrow to its right. On clicking the arrow, the menu opens and displays options for a user to select.

To create a choice menu, a Choice object is instantiated. Then, various items are added to the Choice by using the addItem() method. The various constructors that can be used to create a choice object is given below:

Constructor	Description
Choice()	Creates a new choice menu

The various methods that can be used along with a choice object are listed below:

Method	Description
add(String item)	Adds an item to the corresponding choice menu
addItem(String item)	Adds an item to the corresponding choice
getItem(int index)	Gets the string at the specified index of the corresponding choice menu
getItemCount()	Returns the number of items in the corresponding choice menu
getSelectedIndex()	Returns the index of the currently selected item
getSelectedItem()	Gets a representation of the current choice as a string.
getSelectedObjects()	Returns an array (length 1) containing the currently selected item.
insert(String item, int index)	Inserts the item into the corresponding choice at the specified position
remove(int position)	Removes an item from the corresponding choice menu at the specified position.
remove(String item)	Remove the first occurrence of item from the corresponding Choice menu.
removeAll()	Removes all items from the corresponding choice menu.
select (int pos)	Sets the selected item in the corresponding Choice menu to be the item at the specified position
select(String str)	Sets the selected item in the corresponding Choice menu to be the item whose name is equal to the specified string.

Example 2 :

```
1: import java.awt.*;
2: import java.awt.event.*;
3: import java.applet.*;
4: public class radioTest extends Applet {
5: public void init() {
6: CheckboxGroup c=new checkboxGroup( );
7: Checkbox c1=new checkbox("Colour and White", c, true) ;
8: Checkbox c2=new checkbox("Colour" ,c, false);
9: c1.addMouseListener(new check1 ( ));
10: c2.addMouseListener(new check2 ( ));
11: add(c1);
12: add(c2);
13: Choice abc= new Choice( );
14: abc.add("Onida");
15: abc.add("BPL");
16: abc.add("Samsung");
17: abc.add("philips");
18: abc.add("Videocon");
19: abc.addItemListener (new ch ( ) );
20: add(abc);
21: }
22: class check1 extends MouseAdapter {
23: public void mouseClicked (MouseEvent e) {
24: showStatus("You have selected : Black and white TV");
25: }
26: }
27: class check1 extends mouseAdapter {
28: public void mouseClicked(MouseEvent e) {
29: showStatus("You have selected : Colour TV") ;
30: }
31: }
32: class Ch implements Itemlistener {
33: public void itemStateChanged (itemEvent e) {
34: String s= (String)e.getItem ( );
35: showStatus ("You have selected" + s + "brand") ;
36: }
37: }
38: }
```

The radioTest.html file code is as follows:

```
<applet code="RadioTest.class" height =100 width =300>
</applet>
```

Q1. Where does the Applet class appear in the hierarchy of Java classes ? Show this through a class hierarchy diagram. [June-06, Q1(h)]

Class Applet
java.lang.Object

 java.awt.Component
 java.awt.Container
 java.awt.Panel
 java.applet.Applet

Q2. Write Do's and Don'ts of Java Applets.

Following are the Do's and Don'ts for Java Applets:

Do's

- . Draw pictures on a web page
- . Create a new window and draw the picture in it.
- . Play sounds.
- . Receive input from the user through the keyboard or the mouse.
- . Make a network connection to the server from where the Applet is downloaded, and send to and receive arbitrary data from that server.

Don'ts

- . Write data on any of the host's disks.
- . Read any data from the host's disks without the user's permission. In some environments, notably Netscape, an Applet cannot read data from the user's disks even with permission.
- . Delete files
- . Read from or write to arbitrary blocks of memory, even on a non-memory-protected operating system like the MacOS
- . Make a network connection to a host on the Internet other than the one from which it was downloaded.
- . Call the native API directly (though Java API calls may eventually lead back to native API calls).
- . Introduce virus or Trojan horse into the host system.

Q3. Is it possible to access the network resources through an Applet on the browser?

There is no way that an Applet can access network resources. You have to implement a Java Query Server that will be running on the same machine from where you are receiving your Applet (that is Internet Server). Your Applet will

communicate with that server which fulfills all the requirement of the Applet. You communicate between Applet and Java query server using Remote Method Invocation (RMI), this may be given preference because it will return the whole object.

Q4. Write an Applet program in which you place a button and a textarea. When you click on button, in text area Your name and address is displayed. You have to take your name and address using <PARAM>.

```
import Java.Applet.*;
import Java.awt.*;
public class DrawStringApplet1 extends Applet
{
    public void paint(Graphics g)
    {
        String str1 = this.getParameter("Message1");
        g.drawString(str1, 50, 25);
        String str2 = this.getParameter(Message2");
        g.drawString(str2, 50, 50);
    }
}
```

DrawStringApplet1.html file:

```
<HTML>
<HEAD>
<TITLE> Draw String </TITLE>
</HEAD>
<BODY>
<APPLET code="DrawStringApplet1" width="300" height="250">
<PARAM name="Massage1" value="Dinesh Verma">
<PARAM name = "Massage2" value= "GPH, New Delhi-35">
</APPLET>
</BODY>
</HTML>
```

Compile DrawStringApplet1.Java file then run DrawStringApplet1.html file either in web browser or through Appletviewer.

Q5. Give examples of stretchable components and non-stretchable components

Stretchable: Button, Label and TextField

Non-Stretchable: CheckBox

Q6. How many Listeners are there for trapping mouse movements.

MouseListener and MouseMotionListener.

Q7. Define the term Swing.

Swing is a GUI toolkit for Java. Swing is one part of the Java Foundation Classes (JFC). Swing includes graphical user interface (GUI) widgets such as text boxes, buttons, split-panes, and tables.

Swing widgets provide more sophisticated GUI components than the earlier Abstract Windowing Toolkit. Since they are written in pure Java, they run the same on all platforms, unlike the AWT which is tied to the underlying platform's windowing system. Swing supports pluggable look and feel – not by using the native platform's facilities, but by roughly emulating them. This means you can get any supported look and feel on any platform. The disadvantage of lightweight components is possibly slower execution. The advantage is uniform behavior on all platforms. In short we can say one line about Swing, "Swing" refers to the new library of GUI controls (buttons, sliders, checkboxes, etc.) that replaces the somewhat weak and inflexible AWT controls.

Q8. Write some main features of Swing.

Lightweight. Not built on native window-system windows.

Many miscellaneous new features. Double-buffering built in, tool tips, dockable tool bars, keyboard accelerators, custom cursors, etc.

Much bigger set of built-in controls. Trees, image buttons, tabbed panes, sliders, toolbars, color choosers, tables, text areas to display HTML or RTF, etc.

Much more customizable. Can change border, text alignment, or add image to almost any control. Can customize how minor features are drawn. Can separate internal representation from visual appearance.

"Pluggable" look and feel. Can change look and feel at runtime, or design own look and feel.

Q9. Why Swing is preferred over AWT ? Can we combine AWT and Swing in the same container ? Justify your answer. Also write some differences between Swing and AWT.

You must be thinking that when you can make GUI interface with AWT package then what is the purpose of learning Swing-based GUI? Actually Swing has *lightweight* components and does not write itself to the screen, but redirects it to the component it builds on. On the other hand AWT are heavyweight and have their own view port, which sends the output to the screen. Heavyweight components also have their own look and feel dependent on the machine on which the program is running. This is the reason why you can't

combine AWT and Swing in the same container. If you do, AWT will always be drawn on top of the Swing components.

Another difference is that Swing is pure Java, and therefore platform independent. Swing looks identically on all platforms, while AWT looks different on different platforms.

See, basically Swing provides a rich set of GUI components; features include model-UI separation and a pluggable look and feel. Actually you can make your GUI also with AWT but with Swing you can make it more user-friendly and interactive. Swing components make programs efficient.

Swing GUI components are packaged into Package javax.swing. In the Java class hierarchy there is a class.

Class Component which is a contains method paint for drawing Component onscreen Class container which is a collection of related components and contains method add for adding components and Class JComponent and contains method adding components and Class JComponent which has *Pluggable look and feel* for customizing look and feel.

Q10. Write complete hierarchy upto JComponent.

The Hierarchy is as follows:

Object—>Component—>Container—>JComponent

In Swing we have classes prefixed with the letter 'J' like

JLabel ->Displays single line of read only text

JtextField ->Displays or accepts input in a single line.

JtextArea -> Displays or accepts input in multiple lines

JCheckBox ->Give choices for multiple options

Jbutton ->Accepts command and does the action

Jlist - > Gives multiple choices and display for selection

JRadioButton - > Gives choices for multiple option, but can select one at a time.

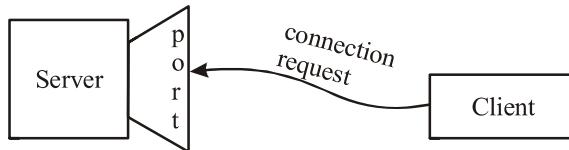
CHAPTER 10

NETWORKING FEATURES

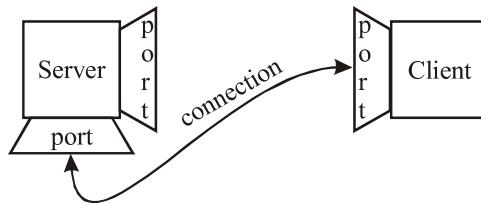
Q1. Define the concept of Client/Server and Socket.

Normally, a server runs on a specific computer and has a socket that is bound to a specific port number. The server just waits, listening to the socket for a client to make a connection request.

On the client-side: The client knows the hostname of the machine on which the server is running and the port number to which the server is connected. To make a connection request, the client tries to rendezvous with the server on the server's machine and port.



If everything goes well, the server accepts the connection. Upon acceptance, the server gets a new socket bound to the same port. It needs a new socket so that it can continue to listen to the original socket for connection requests while tending to the needs of the connected client.



On the client side, if the connection is accepted, a socket is successfully created and the client can use the socket to communicate with the server. The client and server can now communicate by writing to or reading from their sockets.

Q2. Define the term Socket.

A *socket* is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent.

Q3. Discuss about java.net package in brief.

The `java.net` package in the Java platform provides a class, `Socket`, that implements one side of a two-way connection between your Java program and another program on the network. The `Socket` class sits on top of a platform-dependent implementation, hiding the details of any particular system from your Java program. By using the `java.net.Socket` class instead of relying on native code, your Java programs can communicate over the network in a platform-independent fashion.

Additionally, `java.net` includes the `ServerSocket` class, which implements a socket that servers can use to listen for and accept connections to clients. This lesson shows you how to use the `Socket` and `ServerSocket` classes.

If you are trying to connect to the Web, the `URL` class and related classes (`URLConnection`, `URLEncoder`) are probably more appropriate than the socket classes. In fact, URLs are a relatively high-level connection to the Web and use sockets as part of the underlying implementation.

Q4. Define various types of Sockets. Also write about socket selection criteria.

(i) SOCK_STREAM : This type of socket uses Transmission Control Protocol (TCP). The stream socket (SOCK_STREAM) interface defines a reliable connection oriented service. Data is sent without errors or duplication and is received in the same order as it is sent.

(ii) SOCK_DGRAM : This type of socket uses User Datagram Protocol (UDP). This datagram socket (SOCK_DGRAM) interface defines a connectionless service for datagrams, or messages. Datagrams are sent as independent packets. The reliability is not guaranteed, data can be lost or duplicated. However, datagram sockets have improved performance capability over stream sockets and are easier to use.

(iii) SOCK_RAW : This type of socket uses IP, ICMP, and RAW protocols. The raw socket (SOCK_RAW) interface allowed direct access to lower layer protocols such as Internet Protocol(IP).

Usually it depends on the data that which type of socket one should use to transfer data. When the integrity of the data is high priority, you must use stream sockets. On the other side when data integrity is not of high priority(e.g. terminal inquiries), datagram sockets should be used.

Two communication protocols can be used for socket programming (i) Datagram communication or UDP. (ii) Stream Communication or TCP.

Q5. What Is a URL?

URL is an acronym for Uniform Resource Locator and is a reference (an address) to a resource on the Internet. It's often easiest, although not entirely accurate, to think of a URL as the name of a file on the World Wide Web because most URLs refer to a file on some machine on the network. However, remember that URLs also can point to other resources on the network, such as database queries and command output.

Q6. Discuss in detail Reading from and Writing to a URLConnection Class.

If you've successfully used openConnection to initiate communications with a URL, then you have a reference to a URLConnection object. The URLConnection class contains many methods that let you communicate with the URL over the network. URLConnection is an HTTP-centric class; that is, many of its methods are useful only when you are working with HTTP URLs. However, most URL protocols allow you to read from and write to the connection. This section describes both functions.

Reading from a URLConnection

Rather than getting an input stream directly from the URL, this program explicitly opens a connection to a URL and gets an input stream from the connection. Then, like URLReader, this program creates a BufferedReader on the input stream and reads from it.

```
import java.net.*;
import java.io.*;

public class URLConnectionReader {
    public static void main(String[] args) throws Exception {
        URL yahoo = new URL("http://www.yahoo.com/");
        URLConnection yc = yahoo.openConnection();
        BufferedReader in = new BufferedReader(
            new InputStreamReader(
                yc.getInputStream()));
        String inputLine;
        while ((inputLine = in.readLine()) != null)
            System.out.println(inputLine);
        in.close();
    }
}
```

{

The output from this program is identical to the output from the program that opens a stream directly from the URL. You can use either way to read from a URL. However, reading from a URLConnection instead of reading directly from a URL might be more useful. This is because you can use the URLConnection object for other tasks (like writing to the URL) at the same time.

Again, if the program hangs or you see an error message, you may have to set the proxy host so that the program can find the Yahoo server.

Writing to a URLConnection

Many HTML pages contain forms-- text fields and other GUI objects that let you enter data to send to the server. After you type in the required information and initiate the query by clicking a button, your Web browser writes the data to the URL over the network. At the other end, a cgi-bin script (usually) on the server receives the data, processes it, and then sends you a response, usually in the form of a new HTML page.

Many cgi-bin scripts use the POST METHOD for reading the data from the client. Thus writing to a URL is often called posting to a URL. Server-side scripts use the POST METHOD to read from their standard input.

Note: Some server-side cgi-bin scripts use the GET METHOD to read your data. The POST METHOD is quickly making the GET METHOD obsolete because it's more versatile and has no limitations on the amount of data that can be sent through the connection.

A Java program can interact with cgi-bin scripts also on the server side. It simply must be able to write to a URL, thus providing data to the server. It can do this by following these steps:

Create a URL.

Open a connection to the URL.

Set output capability on the URLConnection.

Get an output stream from the connection. This output stream is connected to the standard input stream of the cgi-bin script on the server.

Write to the output stream.

Close the output stream.

Here's an example program that runs the backwards script over the network through a URLConnection:

```
import java.io.*;
```

```
import java.net.*;  
  
public class Reverse {  
    public static void main(String[] args) throws Exception {  
  
        if (args.length != 1) {  
            System.err.println("Usage: java Reverse "  
                + "string_to_reverse");  
            System.exit(1);  
        }  
  
        String stringToReverse = URLEncoder.encode(args[0]);  
  
        URL url = new URL("http://java.sun.com/cgi-bin/backwards");  
       URLConnection connection = url.openConnection();  
        connection.setDoOutput(true);  
  
        PrintWriter out = new PrintWriter(  
            connection.getOutputStream());  
        out.println("string=" + stringToReverse);  
        out.close();  
  
        BufferedReader in = new BufferedReader(  
            new InputStreamReader(  
                connection.getInputStream()));  
        String inputLine;  
  
        while ((inputLine = in.readLine()) != null)  
            System.out.println(inputLine);  
  
        in.close();  
    }  
}
```

Let's examine the program and see how it works. First, the program processes its command-line arguments:

```
if (args.length != 1) {  
    System.err.println("Usage: java Reverse " +  
        "string_to_reverse");  
    System.exit(-1);  
}
```

```
String stringToReverse = URLEncoder.encode(args[0]);
```

These statements ensure that the user provides one and only one command-line argument to the program, and then encodes it. The command-line argument is the string that will be reversed by the cgi-bin script backwards. It may contain spaces or other non-alphanumeric characters. These characters must be encoded because the string is processed on its way to the server. The URLEncoder class methods encode the characters.

Next, the program creates the URL object--the URL for the backwards script on java.sun.com--opens a URLConnection, and sets the connection so that it can write to it:

```
URL url = new URL("http://java.sun.com/cgi-bin/backwards");
URLConnection c = url.openConnection();
c.setDoOutput(true);
```

The program then creates an output stream on the connection and opens a PrintWriter on it:

```
PrintWriter out = new PrintWriter(c.getOutputStream());
```

If the URL does not support output, getOutputStream method throws an UnknownServiceException. If the URL does support output, then this method returns an output stream that is connected to the standard input stream of the URL on the server side--the client's output is the server's input.

Next, the program writes the required information to the output stream and closes the stream:

```
out.println("string=" + stringToReverse);
out.close();
```

This code writes to the output stream using the println method. So you can see that writing data to a URL is as easy as writing data to a stream. The data written to the output stream on the client side is the input for the backwards script on the server side. The Reverse program constructs the input in the form required by the script by concatenating string= to the encoded string to be reversed.

Often, when you are writing to a URL, you are passing information to a cgi-bin script, as in this example. This script reads the information you write, performs some action, and then sends information back to you via the same URL. So it's likely that you will want to read from the URL after you've written to it. The Reverse program does this:

```

BufferedReader in = new BufferedReader(
    new InputStreamReader(c.getInputStream()));
String inputLine;

while ((inputLine = in.readLine()) != null)
    System.out.println(inputLine);
in.close();

```

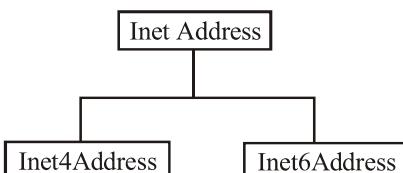
When you run the Reverse program using "Reverse Me" as an argument (including the double quote marks), you should see this output:
 Reverse Me reversed is: eM esreveR.

java.net Package

java.net provides the following addressing-related classes:

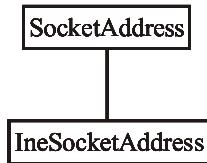
- InetAddress
- Inet4Address
- Inet6Address
- SocketAddress
- InetSocketAddress

For IP addressing, three classes are provided: InetAddress, Inet4Address, and Inet6Address. InetAddress represents an IP address, which is either a 32- or 128-bit unsigned number used by IP, the lower-level protocol on which protocols like TCP and UDP are built. To represent 32-bit IPv4 address, Inet4Address is provided. (An IPv4 address has the familiar form nnn.nnn.nnn.nnn, where n is an integer; e.g., 130.251.36.251). It is a subclass of InetAddress. To represent 128-bit IPv6 addresses, Inet6Address is provided. It is also a subclass of InetAddress.



For socket addressing, two classes are provided: SocketAddress and InetSocketAddress. SocketAddress is an abstract socket address, independent of a specific protocol. It is intended for subclassing for a specific protocol. InetSocketAddress below is an example. InetSocketAddress is a subclass of SocketAddress; it represents an IP socket address. It can include an IP address (e.g., 216.157.138.192) and port (e.g., 80); a hostname (e.g., gullybaba.com) and port (e.g., 1000); or port only (e.g., 1010). In the latter

case, a wildcard IP address is assumed.



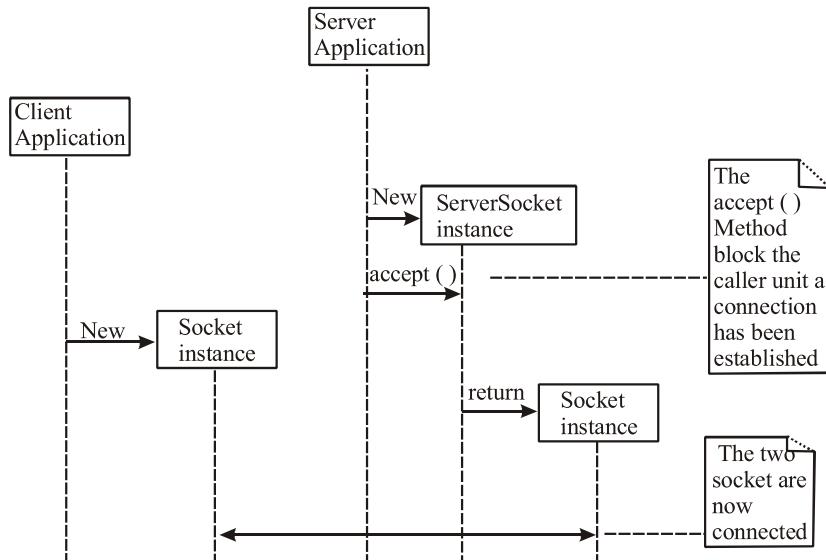
Making TCP Connections

These classes are related to making normal TCP connections:

- ServerSocket
- Socket

For simple connections between a client and a server, ServerSocket and Socket are all that you will probably need.

ServerSocket represents the socket on a server that waits and listens for requests for service from a client. Socket represents the endpoints for communication between a server and a client. When a server gets a request for service, it creates a Socket for communication with the client and continues to listen for other requests on the ServerSocket. The client also creates a Socket for communication with the server. The sequence is shown below:



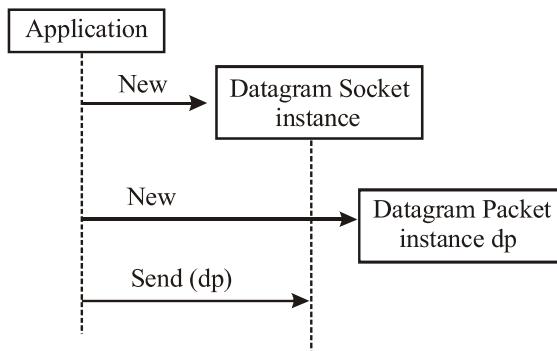
Once the connection is established, `getInputStream()` and `getOutputStream()` may be used in communication between the sockets

Sending/Receiving Datagram Packets via UDP

The following are related to sending and receiving datagram packets via UDP.

- DatagramPacket
- DatagramSocket

DatagramPacket represents a datagram packet. Datagram packets are used for connectionless delivery and normally include destination address and port information. DatagramSocket is a socket used for sending and receiving datagram packets over a network via UDP. A DatagramPacket is sent from a DatagramSocket by calling the send(...) method of DatagramSocket with DatagramPacket as the argument: send(DatagramPacket dp). receive(DatagramPacket dp) is use for receiving a DatagramPacket. (The MulticastSocket class may be used for sending/receiving a DatagramPacket to a mulitcast group. It is a subclass of DatagramSocket that adds functionality for multicasting.)



CHAPTER 11

ADVANCED JAVA

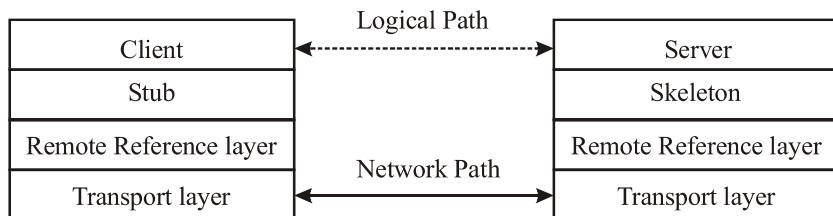
RMI

Q1. Discuss about RMI in detail.

RMI is one of the core Java APIs since version 1.1. It provides a framework for distributed computing. With RMI, separate parts of a single program can exist in multiple Java environments on multiple machines. To the client and server, the program operates much like it is local to their environment and within a single memory space. RMI is one of the fundamental APIs that Enterprise Java Beans are built on.

There are many cases where a distributed environment is beneficial. As applications grow large and complex they are more manageable, highly performant, and more robust when they are distributed among multiple machines. Resources can be physically located where they make the most sense. Dedicated machines can be configured optimally to perform specialized operations. Heavy processing can be allocated to the well suited hardware, data processing can be handled by centralized data servers, and web servers can just serve web pages. A large application can distribute the workload and achieve better efficiency and data isolation. RMI is a proven framework to design and implement a distributed application in a Java environment. Due to the multi-platform nature of Java, RMI is probably the best choice of API for this type of application.

The client is defined as the application which uses a remote object, and the server is defined as the machine which hosts the remote object. Communication is two way. Below is the Java RMI architecture.



To both the client and server, the application appears to be local even though it is actually distributed among multiple Java environments. This is achieved by implementing interfaces which define the methods which can be invoked. The interfaces define the contract between the objects. What is actually passed back and forth are primitive data types (passed by value), remote references (objects which implement `java.Remote`), or copies of local objects (objects which implement `java.io.Serializable`). As one would expect, changes to the remote reference results in changes to the object; changes to the copy do not. Remote objects are listed in the **rmiregistry** of the server. A client will query the rmiregistry for an object by its handle and will receive a reference to the remote object. The client then invokes methods on the remote object. What actually is happening is that the client is invoking methods locally on the stub which carries out the interaction. Likewise on the server side, the server interacts with the skeleton. The operations of the Remote Reference Layer and the Transport Layer are hidden from both sides. This allows different implementations of the Java Virtual Machine, which may be on different platforms, to integrate seamlessly.

Security

One of the most common problems one encounters with RMI is a failure due to security constraints.

A Java program may specify a security manager that determines its security policy. A program will not have any security manager unless one is specified. One sets the security policy by constructing a `SecurityManager` object and calling the `setSecurityManager` method of the `System` class. Certain operations require that there be a security manager. For example, RMI will download a `Serializable` class from another machine only if there is a security manager and the security manager permits the downloading of the class from that machine. The `RMISecurityManager` class defines an example of a security manager that normally permits such downloads. Above explanation is based on Java1.2 version.

Creating Distributed Applications Using RMI

When you use RMI to develop a distributed application, you follow these general steps.

Design and implement the components of your distributed application.

Compile sources and generate stubs.

Make classes network accessible.

Start the application.

Design and Implement the Application Components

First, decide on your application architecture and determine which components are local objects and which ones should be remotely accessible. This step includes:

- **Defining the remote interfaces:** A remote interface specifies the methods that can be invoked remotely by a client. Clients program to remote interfaces, not to the implementation classes of those interfaces. Part of the design of such interfaces is the determination of any local objects that will be used as parameters and return values for these methods; if any of these interfaces or classes do not yet exist, you need to define them as well.
- **Implementing the remote objects:** Remote objects must implement one or more remote interfaces. The remote object class may include implementations of other interfaces (either local or remote) and other methods (which are available only locally). If any local classes are to be used as parameters or return values to any of these methods, they must be implemented as well.
- **Implementing the clients:** Clients that use remote objects can be implemented at any time after the remote interfaces are defined, including after the remote objects have been deployed.

Compile Sources and Generate Stubs

This is a two-step process. In the first step you use the javac compiler to compile the source files, which contain the implementation of the remote interfaces and implementations, the server classes, and the client classes. In the second step you use the rmic compiler to create stubs for the remote objects. RMI uses a remote object's stub class as a proxy in clients so that clients can communicate with a particular remote object.

Make Classes Network Accessible

In this step you make everything—the class files associated with the remote interfaces, stubs, and other classes that need to be downloaded to clients—accessible via a Web server.

Start the Application

Starting the application includes running the RMI remote object registry, the server, and the client.

The rest of this lesson walks through the steps to create a compute engine.

Why use Distributed Objects?

Obviously, for the above application, it is probably overkill. However, when the resources become large, extensive processing is required, or resources exist in multiple physical locations, then distributed objects are the way to go.

The advantages include all of the power words that are commonly tossed around: scalability, maintainability, reliability, availability, extensibility, and manageability.

JAVA SERVLETS

Q2. What are Java Servlets ? Also list some example uses of it.

A *servlet* is a Java programming language class used to extend the capabilities of servers that host applications accessed via a request-response programming model. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by Web servers. For such applications, Java Servlet technology defines HTTP-specific servlet classes. The `javax.servlet` and `javax.servlet.http` packages provide interfaces and classes for writing servlets. All servlets must implement the Servlet interface, which defines life-cycle methods.

When implementing a generic service, you can use or extend the GenericServlet class provided with the Java Servlet API. The HttpServlet class provides methods, such as `doGet` and `doPost`, for handling HTTP-specific services.

Example Uses

A few of the many applications for servlets include,

- ▶ Processing data POSTed over HTTPS using an HTML form, including purchase order or credit card data. A servlet like this could be part of an order-entry and processing system, working with product and inventory databases, and perhaps an on-line payment system.
- ▶ Allowing collaboration between people. A servlet can handle multiple requests concurrently; they can synchronize requests to support systems such as on-line conferencing.
- ▶ Forwarding requests. Servlets can forward requests to other servers and servlets. This allows them to be used to balance load among several servers that mirror the same content. It also allows them to be used to partition a single logical service over several servers, according to task type or organizational boundaries.
- ▶ Being a community of active agents. A servlet writer could define active agents that share work among each other. Each agent would be a servlet, and the agents could pass data among themselves.

Q3. Discuss briefly about “Servlet Architecture”.

The main abstraction in the Servlet API is the Servlet interface. All servlets implement this interface, either directly or, more commonly, by extending a

class that implements it such as HttpServlet. The Servlet interface provides for methods that manage the servlet and its communications with clients. Servlet writers provide some or all of these methods when developing a servlet. When a servlet accepts a call from a client, it receives two objects: one is a ServletRequest and the other is a ServletResponse. The ServletRequest class encapsulates the communication from the client to the server, while the ServletResponse class encapsulates the communication from the servlet back to the client.

The ServletRequest interface allows the servlet access to information such as the names of the parameters passed in by the client, the protocol (scheme) being used by the client, and the names of the remote host that made the request and the server that received it. It also provides the servlet with access to the input stream, ServletInputStream, through which the servlet gets data from clients that are using application protocols such as the HTTP POST and PUT methods. Subclasses of ServletRequest allow the servlet to retrieve more protocol-specific data. For example, HttpServletRequest contains methods for accessing HTTP-specific header information.

The ServletResponse interface gives the servlet methods for replying to the client. It allows the servlet to set the content length and mime type of the reply, and provides an output stream, ServletOutputStream, and a Writer through which the servlet can send the reply data. Subclasses of ServletResponse give the servlet more protocol-specific capabilities. For example, HttpServletResponse contains methods that allow the servlet to manipulate HTTP-specific header information.

The classes and interfaces described above make up a basic Servlet. HTTP servlets have some additional objects that provide session-tracking capabilities. The servlet writer can use these APIs to maintain state between the servlet and the client that persists across multiple connections during some time period. All this makes together the Servlet Architecture.

Q4. Discuss briefly about “Servlet Lifecycle”.

Servers load and run servlets, which then accept zero or more requests from clients and return data to them. They can also remove servlets. These are the steps of a servlets lifecycle **In detail** : when a server loads a servlet, it runs the servlet's init method. Even though most servlets are run in multi-threaded servers, there are no concurrency issues during servlet initialization. This is because the server calls the init method once, when it loads the servlet, and will not call it again unless it is reloading the servlet. The server can not reload a servlet until after it has removed the servlet by calling the destroy method. Initialization is allowed to complete before client requests are handled (that is, before the service method is called) or the servlet is destroyed.

After the server loads and initializes the servlet, the servlet is able to handle client requests. It processes them in its service method. Each client's request has its call to the service method run in its own servlet thread: the method receives the client's request, and sends the client its response.

Servlets can run multiple service methods at a time. It is important, therefore, that service methods be written in a thread-safe manner. For example, if a service method might update a field in the servlet object, that access should be synchronized. If, for some reason, a server should not run multiple service methods concurrently, the servlet should implement the SingleThreadModel interface. This interface guarantees that no two threads will execute the servlet's service methods concurrently.

Servlets run until they are removed from the service, for example, at the request of a system administrator. When a server removes a servlet, it runs the servlet's destroy method. The method is run once; the server will not run it again until after it reloads and reinitializes the servlet. When the destroy method runs, however, other threads might be running service requests. If, in cleaning up, it is necessary to access shared resources (such as network connections to be closed), that access should be synchronized.

JAVA BEANS

Q5. Define Java Beans. Also write the main difference between ActiveX control and Java Beans.

Java beans is a specification developed by Sun Microsystems that defines how Java objects interact. Or A Java Bean is a reusable software component that can be visually manipulated in builder tool. An object that conforms to this specification is called a *JavaBean*, and is similar to an ActiveX control. It can be used by any application that understands the JavaBeans format.

The principal difference between ActiveX controls and JavaBeans are that ActiveX controls can be developed in any programming language but executed only on a Windows platform, whereas JavaBeans can be developed only in Java, but can run on any platform.

Q6. Discuss briefly about “JavaBeans Architecture”.

This architecture supports the features of software reuse, component models, and object orientation. One of the most important features of JavaBeans is that it does not alter the existing Java language. JavaBeans is an architecture for both using and building components in Java.

Although Beans are intended to work in a visual application development tool, they don't necessarily have a visual representation at run-time (although many will). What this does mean is that Beans must allow their property values to be

changed through some type of visual interface, and their methods and events should be exposed so that the development tool can write code capable of manipulating the component when the application is executed.

Here is some code that implements a simple Bean:

```
public class MyBean implements java.io.Serializable
{
    protected int theValue;

    public MyBean()
    {
    }

    public void setMyValue(int newValue)
    {
        theValue = newValue;
    }

    public int getMyValue()
    {
        return theValue;
    }
}
```

This is a real Bean named MyBean that has state (the variable theValue) that will automatically be saved and restored by the JavaBeans persistence mechanism, and it has a property named MyValue that is usable by a visual programming environment. This Bean doesn't have any visual representation, but that isn't a requirement for a JavaBean component.

JavaSoft is using the slogan "Write once, use everywhere." Of course "everywhere" means everywhere the Java run-time environment is available. But this is very important. What it means is that the entire run-time environment required by JavaBeans is part of the Java platform. No special libraries or classes have to be distributed with your components. The JavaBeans class libraries provide a rich set of default behaviors for simple components (such as the one shown earlier). This means that you don't have to spend your time building a lot of support for the Beans environment into your code.

Q7. Discuss briefly about “Main Features of JavaBeans”.

Compact and Easy

JavaBeans components are simple to create and easy to use. This is an important goal of the JavaBeans architecture. It doesn't take very much to write a

simple Bean, and such a Bean is lightweight? it doesn't have to carry around a lot of inherited baggage just to support the Beans environment. If a Bean does not require the advanced features of the architecture, it doesn't get them, nor does it get the code that goes with them. This is an important concept. The JavaBeans architecture scales upward in complexity, not downward like other component models. This means it really is easy to create a simple Bean. (The previous example shows just how simple a Bean can be.)

Portable

Since JavaBeans components are built purely in Java, they are fully portable to any platform that supports the Java run-time environment. All platform specifics, as well as support for JavaBeans, are implemented by the Java virtual machine. You can be sure that when you develop a component using JavaBeans it will be usable on all of the platforms that support Java (version 1.1 and beyond). These range from workstation applications and web browsers to servers, and even to devices such as PDAs and set-top boxes.

Leverages the Strengths of the Java Platform

JavaBeans uses the existing Java class discovery mechanism. This means that there isn't some new complicated mechanism for registering components with the run-time system.

The Java class libraries provide a rich set of default behaviors for components. Use of Java Object Serialization is one example, a component can support the persistence model by implementing the `java.io.Serializable` interface. By conforming to a simple set of design patterns, you can expose properties without doing anything more than coding them in a particular style.

Flexible Build-Time Component Editors

The JavaBeans architecture also allows you to associate a custom editor with your component. If the task of setting the property values and behaviors of your component is complicated, it may be useful to create a component wizard that guides the user through the steps. The size and complexity of your component editor is entirely up to you.

Q8. Define Properties, Methods, and Events.

Properties are attributes of a Bean that are referenced by name. These properties are usually read and written by calling methods on the Bean specifically created for that purpose.

The methods of a Bean are just the Java methods exposed by the class that implements the Bean. These methods represent the interface used to access and manipulate the component. Usually, the set of public methods defined by

the class will map directly to the supported methods for the Bean, although the Bean developer can choose to expose only a subset of the public methods. Events are the mechanism used by one component to send notifications to another. One component can register its interest in the events generated by another. Whenever the event occurs, the interested component will be notified by having one of its methods invoked. The process of registering interest in an event is carried out simply by calling the appropriate method on the component that is the source of the event. In turn, when an event occurs a method will be invoked on the component that registered its interest. In most cases, more than one component can register for event notifications from a single source. The component that is interested in event notifications is said to be listening for the event.

Q9. Define Introspection, Customization, and Persistence w. r. t. JavaBeans.

Introspection is the process of exposing the properties, methods, and events that a JavaBean component supports. This process is used at run-time, as well as by a visual development tool at design-time. The default behavior of this process allows for the automatic introspection of any Bean. A low-level reflection mechanism is used to analyze the Bean's class to determine its methods. Next it applies some simple design patterns to determine the properties and events that are supported. To take advantage of reflection, you only need to follow a coding style that matches the design pattern. This is an important feature of JavaBeans. It means that you don't have to do anything more than code your methods using a simple convention. If you do, your Beans will automatically support introspection without you having to write any extra code.

This technique may not be sufficient or suitable for every Bean. Instead, you can choose to implement a BeanInfo class.

Customization

When you are using a visual development tool to assemble components into applications, you will be presented with some sort of user interface for customizing Bean attributes. These attributes may affect the way the Bean operates or the way it looks on the screen. The application tool you use will be able to determine the properties that a Bean supports and build a property sheet dynamically. This property sheet will contain editors for each of the properties supported by the Bean, which you can use to customize the Bean to your liking. The Beans class library comes with a number of property editors for common types such as float, boolean, and String.

Customizers are also kept separate from the Bean class so that it is not a

burden to the Bean when it is not being customized. This idea of separation is a common theme in the JavaBeans architecture. A Bean class only has to implement the functionality it was designed for; all other supporting features are implemented separately.

Persistence

It is necessary that Beans support a large variety of storage mechanisms. This way, Beans can participate in the largest number of applications. The simplest way to support persistence is to take advantage of Java Object Serialization. This is an automatic mechanism for saving and restoring the state of an object. Java Object Serialization is the best way to make sure that your Beans are fully portable, because you take advantage of a standard feature supported by the core Java platform.

Q10. What is the difference between a JavaBeans and an instance of a normal Java class?

The difference in Beans from typical Java classes is introspection. Tools that recognize predefined patterns in method signatures and class definitions can “look inside” a Bean to determine its properties and behaviour. A Bean’s state can be manipulated at the time it is being assembled as a part within a larger application. The application assembly is referred to as design time in contrast must follow a certain pattern for introspection tools to recognize how Beans can be manipulated, both at design time, and run time.

CHAPTER 12

IMPORTANT QUESTIONS(PRACTICAL)

Q1. Write a program which calculates the size of a given file and then renames that file to another name.

```
//program
import Java.io.*;
{
public static void main(String args[])
{
File f1 = new File("out1.txt");
File f2 = new File("out2.txt");
if(f1.exists())
{
System.out.println(f1 + "File exists");
System.out.println("Size of file is: " + f1.length() + "bytes" );
F1.renameTo(f2);
System.out.println("Rename file is: " + f2);
System.out.println("Deleting file is: " + f1);
f1.delete();
}
else
System.out.println("f1 " + "file is not existing");
}
}
```

Output:

Out1.txt File exists
Size of file is: 22bytes
Rename file is: out2.txt
Deleting file : out1.txt

Q2. Write a program to print all primitive data values into a File using FileWriter stream wrapped with a Printwriter.

This program use FileWriter stream wrapped with a Printwriter to write binary data into file.

```
//program
    import Java.io.*;
public class PrimitivesToFile_Appl
{
    public static void main(String args[])
    {
        try
        {
            FileWriter filewriter = new FileWriter("prim.txt");
            // Wrap the PrintWriter with a PrintWriter
            // for sending the output to the file
            PrintWriter printWriter = new PrintWriter (filewriter);
            boolean b=true;
            byte   by=127;
            short  s=1111;
            int    i=1234567;
            long   l=987654321;
            float  f=432.5f;
            double d=1.23e-15;
            printWriter.print(b);
            printWriter.print(by);
            printWriter.print(s);
            printWriter.print(i);
            printWriter.print(l);
            printWriter.print(f);
            printWriter.print(d);
            printWriter.print();
        }
        catch (Exception e )
        {
            System.out.println("IO erro" + e);
        }
    }
}
```

Output:

File prim.txt will contain : true 1271111234567987654321432.51.23E-15

Q3. How should you append data to the end of a file? Show the constructor for the class you would use and explain your answer.

First let us use the FileWriter and BufferedWriter classes to append data to the

end of the text file. Here is the `FileWriter` constructor, you pass in true value in second argument to write to the file in append mode:

```
FileWriter writer = new FileWriter (String filename, boolean append);
```

An alternate answer is to use `RandomAccessFile` and skip to the end of the file and start writing.

```
RandomAccessFile file = new RandomAccessFile(datafile, "rm");
File.skipBytes(int)file.length(); //skip to the end of the file
File.writerBytes("Add this text to the end of datafile");
//writer at the end the of the file
file.close();
```

Q4. Write a program to read text from the keyboard and write the text to a file called junk.txt.

```
//program
import Java.io.*;
class FileWriterDemo
{
    public static void main(String[] args) throws IOException
    {
        //open keyboard for input
BufferedReader      stdin      =      new      BufferedReader(new
InputStreamReader(System.in));
//output file 'junk.txt'
String s = "junk.txt";
File f = new File(s);
if (f.exists())
{
System.out.print("Overwrite" + s + " (y/n)? ");
if(!stdin.readLine().toLowerCase().equals("y"))
return;
}
PrintWriter outFile = new PrintWriter(new BufferedWriter(new FileWriter(s)));
System.out.println("Enter some text on the keyboard... ");
System.out.println("^z to terminate");
String s2;
While ((s2 = stdin.readLine())!= null)
outFile.println(s2);
outFile.close();
}
```

Q5. Create a file test.txt and open it in read write mode. Add 4 lines to it and randomly read from line 2 to 4 and line 1.

```
//program
import Java.lang.System;
import Java.io.RandomAccessFile;
import Java.io.IOException;
public class RandomIOApp
{
    public static void main(String args[]) throws IOException
    {
        RandomAccessFile = new RandomAccessFile("test.txt", "rw");
        file.writeBoolean(true);
        file.writeInt(123456);
        file.writeChar('j');
        file.writeDouble(1234.56);
        file.seek(1);
        System.out.println(file.readInt());
        System.out.println(file.readChar());
        System.out.println(file.readDouble());
        file.seek (0);
        System.out.println(file.readBoolean());
        file.close();
    }
}
```

Output:

123456
j
1234.56
true

Q6. Write a Java program to calculate the sum of the digits of a number. The sum should be calculated recursively to get a single digit sum. (For example, if input is 189, then the result will be 9).

The input should be accepted from the command line.

```
import java.awt.*;
import java.io.*;
class digit
{
    public static void main(String args[])
    {
```

```
int sum;
int n=Integer.parseInt(args[0]);
sum=calcsum(n);
System.out.println("The sum is"+sum);
}
static int calcsum(int n)

{
int s;
if(n==0)
return 0;
else
s=n%10+calcsum(n/10);

return s;

}
```

Q7. Write a program in Java that enters a string at the command line. If the string is not equal to “BCA” then a user-defined exception is thrown called “No match”. Also do a proper documentation of the program.

```
import java.awt.*;
import java.io.*;
class MyException extends Exception{
private String a;
MyException(String s){
a=s;}
public String toString()
{
return "MyException:String should be BCA"+a;
}
}
class except
{
static void reply(String s)
{
if(s.compareTo("BCA")!=0)
throw new MyException("No Match");
System.out.println("Alright");
```

```
}

public static void main(String args[]) throws IOException
{
    BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
    String str;
    System.out.println("Enter the string");
    str=br.readLine();
    try{
        reply(str);
    }
    catch(MyException e){
        System.out.println(e);
    }
}
```

Q8. Write the output of the following program and justify your answer:

```
class Demo {
    static void f( ) {
        Try {
            System.out.println ("Inside f( )");
            throw new RuntimeException ("demo");
        }
        finally { System.out.println ("f( ) finally");
        }
    }

    Static void g( ) { try
    { System.out.println ("Inside g( )");
    return;
    }
    finally {System.out.println ("g( ) finally")
    }
}

public static void main(String args[])
{
    try {
        f( );
    }
    catch (Exception e)
    { System.out.println ("Exception caught");
    }
```

```
g( );
}
}
OUTPUT :
inside f()
f() finally
Exception caught
inside g()
```

The control first goes to main(), where in try/catch block function f() is called. In function f() try/finally block having output statement “Inside f()”. Then an exception is thrown that is caught by finally block having output statement “f() finally” we then return to the function main(), where its catch block prints the output statement “Exception Caught” function g() is called at last in void main. This function has output statement “Inside g()”, which is then printed.

Q9. Implement a class programmer. A programmer has name, company name, working for and basic_salary. Write a default constructor and constructor with parameters to create a programmer object. Further, show the implementation of the constructor method with parameters.

```
import java.awt.*;
class programmer
{
String name;
String working_for;
double bas_salary;
programmer()
{
name="GullyBaba";
working_for="MEWS";
bas_salary=10000;
}
programmer(String nam, String company, double s)
{
name=nam;
working_for=company;
bas_salary=s;
}
void print_company()
{
System.out.println("Working for Company"+working_for);
}
```

```

double gross()
{
double gs=0;
gs=bas_salary*.1+bas_salary*.2+bas_salary*.3;
return(gs);
}
}
class prog
{
public static void main(String args[])
{
programmer p=new programmer();
p.print_company();
System.out.print("Gross Salary is");
System.out.print(p.gross());
}
}

```

Q10. Write the output of the following;

```

class Demo {
public static void main(String args[ ])
{
int I;
int num = 0xFFFFFE;
for (I = 0; I < 2; I++)
{
num = num << 1;
System.out.println(num);
}
}
}

```

1073741816

-32

Q11. Write a recursive function in Java to print the contents of an array of integers.

```

import java.awt.*;
class print1
{
int val[];
print1(int i)
{

```

```

val=new int[i];
}
void printarray(int i)
{
if(i==0)
return;
else
printarray(i-1);
System.out.println("[ "+(i-1)+" ] "+val[i-1]);
}
}
class print{
public static void main(String args[])
{
print1 ob=new print1(10);
int i;

for(i=0;i<10;i++)
ob.val[i]=i+2;
ob.printarray(10);
}
}

```

Q12. Write a Java program to calculate nC_r ,

$$\text{where } {}^nC_r = \frac{n!}{(n - r)! r!}.$$

The input of n and r should be accepted from command line.

```

import java.awt.*;
class combination
{
public static void main(String args[])
{
float f1=1, f2=1, f3=1;
int i=0;
int n=Integer.parseInt(args[0]);
int r=Integer.parseInt(args[1]);
float result=0;
for( i=2;i<=n;i++)
f1=f1*i;
for(i=2,i<=n-r;i++)

```

```
f2=f2*i;
for(i=2;i<=r;i++)
f3=f3*i;
System.out.println("k"+f1+"l"+f2+"n"+f3);
result=f1/(f2*f3);
System.out.println("The combinations are"+result);
}
}
```

Q13. Write a java program to calculate Fibonacci series recursively.

```
import java.awt.*;
import java.io.*;
class fibo
{
public static void main(String args[]) throws IOException
{
int n=0;
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
String str;
System.out.println("Enter how many no:s");
str=br.readLine();
try{
n=Integer.parseInt(str);
}catch(NumberFormatException e)
{
System.out.println("Invalid format");
}
System.out.print("0 1 ");

fib(n-2,0,1);
}
static void fib(int n,int d1,int d2)
{
if(n==0)
return;
else
{
int sum=d1+d2;
System.out.print(sum+" ");
d1=d2;
d2=sum;
}
```

```
fib(n-1,d1,d2);
}
}
}
```

Q14. Implement a student class. A student has a name, stipend, enrolment number. Write a default constructor and a constructor with two parameters (name and enrolment number) and two methods.

- (a) To return the name and enrolment number.
- (b) A method that increases the student's stipend.

Write a small program that tests your class.

```
java.awt.*;
class student1
{
    String name;
    int enrol_no=0;
    double stipend;
    student1()
    {
        name="Dhaval";
        stipend=1000;
    }
    student1(String n,double s)
    {
        name=n;
        stipend=s;
    }
    void print()
    {
        System.out.println(name);
        System.out.println(enrol_no);
    }
    double raise(double d)
    {
        stipend=stipend+d;
        return(stipend);
    }
}
class stud
{
    public static void main(String args[])
}
```

```

{
    student1 e=new student1();
    e.print();
    System.out.println("Now raise equal to 1000");
    System.out.println(e.raise(1000));
}
}

```

Q15. Write a complete Java program to find out whether the string t, is contained in the string t. If so, it returns the offset of the first match. If not, it returns -1.

```

import java.awt.*;
class abc
{
public static void main (String args[])
{
try {
String t = "Hello world";
String t1 = "Hello";
if (t1. indexOf(t) < 0)
{
System.out.println ("string not found");
}
else
{
System.out.println ("String match");
}
}
catch (Exception e)
{
System.out.println ("Some error has found");
}
}
}

```

Q16. Write a Java program that reads in an integer and breaks it into a sequence of individual digits. for example, the input 16384 is displayed as 1 6 3 8 4.

```

import java.io.DataInputStream;
class breakno

```

```

{
public static void main(String args[])
{
int i=0; int j=0;
int remainder=0;
int number=0;
int store[];
store=new int[10];
DataInputStream in=new DataInputStream(System.in);
try
{
System.out.println("Please enter the Number:");
number=Integer.parseInt(in.readLine());
}
catch(Exception e)
{
}
while(number>0)
{
remainder=number%10;
store[i]=remainder;
System.out.println(remainder);
number=number/10;
i=i+1;
for(j=i-1;j>=0;j--)
System.out.println(" "+store[i]);
}
}
}

```

Q17. Write a program in Java to find out whether a given string is a palindrome or not.

```

import java.io.*;
public class Palindrome
{
public static void main(String args[]) throws IOException
{
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
System.out.println("Enter a string :");
StringBuffer str=new StringBuffer(br.readLine());
int len=str.length();
for(int i=0;i<len/2+1;i++)
{
}

```

```
if(str.charAt(i)!=str.charAt(len-i-1))
{
System.out.println("The string is not a palindrome");
System.exit(0);
}
}
System.out.println("The string is a palindrome");
}
```

Q18. Implement a class student. A student has a name and stipend (double data type). Write a default constructor and constructor with two parameters (name and stipend) and two methods:

- * to return the name and stipend and
- * a method that raises the employee's salary. Write a small program that tests your class.

```
import java.io.*;
class student
{
String name;
float stipend;
student()
{
name = " ";
stipend = 0.00;
}
student (String n,float s)
{
name = n;
stipend = s;
}
public void displaysal()
{
System.out.println ("student name = " + name) ;
System.out.println ("student's stipend = "+ stipend);

stipend = (float) stipend + (stipend* 10);
System.out.println ("Raised salary" + stipend);
}
}
class test
```

```
{  
public static void main (String args[] )  
{  
student s = new student ("Anand", 8000f);  
s.displaysal() ;  
  
}  
}
```

Q19. Answer the following questions :

(a) How do you create a text area with 10 rows and 20 columns? How do you insert 2 lines into the text area?

```
import java.awt.*;  
import java.awt.event.*;  
class test_textarea extends Frame implements ActionListener  
{  
static test_textarea t;  
FlowLayout f;  
MyWindowAdapter1 adapter;  
TextArea t_area;  
Button ok;  
public static void main(String args[])  
{  
t=new test_textarea();  
t.show();  
t.setSize(800,400);  
}  
test_textarea()  
{  
adapter=new MyWindowAdapter1(this);  
addWindowListener(adapter);  
t_area = new TextArea(10,20);  
ok=new Button("Click here");  
add(t_area);  
add(ok);  
f=new FlowLayout();  
setLayout(f);  
ok.addActionListener(this);  
}  
public void actionPerformed(ActionEvent ae)  
{
```

```
if(ae.getSource()==ok)
{
String st1="hello!\nhow r u?\n";
t_area.setText(st1);
}
}
}
}

class MyWindowAdapter1 extends WindowAdapter
{
test_textarea t;
public MyWindowAdapter1(test_textarea t1)
{
t=t1;
}
public void windowClosing(WindowEvent we)
{
t.setVisible(false);
System.exit(0);
}
}
```

Q20. How do you create a checkbox group with three items? How do you determine if a box is checked?

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.lang.*;
public class chkbox extends Applet implements ItemListener
{
Checkbox popcorn, icecream, chocolate;
String message;
public void init()
{
popcorn=new Checkbox("popcorn",true);
icecream=new Checkbox("Icecream",false);
chocolate=new Checkbox("Chocolate",false);
add(popcorn);
add(icecream);
add(chocolate);
popcorn.addItemListener(this);
icecream.addItemListener(this);
```

```
chocolate.addItemListener(this);
}
public void itemStateChanged(ItemEvent ie)
{
repaint();
}
public void paint(Graphics g)
{
message="popcorn :" + popcorn.getState();
g.drawString(message,10,100);
message="icecream :" + icecream.getState();
g.drawString(message,10,150);
message= "chocolate :" + chocolate.getState();
g.drawString(message,10,200);
}
}
```

Q21. Write is program that let the user enter text from the text field and then append it to the text areas.

```
import java.awt.*;
import java.awt.event.*;
class test_text extends Frame implements ActionListener
{
static test_text t;
FlowLayout f;
My Window Adapter adapter;
TextField t_field;
TextArea t_area;
Button ok;
public static void main(String args[])
{
t = new test_text();
t.show();
t.setSize(800,400);
}
test_text()
{
adapter=new MyWindowAdapter(this)
addWindowListener(adapter);
t_field = new TextField(25)
```

```

t_area = new TextArea();
ok=new Button("click here");
add(t_field);
add(ok);
f=new FlowLayout();
setLayout(f);
ok.addActionListener(this);

}

public void actionPerformed(ActionEvent ae)
{
    if(ae.getSource()==ok)
    {
        String st1=t_area.getText() + t_field.getText();
        t_area.setText(st1);
    }
}
}

class MyWindowAdapter extends Window Adapter
{
    test_text t;
    public MyWindowAdapter(test_text tl)
    {
        t=tl;
    }
    public void windowClosing(WindowEvent we)
    {
        t.setVisible(false);
        System.exit(0);
    }
}

```

Q22. Write a Java program to count 1 to 1000 numbers using 10 threads.

```

import java.awt.*;
class count
{
    public static int sum=0;
    public static void main(String args[]) throws InterruptedException
    {

```

```
sub_count t1,t2,t3,t4,t5,t6,t7,t8,t9,t10;
t1=new sub_count(1,100);
t1.start();
t2=new sub_count(101,200);
t2.start();
t3=new sub_count(201,300);
t3.start();
t4=new sub_count(301,400);
t4.start();
t5=new sub_count(401,500);
t5.start();
t6=new sub_count(501,600);
t6.start();
t7=new sub_count(601,700);
t7.start();
t8=new sub_count(701,800);
t8.start();
t9=new sub_count(801,900);
t9.start();
t10=new sub_count(901,1000);
t10.start();
t1.join();
t2.join();
t3.join();
t4.join();
t5.join();
t6.join();
t7.join();
t8.join();
t9.join();
t10.join();
System.out.println("Sum is:"+sum);
}
}
class sub_count extends Thread
{
int start_no;
int end_no;
sub_count(int start,int end)
{
start_no=start;
```

```

end_no=end;
}
public void run()
{
int i,tmp=0;
for(i=start_no;i<=end_no;i++)
{
tmp=tmp+i;
}
count.sum=count.sum+ tmp;
}
}

```

Q23. Write a complete Java program to calculate Fibonacci series non recursively.

```

import java.awt.*;
class fibonacci
{
public static void main(String args[])
{
int firstno= 1;
int secondno= 1;
int i=0;
int fibono=0;
System.out.println(firstno);
System.out.println(" ");
System.out.println(secondno);
System.out.println(" ");
for(i=0;i<30;i++)
{
fibono=firstno+secondno;
System.out.println(fibono);
firstno=secondno;
secondno=fibono;
}
}
}

```

Q24. How do you create a checkbox with three choice items? How do you determine if a box is checked?

```
import java.awt.*;
```

```
import java.awt.event.*;
import java.applet.*;
import java.lang.*;
public class chkbox extends Applet implements ItemListener
{
    Checkbox popcorn, icecream, chocolate;
    String message;
    public void init()
    {
        popcorn=new Checkbox("popcorn",true);
        icecream=new Checkbox("Icecream",false);
        chocolate=new Checkbox("Chocolate",false);
        add(popcorn);
        add(icecream);
        add(chocolate);
        popcorn.addItemListener(this);
        icecream.addItemListener(this);
        chocolate.addItemListener(this);
    }
    public void itemStateChanged(ItemEvent ie)
    {
        repaint();
    }
    public void paint(Graphics g)
    {
        message="popcorn :" + popcorn.getState();
        g.drawString(message,10,100);
        message="icecream :" + icecream.getState();
        g.drawString(message,10,150);
        message= "chocolate :" + chocolate.getState();
        g.drawString(message,10,200);
    }
}
```

Q25. Write a program in Java to find the largest and smallest of n numbers stored in an array, where n is a positive integer.

Program to find Max and Min in an Array.

```
class findMAX_MIN
{
    public static void main(String args[])
    {
```

```

int a[]={40,52,75,65,72,75,24,64,73,75,71,52,75,65,72,75,24,64,73,75,71};
int max=a[0];
int min=a[0];
int pmax=0;
int pmin=0;
for(int i=1;i<=a.length-1;i++)
{
    if(max<a[i])
    {
        max=a[i];
        pmax=i;
    }
    if(min>a[i])
    {
        min=a[i];
        pmin=i;
    }
}
System.out.println("MAX=( " + max + ","+pmax+" ) , MIN=( " + min + "," +
pmin +" )");
}
}

```

Q26. Write the output of the following program segments. Also explain step by step processing.

[6]

```

for (int i = 1; i<10; i++) {
    for (int j=1; j<4; j++) {
        if (j == 2)
            break;
        System.out.println(i +","+j);
    }
}

```

What is the difference between ‘throw’ and ‘throws’ ? Explain with proper examples.

- 1,1
- 2,1
- 3,1
- 4,1
- 5,1
- 6,1

7,1
8,1
9,1

So far, you have only been catching exceptions that are thrown by the Java run-time system. However, it is possible for your program to throw an exception explicitly, using the **throw** statement. The general form of throw is shown here;

Throw *Throwableinstance*;

Here, *Throwableinstance* must be an object of type **Throwable** or a subclass, of **Throwable**.

The flow of execution stops immediately after the throw statement; any subsequent statements are not executed. The nearest **try** block is inspected to see if it has a **catch** statement that matches the type of the exception. If it does find a match, control is transferred to that statement. If not, then the next enclosing **try** statement is inspected, and so on. If no matching **catch** is found, then the default exception handler halts the program and prints the stack trace.

```
class exc_throw
{
    public static void main(String args[])
    {
        try
        {
            throw new ArithmeticException("hello");
        }
        catch(ArithmeticException e)
        {
            System.out.println("The exception catched is : " + e);
        }
    }
}

*****
```

Output is:

The exception catched is : java.lang.ArithmaticException: hello

Here, **new** is used to construct an instance of **NullPointerException**. All of Java's built-in run-time exceptions have two constructors: one with no parameter and one that takes a string parameter.

Throws

If a method is capable of causing an exception that it does not handle, it must specify this behavior so that callers of the method can guard themselves against that exception. You do this by including a throws clause in the method's

declaration. A **throws** clause lists the types of exceptions that method might throw.

This is the general (*parameter-list*) throws exception-list

```
{
    // body of method
}
```

Here, *exception-list* is a comma-separated list of the exceptions that a method can throw.

Following is an example of an incorrect program that tries to throw an exception that it does not catch. Because the program does not specify a **throws** clause to declare this fact, the program will not compile.

```
// This program contains an error and will not compile.
```

```
class Throws Demo {
    Static void throwOne( ) {
        System.out.println ("Inside throwOne");
        Throw new IllegalAccessException ("demo");
    }
    public static void main (String args[ ]) {
        throwOne ( );
    }
}
```

To make this example compile, you need to make two changes. First you need to declare that **throwOne()** throws **IllegalAccessException**. Second, **main()** must define a **try/catch** statement that catches this exception.

The corrected example is shown here:

```
//This is now correct.
```

```
class ThrowsDemo {
    Static void throwOne( ) throws IllegalAccessException {
        System.out.println("Inside throwOne.");
        Throw new IllegalAccessException("demo");
    }
    public static void main(String args[ ]) {
        try {
            throwOne();
        } catch(IllegalAccessException e) {
            System.out.println("Caught " + e);
        }
    }
}
```

Here is the **output** generated by running this example program;

Inside throwOne

Caught Java.Lang.IllegalAccessException: demo

CHAPTER 13

IMPORTANT QUESTIONS(THEORY)

Q1. Write comparison between Interface and Class.

Using the keyword interface, you can fully abstract a class interface from its implementation. That is, using interface, you can specify what a class must do, but not how it does it. Interfaces are syntactically similar to classes, but they lack instance variables, and their methods are declared without anybody. In practice, this means that you can define interfaces which don't make assumptions about how they are implemented. Once it is defined, any number of classes can implement an interface. Also, one class can implement any number of interfaces.

To implement an interface, a class must create the complete set of methods defined by the interface. However, each class is free to determine the details of its own implementation. By providing the interface keyword, Java allows you to fully utilize the “one interface, multiple methods” aspect of polymorphism.

Interfaces are designed to support dynamic method resolution at run time. Normally, in order for a method to be called from one class to another, both classes need to be present at compile time so the Java compiler can check to ensure that the method signatures are compatible. This requirement by itself makes for a static and non-extensible classing environment. Inevitably in a system like this, functionality gets pushed by higher and higher in the class hierarchy so that the mechanisms will be available to more and more subclasses. Interfaces are designed to avoid this problem. They disconnect the definition of a method or set of methods from the inheritance hierarchy. Since interfaces are in a different hierarchy from classes, *it is possible for classes that are unrelated in terms of the class hierarchy to implement the same interface. This is where the real power of interfaces is realized.*

Interfaces add most of the functionality that is required for many applications which would normally resort to using multiple inheritance in a language such as C++.

Q2. What is an access specifier? Discuss the available access specifier in Java.

Access control is controlling visibility of a variable or method. When a method

or variable is visible to another class, its methods can reference the method or variable. There are four levels of visibility that are used. Each level is more restrictive than the other and provides more protection than the one preceding it.

Public : Any method or variable is visible to the class in which it is defined. If the method or variable must be visible to all classes, then it must be declared as public. A variable or method with public access has the widest possible visibility. Any class can use it.

Package : Package is indicated by the lack of any access modifier in a declaration. It has an increased protection and narrowed visibility and is the default protection when none has been specified.

Protection : This specifier is a relationship between a class and its present and future subclasses. The subclasses are closer to the parent class than any other class. This level gives more protection and narrows visibility.

Private : It is narrowly visible and the highest level of protection that can possibly be obtained. Private methods and variables cannot be seen by any class other than the one in which they are defined. They are extremely restrictive but are most commonly used. Any representation unique to the implementation is declared private. An object's primary job is to encapsulate its data and limit manipulation. This is achieved by declaring data as private.

Q3. Difference between Method overloading and Overriding method

Method Overloading	Method Overriding
<ul style="list-style-type: none"> (i) Overloaded methods supplement each other. (ii) Overloading methods may have different argument lists of different types. (iii) The return type may be different for overloaded methods (iv) Since overloading methods are essentially different methods, there is no restriction on exceptions they can throw. 	<ul style="list-style-type: none"> (i) An overriding method replaces the method it overrides. (ii) An overriding method must have argument lists of identical type and order. (iii) The return type must be same as that of overridden method. (iv) The overriding method must not throw checked exceptions which cannot be thrown by the original method.

Q4. List the Java core libraries and describe their functionalities.

Java core libraries permit flexibility in interface design and programming.

We have following libraries :

1. Java.lang : It has classes

this package permits data conversion, threading, string handling etc.

2. Java.util : It has classes

We can work on arrays, collections, linked list, trees, stacks, vectors, with this package.

3. Java.io : It has classes that are used for I/O at byte and character level. We can serialize classes with it. We can use networking also.

4. Java.awt : It has classes for designing and managing windows. It helps in making buttons, menus, checkboxes, adding images etc.

Q5. List the modifiers and describe their purpose.

Default modifier : *In the absence of any modifiers fields are associated with objects and are visible to all classes in a package, and accessible by a subclass in the same package only.*

public modifier : *A public variable or function can be accessed in any other class.*

private modifier : *A private variable or function can only be accessed its own class.*

protected modifier : *A degree of hiding lie between public and private, subclasses and classes in the same package have access to these items.* — deprecated since any class can claim to be a member of the same package! Note for C++ programming, the C++ protected is the Java 1.0 *private_protected*.

private protected : ‘A private protected field or function can be accessed by any subclass, but can not be used otherwise’, — in Java 1.0 only, and equivalent to the C++ protected.

static : *A variable or functions associated with a class rather than with the objects of the class.*

Note for C programmers. In C a static variable declared in a functions is the same variable each time you call it. This is not available in Java. In C a static variable declared outside a function is shared by the set of functions in the same file and following the declaration an hidden from other functions. In Java a static variable is shared by all functions in that class. Whether other classes can access it depend on whether it is private, public, and/or protected.

A Java variable that is nonstatic is something that is not found in C. Each object in the class has its own variable, and all functions refer to that particular variable. The variable is created as part of the object and deleted with it.

In Java instance variables are the same each time you call them. A variable or function is declared to be static when it is associated with the class of objects rather than a particular object in that class. This means that a static function can not refer any data in an object - because it has no object. It also means that static data is essentially shared in common by every object but only stored in a single place. If you are in doubt about whether some data should be static or not - ask yourself how many times it occurs: once per class(static) or once per object(not static) or many time per object (its in a different class!).

final : *once initialized it can not be changed.* A variable(a field of a class or object) is declared *final* when you want it to keep its initial value for ever. It makes it a constant variable. A class is declared as *final* if it can not be extended.

native : *Preprogrammed in another language and running as machine code.*

synchronized : *Only one thread can execute this at a time.* A method or block of code is marked as *synchronized* if only one thread of control can be in it at a time. Other threads are locked out. The code is reentrant. It protects resources from interference by multiple access at one time.

abstract method : *A method that must exist for objects in a class but is fully defined only in subclasses,* abstract::class=*A class with one or more abstract methods.* abstract::=not concrete, not yet implemented, deferred, prototypical. An abstract method is one that is defined in classes derived from this class. All subclasses have a version as defined in the base class or also declare it as abstract. Certain methods can not be abstract: constructors, static, private, final, native, synchronized, plus those that override superclass methods. An abstract method makes the whole class *abstract*. A class that inherits an abstract method and does not override it is also an abstract class. An abstract class can not be used to construct new objects. You can only call an abstract method via an object in a class that has extended the abstract class and defined all the abstract methods.

threadsafe : *If another thread executing this code at the same time can not change the value of a variable then the variable is threadsafe and the compiler may do clever things with it to make the code faster or smaller..*

transient : something that does last longer than a function call. If an object can exist longer than a given applet....is persistent.... then its transient data does not have to be preserved when a function exits.

Q6. State true or false with reasons:

Every element in the array has the same type.

TRUE

Reason : by definition array is a collection of data of same type only.

Array size is fixed after it is created.

TRUE

Reason : A runtime error occurs if we try to

Java supports multiple inheritance.

FALSE

Every java class has a default constructor.

TRUE

Reason : Java uses default constructor to initialize the object at runtime.

The static keyword is used to access a program code without instantiating the class to which that code belongs.

TRUE

Reason : If you need to do computation in order to initialize your **static** variables, you can declare a **static** block which gets executed exactly once, when the class is first loaded the following example shows a class that has a static method, some **static** variables, and a **static** initialization block:

```
class Stat_Class
{
    static int counter;
    Stat_Class()
    {
        counter++;
    }
    //This Block initialized only once during entire life of Stat_Class
    static
    {
        System.out.println("Static block initialized");
        counter=1;
    }
}

class Static_Demo
{
    public static void main(String args[])
    {
        www.DoeaccOnline.com
```

```

{
    Stat_Class stat1=new Stat_Class();
    System.out.println(stat1.counter);
    // We can also access the static variable counter with
    //out using Object name, for example Stat_Class.counter
    //which we done next

    Stat_Class stat2=new Stat_Class();
    System.out.println(Stat_Class.counter);
}
}

```

Outside of the class in which they are defined, **static** methods and variables can be used independently of any object. To do so, you need only to specify the name of their class followed by the dot operator. For example, if you wish to call a **static** method from outside its class, you can do so by using the following, general form:

Classname.Method();

Here, classname is the name of the class in which the **static** method is declared. As you can see, this format is similar to that used to call non-**static** methods through object-reference variables. A **Static** variable can be accessed in the same way-by use of the dot operator on the name of the class. This is how Java implements a controlled version of global function and global variables. This whole concept permits us to call the main() function from outside Java classes.

Q7. What happens when your program attempts to access an array element with an invalid index ?

If we attempt to go out of valid indexes it will create a runtime error to be prompted. It will generate `ArrayIndexOutOfBoundsException`. So it is advisable to handle such an error in the program otherwise it can lead to an inefficient programming style/code.

Q8. How do you set colours and fonts in a graphics context? How do you find the current colour and font style?

We can set fonts by using `setFont()` method in a `Graphics` class.

```
Public void paint(Graphics g) {
```

```
    Font f = new font("Times New Roman" font.PLAIN, 36);
    g.setFont();
    g.drawString("This is a big font: " 10, 100);
}
```

To set a colour, we use `setColor()` method.

To find the current colours and font style we use :

`getColor()` and `getStyle()` method.

Q9. Describe the `paint()` method. Where is it defined? How is it invoked?

The `paint()` method is called each time your applet's output must be redrawn. This situation can occur for several reasons. For example, the window in which the applet is running may be overwritten by another window and then uncovered. Or the applet window may be minimized and then restored. `paint()` is also called when the applet begins execution. Whatever the cause, whenever the applet must redraw the output, `paint()` is called. The `paint()` method has one parameter of type `Graphics`. This parameter will contain the graphics context, which describes the graphics environment in which the applet is running. This context is used whenever output to the applet is required. `paint()` is defined by the AWT(Abstract Window Toolkit) Component class. To use this method, it is necessary to import the `Graphics` class as *import java.awt.Graphics;*. It is invoked by overriding `paint()` method in the class and using `repaint()` method to call it when needed.

Q10. Differentiate between the Window and Panel

The `Window` class creates a top-level window. A top-level window is not contained within any other object; it sits directly on the desktop. Generally, you won't create `Window` objects directly. Instead, you will use a subclass of `Window` called `Frame`, described next.

While the `Panel` class is a concrete subclass of `Container`. It doesn't add any new methods; it simply implements `Container`. A **Panel** may be thought of as a recursively nestable, concrete screen component. **Panel** is the superclass for `Applet`. When screen output is directed to an applet, it is drawn on the surface of a `Panel` object. In essence, a **Panel** is a window that does not contain a title bar, menu bar, or border. This is why you don't see these items when an applet is run inside a browser. When you run an applet using an applet viewer, the applet viewer provides the title and border.

Other components can be added to a **Panel** object by its `add()` method (inherited from `Container`). Once these components have been added, you can position and resize them manually using the `setLocation()`, `setSize()`, or `setBounds()` methods defined by `Component`.

Q11. Differentiate between the Thread and Process.

Unlike most other computer languages, JAVA provides built-in support for multithreading programming. A multithreading program contains two or more parts that can run concurrently. Each point of such program is called thread;

specialized form of multitasking.

You are most certainly acquainted with multitasking, because it is supported by virtually all modern Operating Systems. However, there are two distinct types of multitasking; process-based & thread-based. It is important to understand the difference between two. For most readers, process-based multitasking is the more familiar form. A process is, in essence, a program that is executing. Thus process-based multitasking is the feature that allows your computer to run two or more programs concurrently. For example, process-based multitasking enables you to run the Java compiler at the time your text editor is also running. In process-based multitasking, a program is the smallest unit of code that can be dispatched by the scheduler.

In a thread-based multitasking environment, the thread is the smallest unit of dispatchable code. This means that a single program can perform two or more tasks simultaneously. For instance a text editor can format text at the same time that it is printing, as long as these two actions are being performed by two separate threads. Thus, process-based multitasking deals with the “big-picture”, and thread-based multitasking handles the details.

Multitasking threads require less overhead than multitasking process. Processes are heavy weight task that require their own separate address spaces. Inter process communication is expensive and limited. Context switching from one process to another is also costly. Threads, on the other hand, are lightweight. They share the same address space and cooperatively share the same heavyweight process. Inter thread communication is less expensive and context switching from one thread to the next is at low cost. While Java programs make use of process-based multitasking environments, process-based multitasking is not under the control of Java. However, multithreaded multitasking is.

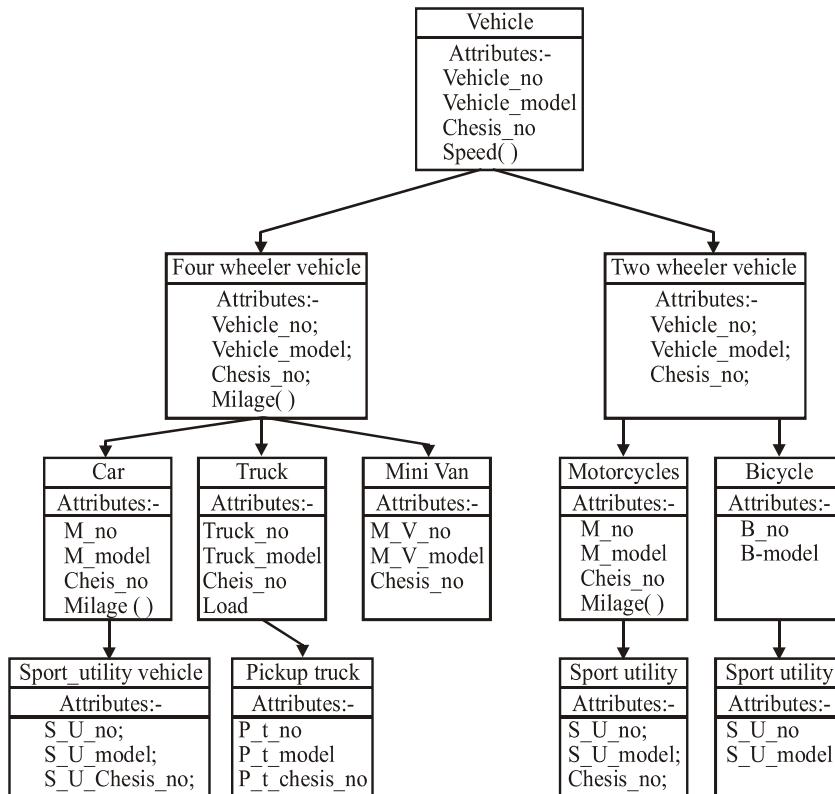
Multithreading enables you to write very efficient programs that make maximum use of the CPU, because idle time can be kept to minimum, this is especially important for the interactive, networked environment in which Java operates, because idle time is common. For example, the transmission rate of data over a network is much slower than the rate at which the CPU can process it.

If you have programmed for OS such as Windows 98 or Window NT, then you are already familiar with multithreaded programming. However, the fact that Java threads manager makes multithreading especially convenient, because many of the details are handled for you.

Q12. In an object oriented traffic simulation system, we have the following classes :

- **Vehicle**
- **Car**
- **Truck**
- **Pick up truck**
- **Sport utility vehicle**
- **Mini van**
- **Bicycle**
- **Motorcycle**

Draw an inheritance diagram that shows the relationship between these classes. Identify methods, variables in each class.



Q13. What is the difference between radio buttons and check boxes?

We can create a set of mutually exclusive check boxes in which one and only one check box in the group can be checked at any one time. These check boxes are often called radio buttons, because they act like the station selector on a car radio—only one station can be selected at any one time. To create a set

of mutually exclusive check boxes, you must first define the group to which they will belong and then specify that group when you construct the check boxes. Check box groups are objects of type CheckboxGroup. Only the default constructor is defined, which creates an empty group.

You can determine which check box in a group is currently selected by calling getSelectedCheckbox(). You can set a check box by calling setSelectedCheckbox(). These methods are as follows:

```
Checkbox getSelectedCheckbox();
void setSelectedCheckbox(Checkbox which);
```

Here, which is the check box that you want to be selected. The previously selected check box will be turned off.

While a check box is a control that is used to turn an option on or off. It consists of a small box that can either contain check mark or not. There is a label associated with each check box that describes what option the box represents. You change the state of a check box by clicking on it. Check boxes can be used individually or as part of a group. Check boxes are objects of the Checkbox class.

When a group of check boxes are made then it is called Radio Buttons. And as individually it can be used to select or deselect an option.

Q14. What is the difference between a menu, menu bar and menu item?

A top-level window can have a menu bar associated with it. A menu bar displays a list of top-level menu choices. Each choice is associated with a drop-down menu. This concept is implemented in Java by the following classes: Menu,MenuBar, and MenuItem. In general, a menu bar contains one or more Menu objects. Each Menu object contains a list of MenuItem objects. Each MenuItem object represents something that can be selected by the user. Since Menu is a subclass of MenuItem, a hierarchy of nested submenus can be created. It is also possible to include checkable menu items. These are menu options of type CheckboxMenuItem and will have a check mark next to them when they are selected.

To create a menu bar, first create an instance ofMenuBar. This class only defines the default constructor. Next, create instances of Menu that will define the selections displayed on the bar. Following are the constructors for Menu:

```
Menu();
Menu(String optionName);
Menu(String optionName, boolean removable);
```

Here, optionName specifies the name of the menu selection. If removable is true, the pop-up menu can be removed and allowed to float free. Otherwise, it will remain attached to the menu bar. (Removable menus are implementation-dependent.) The first form creates an empty menu.

Individual menu items are of type MenuItem. It defines these constructors:

MenuItem();

MenuItem(String itemName);

MenuItem(String itemName, MenuShortcut keyAccel);

Here, itemName is the name shown in the menu, and keyAccel is the menu shortcut for this item.

Once you have added all items to a Menu object, you can add that object to the menu bar by using this version of add() defined byMenuBar:

Menu add(Menu menu);

Here, menu is the menu being added. The menu is returned.

Menus only generate events when a item of type MenuItem or CheckboxMenuItem is selected.

Q15. What is a integrated programming environment?

If you are new to Java programming, you are probably starting by using notepad. When you want to compile and run a program, you need to open a DOS window and type javac MyProgram.java and java MyProgram. Also, what you type is what you get: notepad does not help you any way with Java syntax or design. This is fine if you are just starting, and even experts sometimes use command-line Java options. However, a good Java-savvy editor or Integrated Programming Environment (IPE) will make the job of creating Java code a lot easier. **Editors are simpler :** they generally highlight the Java syntax, indent for you, balance your parentheses and braces, and let you compile from within the editor. But that's about it: they don't write code, integrate tightly with the compiler or app server, or have graphical Java development tools. Provides you Integrated Programming Environment (IPE), which makes the programming easier by saving time in formatting commands, syntax etc. An integrated programming environment is open if it is able to grow up according to the user needs. In other words, it should take into account the possibility of integrating new tools, and to be dynamically configured by a user.

Q16. What method do you use to obtain an input character from a keyboard?

To input a character from a keyboard following procedure has to be followed:
Create an object of DataInputStream class.

Here,

```
DataInputStream in=new DataInputStream(System.in);
```

Now,

We can read a string or character:

```
String a=in.readLine();
```

OR

We use read() method to obtain an input character from keyboard. It reads a character from the input stream and return it as an integer value. We used read() method of ‘System class’ which is defined in ‘java.lang’ package via a predefined stream variable called ‘in’ to obtain an input from the keyboard i.e., System.in.read().

Example:-

```
Char ch;  
Ch = (Char) System.in.read( );
```

Q17. Describe important methods in a container and in a panel.

Methods in container :

A container is a subclass of component. Container allows us to create grouping of objects on the screen.

- (i) protected container()
- (ii) public int getComponentCount()
- (iii) public int countComponents()

Methods in Panel :-

The panel class provides a generic container within an existing display area. It is the simplest of all the container.

- (i) public Panel ()
Creates a panel with a layout manager of flowlayout.
- (ii) public Panel (LayoutManager layout)
- (iii) public void addNotify()

Q18. Draw an inheritance diagram for the following classes:

Person

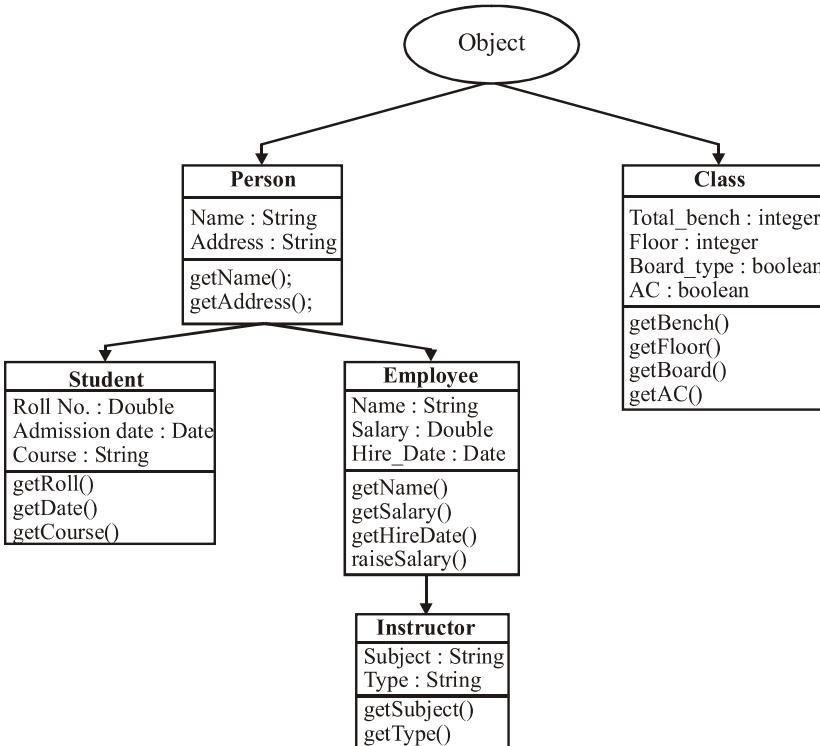
Employee

Student

Instructor

Classroom

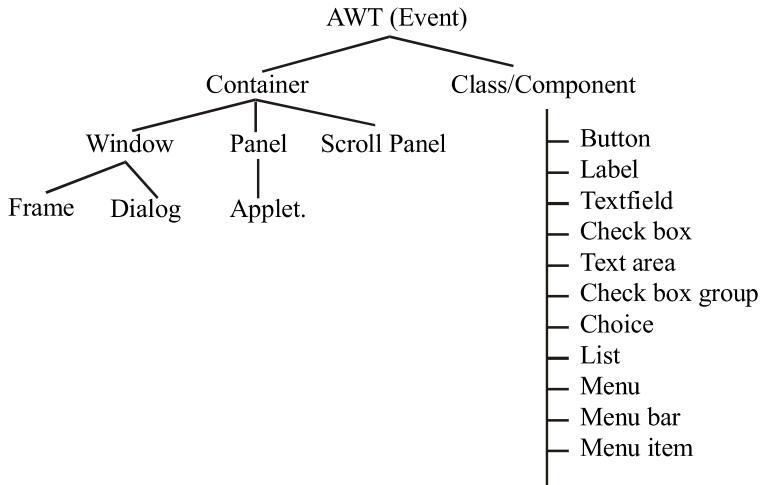
List the important methods and variables in the classes .



Q19. Draw the AWT class hierarchy and discuss important classes and its methods.

The Abstract Windowing Toolkit (AWT) is an API that is responsible for building the Graphical User Interface (GUI). It is a part of the Java Foundation Classes (JFC). AWT includes a rich set of user interface components, a powerful event handling model, graphics and image tools, layout managers and support for data transfer using cut and paste through clipboards. AWT also supports javaBeans architecture. Every AWT component is a simple bean. The `java.awt` package contains all classes for creating user interfaces and for painting graphics and images.

AWT Class hierarchy



Methods in Component :

(i) Public static final float BOTTOM_ALIGNMENT

The BOTTOM_ALIGNMENT indicates that the component should align itself to the bottom of its available space.

(ii) Public static final float CENTER_ALIGNMENT constant indicates that the component should align itself to the middle of its available space. This constant represent both the horizontal and vertical center.

(iii) Public static final float LEFT_ALIGNMENT

(iv) Public static final float RIGHT_ALIGNMENT

(v) Public static final float TOP_ALIGNMENT

Methods in Container :

A container is a subclass of component. Container allows us to create grouping of object on the screen.

(i) Protected Container()

(ii) Public int getComponents()

(iii) Public int countComponents()

Methods in Panel :

The Panel class provides a generic container within an existing display area. It is the simplest of all the container.

(i) Public Panel()

Creates a panel with a layout manager of type flowlayout

(ii) Public Panel (layoutmanager layout)

```
public void addNotify ()
```

Methods in Frame :

Frame encapsulates what is commonly thought of as a “window”. It is a subclass of Window and has a title bar, menu bar, borders, and resizing corners.

(i) Frame()

Frame(String title)

These are Frame's constructors.

(ii) void setSize(int newWidth, int newHeight)

To setting the Window's Dimensions.

(iii) void setVisible(boolean visibleFlag)

To hide or show a window.

Q20. Why do we need to use the layout manager?

The need of Layout manager is to automatically arrange the controls within a window. Imagine a window in which we want to put 6 buttons as the size of the window is fixed all the buttons can't be fixed in a single row then layout manager allows to position the controls at an appropriate locations within the window.

Q21. What is the need of a layout manager? Describe flowlayout. How do you create a flowlayout manager? How do you add a component to a flow-layout container? Is the number of components to be added to a flow-layout container limited?

The need of Layout manager is to automatically arrange the controls within a window. Imagine a window in which we want to put 6 buttons as the size of the window is fixed all the buttons can't be fixed in a single row then layout manager allows to position the controls at an appropriate locations within the window.

Flow layout is the default layout manager. Flow layout implements a simple layout style, which is similar to how words flow in a text editor. Components are laid from the upper left corner, left to right and top to bottom. When no more components fit on a line, the next one appears on the next line. A small space is left between each component, above and below, as well as left and right.

A layout manager is an instance of any class that implements the Layout Manager interface. The layout manager is set by the setLayout() method. To set the flow layout manager the following method has to be implemented:

Void setLayout(LayoutManager layoutObj)

In this program we can use :

```

setLayout( new FlowLayout() );
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
public class flowlayout extends Applet implements ItemListener
{
    String msg="";
    Checkbox popcorn,icecream,chocolate;
    Public void Init()
    {
        setLayout(new FlowLayout(FlowLayout.LEFT));
        popcorn=new Checkbox("Popcorn",true)
        icecream=new Checkbox("Icecream",false)
        chocolate=new Checkbox("Chocolate",false)add(popcorn);
        add(icecream);
        add(chocolate);
        popcorn.addItemListener(this);
        icecream.addItemListener(this);
        chocolate.addItemListener(this);
    }
    public void itemStateChanged(ItemEvent ie)
    {
        repaint();
    }
    public void paint(Graphics g)
    {
        g.drawString(popcorn.getState(),10,100);
        g.drawString(icecream.getState(),10,150);
        g.drawString(chocolate.getState(),10,200);
    }
}

```

No there is no limit on the number of components that can be added to the flow layout container.

Q22. Write difference between Extends and interface.

- (i)** Extends is used in Inheritance. Interface is keyword used to declare interfaces (whose implementation is not defined).
- (ii)** Using Extends only one class can be inherited (i.e. Single Inheritance) while through interface we can inherit as many as interfaces (i.e. Multiple Inheritance).
- (iii)** In using Interface more code has to be written, debugged, and maintained than in using Extends.

(iv) Interfaces are slower than Inheritance through Extends.

To inherit a class, Java uses **extends** keyword. To inherit a class, syntax for the derived class declaration is

```
class<derived class name> extends <base class>
```

```
{  
    //body of the class  
}
```

Java doesn't support the multiple inheritance, so a derived class can only inherit one class, but Java fully support multilevel inheritance e.g., given three classes called A, B, and C, C can be a subclass of B, which is a subclass of A. Here C inherits all the traits of B, and A classes.

The example shows a short program, here class B inherits class A and all members of class A included in class B. That is, object of the class B can access any member of class A as it access its own member.

```
class A
```

```
{  
    int i;  
    void get_A_Val()  
    {  
        System.out.println("Value of i is : " + i);  
    }  
}
```

```
class B extends A
```

```
{  
    int j;  
    void get_B_Val()  
    {  
        get_A_Val(); // get_A_val() is now member of B  
        System.out.println("Value of j is : " + j);  
    }  
}
```

```
class mainClass
```

```
{  
    public static void main(String args[])  
    {  
        B objB; //Object of B declared  
        objB=new B(); //Object is instantiated  
  
        objB.j=10;  
  
        objB.i=5; //objB accessing i as its own member  
    }  
}
```

```

        objB.get_B_Val();
    }
}

```

Even though A is a superclass for B, it is also a completely independent, stand-alone class. Being a superclass for a subclass does not mean that the superclass cannot be used by itself. Further, a subclass can be a superclass for another subclass.

Through the use of the **interface** keyword, Java allows you to fully abstract the interface from its implementation. Using interface, you can specify a set of methods which can be implemented by one or more classes. The interface, itself, does not actually define any implementation. Although they are similar to abstract classes, interfaces have an additional capability: A class can implement more than one interface. By contrast, a class can only inherit a single super class (abstract or otherwise).

Packages and interfaces are two of the basic components of a Java program. An interface is defined much like a class. This is the general form of an interface.

```

access interface name {
    return-type method-name1(parameter-list);
    return-type method-name2(parameter-list);
    type final-varname1 = value;
    type final-varname2 = value;
    // ...
    return-type method-nameN(parameter-list);
    type final-varnameN = value;
}

```

Q23. Describe pass by reference and pass by value through example.

In general, there are two ways that a computer language can pass an argument to a subroutine. The first way is *call-by-value*. This method copies the *value* of an argument into the formal parameter of the subroutine. Therefore, changes made to the parameter of the subroutine have no effect on the argument used to call it. The second way an argument can be passed is *call-by-reference*. In this method, a reference to an argument (not the value of the argument) is passed to the parameter. Inside the subroutine, this reference is used to access the actual argument specified in the call. This means that changes made to the parameter will affect the argument used to call the subroutine. As you will see, Java uses both methods, depending upon what is passed.

In Java, when you pass a simple type to a method, it is passed by value.

Thus, what occurs to the parameter that receives the argument which has no effect outside the method.

For example, consider the following program:

```
// Simple types are passed by value.  
class Test {  
    void meth(int i, int j) {  
        i *= 2;  
        j /=2;  
    }  
}  
  
class CallByValue {  
    public static void main(String args[]) {  
        Test ob = new Test();  
        int a = 15, b = 20;  
        System.out.println("a and b before call: " + a + " " + b);  
        ob.meth(a, b);  
        System.out.println("a and b after call: " + a + " "+ b);  
    }  
}
```

The **output** from this program is shown here:

a and b before call: 15 20

a and b after call: 15 20

As you can see, the operations that occur inside **meth()** have no effect on the values of **a** and **b** used in the call; their values here did not change to 30 and 10.

When you pass an object to a method, the situation changes dramatically, because objects are passed by reference. Keep in mind that when you create a variable of a class type, you are only creating a reference to an object. Thus, when you pass this reference to a method, the parameter that receives it will refer to the same object as that referred to by the argument. This effectively means that objects are passed to methods by use of call-by-reference. Changes to the object inside the method do affect the object used as an argument. For example, consider the following program:

```
// Objects are passed by reference.
```

```
class thatTest {  
    int a, b;  
    Test(int i, int j) {  
        a = i;  
        b = j;  
    }  
}
```

```
// pass an object
void meth(Test o) {
    o.a *= 2;
    o.b /= 2;
}
}

class CallByRef {
    public static void main(String args[]) {
        Test ob = new Test(15, 20);
        System.out.println("ob.a and ob.b before call: " + ob.a + " " + ob.b);
        ob.meth(ob);
        System.out.println("ob.a and ob.b after call: " + ob.a + " " + ob.b);
    }
}
```

This program generates the following **output**:

ob.a and ob.b before call : 15 20

ob.a and ob.b after call : 30 10

As you can see, in this case, the actions inside **meth()** have affected the object used as an argument.

As a point of interest, when an object reference is passed to a method, the reference itself is passed by use of call-by-value. However, since the value being passed refers to an object, the copy of that value will still refer to the same object and its corresponding argument does.

When a simple type is passed to a method, it is done by use of call-by-value. Objects are passed by use of call-by-reference.

Q24. What is the purpose of text field? List and explain its constructors and important methods.

Text field allow the user to enter strings and to edit the text. The **TextField** class implements a single-line text-entry area, usually called an edit control. Text fields allow the user to enter strings and to edit the text using the arrow keys, cut and paste keys, and mouse selections. **TextField** is a subclass of **TextComponent**. **TextField** defines the following constructors:

- TextField()
- TextField(in numChars)
- TextField(String str)
- TextField(String str, int numChars)

The first version creates a default text field. The second form creates a text field that is numChars characters wide. The third form initializes the text field with the string contained in str. The fourth form initializes a text field and sets its width.

TextField (and its superclass **TextComponent**) provides several methods that allow you to utilize a text field. To obtain the string currently contained in the text field, call **getTex()**. To set the text, call **setText()**. These methods are as follows:

```
String getText()  
void setText(String str)
```

here, str is the new string.

The user can select a portion of the text in a text filed. Also, you can select a portion of text under program control by using **select()**. Your program can obtain the currently selected text by calling **getSelectedText()**. These methods are shown here:

```
String getSelectedText()  
void select(int startIndex, int endIndex)
```

getSelectedText() returns the selected text. The **select()** method selects the characters beginning at startIndex and ending at endIndex-1.

You can control whether the contents of a text field may be modified by the user by calling **setEditable()**. You can determine editability by calling **isEditable()**. These methods are shown here:

```
boolean isEditable()  
void setEditable(boolean canEdit)
```

isEditable() returns **true** if the text may be changed and **false** if not. In **setEditable()**, if canEdit is **true**, the text may be changed. If it is **false**, the text cannot be altered.

There may be times when you will want the user to enter text that is not displayed, such as a password. You can disable the echoing of the characters as they are typed by calling **setEchoChar()**. This method specifies a single character that the **TextField** will display when characters are entered thus, the actual character that the **TextField** will display when characters are entered thus, the actual characters typed will not be shown. You can check a text field to see if it is in this mode with the **echoCharSet()** method. You can retrieve the echo character by calling the **getEchoChar()** method. These methods are as follows:

```
void setEchoChar(char ch)  
boolean echoCharSet()  
char getEchoChar()
```

Here, ch specifies the character to have echoed.

Q25. Answer the following questions :

(1) How do you create a button labelled “YES”? How do you change a label in a button?

We can create a button labelled “yes” by writing the following code :

Button b1 = new Button('Yes');

And, we can change a label in a button by using setLabel() method.

B1.setLabel("No");

Syntax :

void setLabel(String str) method for a Button object.

(2) Can you specify the size of lists?

Yes, the constructor is List(int numRows) where numRows denotes the number of entries in the list that will always be visible.

(3) Can you remove a menu item from the menu?

Yes, Menu items can be removed if they have been created by the constructor Menu(String name, Boolean removable). If removable is true, the menu item can be "torn" from the main menu and place wherever the user desires on the screen.

(4) What is the return value of a main () method?

"main()" returns nothing, hence return type is "void". It is the method automatically invoked by Java Run Time Environment.

(5) How do you denote a comment line?

Comments are of 3 types:

- i) "://" used to comment a single line.
- ii) "/* */" used to comment out a block of code.
- iii) "/** **/" used for generation of automatic documentation with Javadoc utility.

Q26. What is the need of layout manager? Describe a Border Layout. How do you create a Border Layout Container? List the names of five sections in a Border Layout. Can you add multiple components in the same section?

The need of Layout manager is to automatically arrange the controls within a window. Imagine a window in which we want to put 6 buttons as the size of the window is fixed all the buttons can't be fixed in a single row then layout manager allows to position the controls at an appropriate locations within the window. Compared to the class FlowLayout, which implements the interface java.awt.LayoutManager and sizes each component to be its preferred size, the interface java.awt.LayoutManager allows a constraint to be associated with each component. This constraint is an object that the layout manager uses when determining the size and position of components in a container and is typed as java.lang.Object although each implementer has a specific constraint

that it is designed to work with. The class `java.awt.BorderLayout` implements `LayoutManager2` and its constraint is a string that can be one of the values “North”, “East”, “Center”, “South” and “West”. Two additional constants are used to support relative positioning based on the container’s `ComponentOrientation`. For example, in a container where `ComponentOrientation` is `ComponentOrientation`.

`LEFT_TO_RIGHT`, “Before line begins” maps to “West” and “After line ends” maps to “East”, respectively. Also, mixing the two types of constants is not recommended. When adding components, you will use these constants with the following form of `add()`, which is defined by `Container` :

```
void add(Component compObj, Object region);
```

Here, `compObj` is the component to be added, and `region` specifies where the component will be added.

Yes, we can add more than one component in the same section.

Q27. State true or false :

Every element in the array has the same type.

TRUE

Array size is fixed after it is created.

TRUE

You can add a frame to a panel.

FALSE

You can add panel to a frame.

TRUE

You can add a button to a frame.

TRUE

Q28. How do interfaces and abstract classes differ in Java ? Give an example of each type.

INTERFACES

(1) Using interface you can specify, what a class must do, but not how it does. Interface is a standalone structure.

ABSTRACT CLASSES

(1) Abstract classes also specify what must be done but not how it does but it is used with inheritance i.e. Abstract class is super class of the subclass in which abstract methods will be overridden. It is not a standalone structure.

(2) Method overridden in the class that implements interface must be proceeded by keyword public. (2) Method overridden does n't require public to be proceeded.

(3) They are designed to support dynamic method resolution at runtime without complexon of extensibility (3) They are also for dynamic method resolution but complexon of extensibility is there.

e.g. Interface

```
interface Callback {
    void callback (int param);
}
class client implements Callback{
    public void callback (int p)
    {
        System.out.println ("Callback called with " + p);
    }
}
```

Abstract class

```
abstract class A {
    abstract void callme ();
}

class B extends A {
    void call me ()
    {
        System.out.println ("B's implementation of callme");
    }
}

class AbstractDemo {
    public static void main (String args [ ])
    {
        B b = new B();
        b.callme ();
    }
}
```

Q29. “A java file can have only one public class.” Explain.

Javafile can have onlyone public class i.e. there can only be one origin of entrance in the program by means of Java interpreter which is an absolutely

logical implementation.

Java interpreter searches for main() function in the class which has same name as that of program. The class containing main() function is the public class. You cannot have two main() functions in the program in two different classes as then both class names will be same as program name which is impossible. Moreover, there can't be two entrances to the program for purpose of executing it, that will be illogical.

e.g.. employee.Java

```
class figure {
    void getdata ( )
    {}
}
class employee { ; class name same as program
                name, only one class can have this
                name i.e. "employee"
    public static void main (String args [ ])
    {
        figure f = new figure ( );
        f.getdata ();
    }
}
```

Q30. What is default access for class members in Java ? Explain with the help of suitable example.

When a member does not have an explicit access specification, it is visible to subclass as well as to other classes in the same package. This is default access. The following example shows default access control :

The Protection.java defines two **int** variables. The variable **n** is declared with the default protection, **n_pri** is **private**.

The second class, **Derived**, is a subclass of **Protection** in the same package, **p1**. This grants **Derived** access to every variable in **Protection** except for **n_pri**, the **private** one.

The third class, **SamePackage**, is not a subclass of **protection**, but is in the same package and also has access to all but **n_pri**.

This is file **Protection.java** :

```
package p1;
public class Protection {
    int n = 1; //Default Access Method in Java
    private int n_pri = 2;
```

```
public Protection () {
    System.out.println ("base constructor") ;
    System.out.println ("n = " + n);           //Default Access Method in Java
    System.out.println ("n_pri = " + n_pri) ;
}
}
```

This is file **Derived.java** :

```
package p1;
class Derived extends Protection {
    Derived () {
        System.out.println ("derived constructor") ;
        System.out.println ("n = " + n);           //Default Access Method in Java

        // class only
        // System.out.println ( "n_pri = " + n_pri) ;
    }
}
```

This is file **SamePackage.java** :

```
package p1 ;
class Samepackage {
    SamePackage () {
        Protection p = new Protection () ;
        System.out.println ( "Same Package Constructor") ;
        System.out.println ( "n = " + p.n) ;           //Default Access Method in Java

        // class only
        // System.out.println ( "n_pri = " + p.n_pri ) ;
    }
}
```

Q31. What is inheritance ? Explain how you can inherit a class into another class in Java with the help of a program.

The Java classes can be reused in several ways. Once a class has been written and tested, it can be adapted by other programmers to suit their requirements. This is basically done by creating new classes, reusing the properties of the existing ones. The mechanism of deriving a new class from an old one is called **Inheritance (or Derivation)**. The old class is referred to as the *base class* and the new one is called the *derived class* or *subclass*. Therefore, a subclass is a specialized version of a super class or base class and adds its

own, unique elements.

To inherit a class, Java uses **extends** keyword. To inherit a class, syntax for the derived class declaration is

```
class<derived class name> extends <base class>
```

```
{  
    //body of the class  
}
```

Java doesn't support the multiple inheritance, so a derived class can only inherit one class, but Java fully support multilevel inheritance e.g., given three classes called A, B, and C, C can be a subclass of B, which is a subclass of A. Here C inherits all the traits of B, and A classes.

The example shows a short program, here class B inherits class A and all members of class A included in class B. That is, object of the class B can access any member of class A as it access its own member.

```
class A  
{  
    int i;  
    void get_A_Val()  
    {  
        System.out.println("Value of i is : " + i);  
    }  
}  
class B extends A  
{  
    int j;  
    void get_B_Val()  
    {  
        get_A_Val(); // get_A_val() is now member of B  
        System.out.println("Value of j is : " + j);  
    }  
}  
class mainClass  
{  
    public static void main(String args[])  
    {  
        B objB; //Object of B declared  
        objB=new B(); //Object is instantiated  
  
        objB.j=10;  
  
        objB.i=5; //objB accessing i as its own member
```

```
    objB.get_B_Val();  
}  
}
```

Even though A is a superclass for B, it is also a completely independent, stand-alone class. Being a superclass for a subclass does not mean that the superclass cannot be used by itself. Further, a subclass can be a superclass for another subclass.

.....
SOLVED PAPERS
MCS-24
.....

**MCS-24 : OBJECT ORIENTED TECHNOLOGIES AND
JAVA PROGRAMMING**
DEC- 2005

Note : Question **one** is **compulsory**. Answer any **three** from the rest.

Q1(a). What is an object oriented paradigm? Explain two differences between the object oriented paradigm of programming languages and the structured paradigm of programming languages. 5

Refer to Chapter-1, Q-1, Page no.-1

(b) What is message passing? Explain the need of message passing in object oriented programming with an example. 5

Refer to Chapter-1, Q-4, Page no.-6

(c) What is a constructor? Write a Java program to explain the need of a constructor in problem solving. 5

It can be tedious to initialize all of the variables in a class each time an instance is created. Even when you add convenience function like `getdata()`, it would be simplest and more concise to have all of the setup done at the time the object is first created. Because the requirement for initialization is so common, Java allows objects to initialize themselves when they are created. This automatic initialization is performed through the use of a constructor.

A constructor initializes an object immediately upon creation. It has the same name as the class, in which it resides and is syntactically similar to a method. Once defined, the constructor is automatically called immediately after the object is created, before the `new` operator completes. Constructors look a little strange because they have not return type, not even `void`. This is because the implicit return type of a class's constructor is the class type itself. It is the constructor's job to initialize the internal state of an object so that the code creating an instance will have a fully initialized, usable object immediately.

Example : add a constructor in Item template so the values of variable automatically initialized when an object is constructed.

Item()

{

```
Number = 10;  
Cost = 120;  
}
```

The constructor defined above in class Item. Set the instance variable number and cost for each object to the same values. However, there are cases when we want to start an object with different set of values. For this we need to add parameters to the constructor Item

```
Item (int num, cost cost_item)  
{  
    number = num;  
    cost = cost_item;  
}
```

(d) Write a program in Java which reads two real numbers, finds the sum of these two numbers and prints the real and imaginary part of this sum separately.

5

```
import java.io.*;  
  
class RealNumber{  
    double real,img;  
    void input(){  
        BufferedReader br=new BufferedReader(new  
InputStreamReader(System.in));  
  
        try{  
            System.out.print("Enter Real Part=");  
            String st=br.readLine();  
            real=Double.parseDouble(st);  
  
            System.out.print("Enter Imaginary Part=");  
            st=br.readLine();  
            img=Double.parseDouble(st);  
        }catch( IOException e){}  
    }  
    static RealNumber add(RealNumber a,RealNumber b){
```

```

RealNumber c=new RealNumber();
c.real=a.real+b.real;
c.img=a.img+b.img;
return c;

}

void print(){
    System.out.println("Real="+real+"\tImaginary="+img);
}

}

class ReadNumberAdd{
    public static void main(String arg[]){
        System.out.println("Enter First Real Number");
        RealNumber a=new RealNumber();
        a.input();
        System.out.println("Enter Second Real Number");
        RealNumber b=new RealNumber();
        b.input();

        RealNumber c=RealNumber.add(a,b);

        c.print();
    }
}

```

(e) What is ‘super’ in Java ? Explain at least two different uses of ‘super’ in the Java programs, with an example.

5

Refer to Chapter-6, Q-4, Page no.-82

(f) Write a program in Java to find whether the size of a given file is less than 50 bytes or not; and if it is less add characters to make it 50 bytes.

```

import java.io.*;
class read
{
public boolean enabled(String name)
{

```

```
File f=new File(name +".txt");

if(f.canWrite())
return true;
else
return false;
}

public void read(String name)
{
try{
char c='o';
FileReader fr=new FileReader(name + ".txt");
while((byte)c!=-1)
{c=(char)fr.read();System.out.print(c);}
System.out.println("\n");
}catch(IOException i){}
}

public long length(String name)
{
File f=new File(name +".txt");
System.out.print(f.length());
System.out.println();
return (50-f.length());
}
public static void main(String args[]) throws IOException
{
read rd=new read();
c           h           a           r
H{A,P,P,E,N,D,E,D,B,Y,D,E,V,E,L,O,P,E,R,S,A,N,D,E,E,P,B,A,N,S,A,L};  
FileWriter f1=new FileWriter("Hello.txt",true);

System.out.println("Enter the File Name");
BufferedReader cr=new BufferedReader(new InputStreamReader(System.in));
String name=cr.readLine();
long m=rd.length(name);
System.out.println();
System.out.print("Before Appended");
System.out.println();
rd.read(name);
```

```

if(m!=0)
{
System.out.println("Too Small ...I'll do it");
for(int i=0;i<m;i++)f1.write(hi[i]);
f1.close();
}
System.out.print("After Appended");
System.out.println();
rd.read(name);
}}

```

(g) Explain two situations when String Buffer would be used for string handling. Also write a program which appends the string “programming”, to the string “Java”. Print the final content of the appended string.

5

Below are the two situations when StringBuffer class is used for string handling :

- a) At the time of insertion of a string in the middle or appended to the end.
- b) To make the utility like find & replace a sub-string within a string. Using StringBuffer the coding required will be less as compare to String class.

```

class StringBufferDemo {
    public static void main(String arg[]){
        StringBuffer sb=new StringBuffer("Java");
        System.out.println("Initial String="+sb);
        System.out.println("Initial Capacity="+sb.capacity());

        System.out.println("appended programming.....");
        sb.append("programming");
        System.out.println("Final String="+sb);
        System.out.println("Final Capacity="+sb.capacity());
    }
}

```

(h) What is an event? Explain different components of an event.

5

An event is an object that describes a state change in a source. It can be generated as a consequence of a person interacting with the elements in a graphical user interface. Some of the activities that cause events to be generated are pressing a button, entering a character via the keyboard, selecting an item in a list, and clicking the mouse. Several other user operations could also be cited as examples. Events may also occur that are not directly caused by

interactions with a user interface. For example, an event may be generated when a timer expires, a counter exceeds a value, a software or hardware failure occurs, or an operation is completed. You can freely define events which are useful for your application.

There are four different components of event:

- (i) Event classes, (ii) Event Listeners, (iii) Explicit Event Handling, and (iv) Adapters.

Event Classes : The EventObject class is at the top of the event class hierarchy. It belongs to the Java.util package. While most of the other event classes are present in Java.awt.event package.

The getSource() method of the EventObject class returns the object that initiated the event.

The getId() method returns the nature of the event. For example, if a mouse event occurs, you can find out whether the event was a click, a press, a move or release from the event object.

AWT provides two conceptual types of events: Semantic and Low-level events. Semantic event: These are defined at a higher-level to encapsulate the semantics of user interface component's model.

Low-level events : These events are those that represent a low-level input or windows-system occurrence on a visual component on the screen.

Event listeners : A listener is an object that is notified when an event occurs. It has two major requirements. First, it must have been registered with one or more sources to receive notifications about specific types of events. Second, it must implement methods to receive and process these notifications. The methods that receive and process events are defined in a set of interfaces found in java.awt.event. For example, the MouseMotionListener interface defines two methods to receive notifications when the mouse is dragged or moved. Any object may receive and process one or both of these events if it provides an implementation of this interface.

Explicit Event Handling and Adapters : It is a special feature provided by Java, called an adapter class, that can simplify the creation of event handlers in certain situations. An adapter class provides an empty implementation of all methods in an event listener interface. Adapter classes are useful when you want to receive and process only some of the events that are handled by a particular event listener interface. One can define a new class to act as an event listener by extending one of the adapter classes and implementing only those events in which you are interested. For example, the MouseMotionAdapter

class has two methods, mouseDragged() and mouseMoved(). The signatures of these empty methods are exactly as defined in the MouseMotionListener interface. If anybody need only mouse drag events, then MouseMotionAdapter can be simply extended. After that implement mouseDragged(). The empty implementation of mouseMoved() would handle the mouse motion events.

Q2(a). What is a Servlet? Explain the use of GET and POST methods.

5

As the name indicates Servlet is used to serve. These are the programs run on server which executes and produces output depending on the request. The Java Servlet API allows a software developer to add dynamic content to a Web server using the Java platform. The generated content is commonly HTML, but may be other data such as XML. Servlets are the Java counterpart to dynamic web content technologies such as CGI, PHP or ASP. Servlets can maintain state across many server transactions by using HTTP cookies, session variables or URL rewriting.

The Servlet API, contained in the Java package hierarchy javax.servlet, defines the expected interactions of a web container and a servlet. A web container is essentially the component of a web server that interacts with the servlets. The web container is responsible for managing the lifecycle of servlets, mapping a URL to a particular servlet and ensuring that the URL requester has the correct access rights.

A Servlet is an object that receives requests (ServletRequest) and generates a response (ServletResponse) based on the request. The API package javax.servlet.http defines HTTP subclasses of the generic servlet (HttpServlet) request (HttpServletRequest) and response (HttpServletResponse) as well as an (HttpSession) that tracks multiple requests and responses between the web server and a client. Servlets may be packaged in a WAR file as a Web application.

Use of GET and POST methods : The GET methods is a request made by browsers when the user types in a URL on the address line, follows a link from a Web page, or makes an HTML form that does not specify a METHOD. Servlets can also very easily handle POST requests, which are generated when someone create an HTML form that specifies METHOD= “POST”.

```
Import java.io.*;
Import javax.servlet.*;
Import javax.servlet.http.*;
Public class SomeServlet extends HttpServlet
```

```

{
    public void doGet(HttpServletRequest request, HttpServletResponse re-
sponse)
        throws ServletException, IOException
{
    //Use "request" to read incoming HTTP headers (e.g. cookies )
    //and HTML form data (e.g. data the user entered and submitted)

    //Use "response" to specify the HTTP response line and headers
    // (e.g. specifying the content type, setting cookies .)
    PrintWriter out = response.getWriter();
    //Use "out" to sent content to browser
}
}

```

To act as a servlet, a class should extend HttpServlet and override doGet or doPost (or both), depending on whether the data is being sent by GET or by POST. These methods take two arguments: an HttpServletRequest and an HttpServletResponse objects.)

The HttpServletRequest has methods for information about incoming information such as FORM data, HTTP request headers etc.

The httpServletResponse has methods that let you specify the HTTP response line (200, 404, etc), response headers (Content-type, Set-Cookies, etc.), and, most importantly, a PrintWriter used to send output back to the client.

(b) What is method overloading? What are the important points which should be taken care of while overloading methods? Write a Java program to explain the working of overloaded methods. 10

Refer to Chapter-5, Q-3, Page no.-67

(c) What is Border Layout? Write a Java program which creates Border Layout and adds two text boxes to it. 5

A BorderLayout organizes an applet into North, South, East, West and Center sections. North, South, East and West are the rectangular edges of the applet. They're continually resized to fit the sizes of the widgets included in them. Center is whatever is left over in the middle.

A BorderLayout places objects in the North, South, East, West and Center of an applet. New object of BorderLayout can be created by writing a statement this.setLayout(new BorderLayout());

There's no centering, left alignment, or right alignment in a BorderLayout. However, you can add horizontal and vertical gaps between the areas.

```

import java.applet.*;
import java.awt.event.*;
import java.awt.*;

/*<Applet code="BorderDemo" width=400 height=400>
</Applet>*/
public class BorderDemo extends Applet{
    TextField text1,text2;

    public void init(){
        setLayout(new BorderLayout(10,10));
        text1=new TextField("Text1",30);
        text2=new TextField("Text2",40);

        add(text1,BorderLayout.EAST);
        add(text2,BorderLayout.WEST);
    }
}

```

Q3(a). Explain the following concepts with an example of each: 6

(i) Class and Object;

Refer to Chapter-1, Q-3, Page no.-5

(ii) Inheritance.

Refer to Chapter-1, Q-9, Page no.-10

(b) Find the errors in the following Java program, and correct them. 4

Class Examination:

```

{
public void main (args[])
{
system.out.println ("Java is an OOL")
int i = 10;
for (i > 0; i - - )
system.out.println ("i="i);
}
{
}
}

```

```

class Examination
{
public static void main(String args[])
{
System.out.println("Java is an OOL");
for (int i=10;i>0;i--)
System.out.println("i=" + i);
}
}

```

(c) Write a program in Java which finds the number of lines and number of characters in a given file.

5

```

import java.io.*;
class Quest1
{
public static void main(String args[]) throws IOException
{
char c='o';
int line=0;
int count=0;
FileReader f=new FileReader("Examination.java");
while((byte)c!=1)
{
c=(char)f.read();
count++;
if(c=='\n')
line++;
System.out.print(c);
}
System.out.println("Total no. of lines are:- " +line);
System.out.println("Total no. of Characters are:- " +count);
}}

```

(d) What are the advantages of ‘platform independent languages’? Also explain how Java is platform independent.

5

Platform independent languages can run on multiple platforms like mac, sun, windows, etc.. the advantage of the languages are -

- a) Write once run anywhere
- b) Run on multiple platforms
- c) Easily distributed
- d) Develop programs more quickly
- e) Write less code

**Q4(a). What is a session? How does URL rewriting store session details?
Explain this with an example.** 8

A lasting connection between a user and a {peer}, typically a {server}, usually involving the exchange of many packets between the user's computer and the server. A session is typically implemented as a layer in a network {protocol} (e.g. telnet, FTP). In the case of protocols where there is no concept of a session layer (e.g. UDP) or where sessions at the session layer are generally very short-lived (e.g. HTTP), virtual sessions are implemented by having each exchange between the user and the remote host include some form of cookie which stores state e.g. a unique session ID, information about the user's preferences or authorisation level, etc.

As it is essential to track the user's requests to perform this task, Java Servlets offers two different ways:

(i) It is possible to save information about user state on the server using Session object.

(ii) It is possible to save information on the client system using cookies.

A session is sequence of HTTP requests, from the same client, over a period of time.

URL Rewriting. You can append session details on the end of each URL that identifies the session, and the server can associate that session identifier with data it has stored about that session. This is also an excellent solution, and even has the advantage that it works with browsers that don't support cookies or where the user has disabled cookies. However, it has most of the same problems as cookies, namely that the server-side program has a lot of straightforward but tedious processing to do. In addition, you have to be very careful that every URL returned to the user (even via indirect means like Location fields in server redirects) has the extra information appended. And, if the user leaves the session and comes back via a bookmark or link, the session information can be lost.

Example : look at request information for search servlet:

i) <http://www.IgnouOnline.com/servlet/search>

ii) <http://www.IgnouOnline.com/servlet/search/23422abc>

iii) <http://www.IgnouOnline.com/servlet/search?sessionid=23422abc>

For the original servlet[i] the URL is clean. In [ii] we have URL re-written at the server to add extra path information as embedded links in the pages we send back to the client. When the client clicks on one of these links, the search servlet will do the following:

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
{
```

```

...
String sessionid = request.getPathInfo(); //return 2343abc from[ii]
...
}

```

Extra path information work for both GET and POST methods involved from inside as well as outside of forms with static links.

Technique [iii] simply re-writes the URL with parameter information that can be accessed as follows:

```
request.getParameterValue("sessionid");
```

URL re-writing provides a means to implement anonymous session tracking. However, with URL rewriting you are not limited to forms and you can re-write URLs in static documents to contain the required session information. But URL re-writing suffers from the same major disadvantage that hidden fields do, in that they must be dynamically generated and the chain of HTML page generation cannot be broken.

(b) What is an exception? Explain with an example, how exceptions are handled in Java.

5

An exception is an abnormal condition that arises in a sequence at run time. In other words, an exception is a run-time error. In computer languages that do not support exception handling, errors must be checked and handled manually—typically through the use of error codes, and so on. This approach is as cumbersome as it is troublesome. Java's exception handling avoids these problems and, in the process, brings run-time error management into the object-oriented world.

To guard against and handle a run-time error, simply enclose the code that you want to monitor inside a try block. Immediately following the try block, include a catch clause that specifies the exception type that you wish to catch. To illustrate how easily this can be done, the following program includes a try block and a catch clause which processes the Arithmetic Exception generated by the division-by-zero error:

```

class Exc2 {
    public static void main (String args[ ]) {
        int d, a;
        try {                                     // monitor a block of code.
            d = 0;
            a = 42 / d ;
            System.out.println ("This will not be printed. ");
        } catch (ArithmaticException e) {           //catch divide-by-zero error
            System.out.println ("Division by Zero.");
        }
    }

```

```

System.out.println ("After catch statement.");
}
}
}

```

This program generates the following output:

Division by zero.

After catch statement.

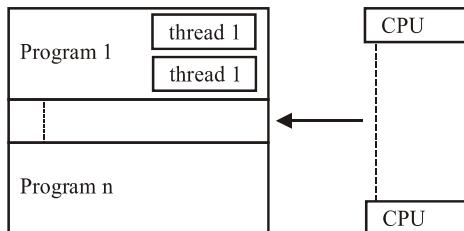
Notice that the call to **println()** inside the **try** block is never executed. Once an exception is thrown, program control transfers out of the **try** block into the **catch** block. Put differently, **catch** is not “called,” so execution never “returns” the try block from a **catch**. Thus, the line “This will not be printed.” is not displayed. Once the **catch** statement has executed, program control continues with the next line in the program following the entire **try/catch** mechanism.

(c) What is multithreading? Explain two advantages of multithreaded programs. Write a program in Java to explain how different priorities can be assigned to different threads.

7

Multithreading is the ability of a program or an operating system process to manage its use by more than one user at a time and to even manage multiple requests by the same user without having to have multiple copies of the programming running in the computer. Or we can say, the ability of an operating system to execute different parts of a program, called threads, simultaneously. Each user request for a program or system service (and here a user can also be another program) is kept track of as a thread with a separate identity. As programs work on behalf of the initial request for that thread and are interrupted by other requests, the status of work on behalf of that thread is kept track of until the work is completed. The programmer must carefully design the program in such a way that all the threads can run at the same time without interfering with each other.

In figure below concept of multithreading is shown, there are many programs running in a single program many threads are running this is the concept of multithreading.



Advantages of Multithreaded Programs :

- i) Concurrency can be used within a process to implement multiple instances of simultaneous services.
- ii) Multithreaded programs require less processing overhead because concurrent threads are able to share common resources more efficiently. A multithreading web server is one of the examples of multithreaded programs running on that. Multithreaded web servers are able to efficiently handle multiple browser requests by running a thread for a request received and utilizes the common resources.
- iii) Multithreaded programs are very efficient that make maximum use of the CPU. Unlike most other programming languages, Java provides built-in support for multithreaded programming..

```
class thread1 implements Runnable
{
    Thread t;
    thread1()
    {
        t=new Thread(this,"thread1");
        t.setPriority(10);
        System.out.println("Child thread :" +t);
        t.start();
    }
    public void run()
    {
        try{
            for(int i=5;i>0;i--)
            {
                System.out.println("Child thread :" +i);
                t.sleep(1000);
            } }catch(InterruptedException e){}
        System.out.println("Exiting child thread.....");
    }
    public static void main(String args[])
    {
        thread1 i=new thread1();
        Thread t1=Thread.currentThread();
        try{
            System.out.println("Main thread "+t1);
            for(int j=5;j>0;j--)
            {

```

```

System.out.println("main thread :" +j);
t1.sleep(500);
}
}catch(InterruptedException e){}
System.out.println("main thread Exiting .....");
}}
```

Q5(a). What is a TCP/IP socket? Explain the use of a TCP/IP socket through an example of a program written in Java. 7

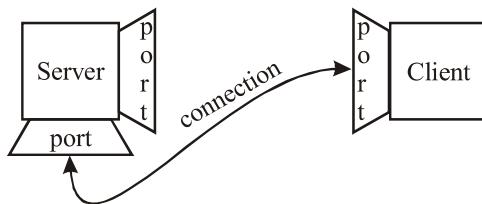
TCP/IP Socket is a java socket that uses TCP/IP(Transmission Control Protocol/Internet Protocol, connection-based protocol) to reliable flow of data between two computers. The Socket provides a point-to-point channel for applications that require reliable communications. A socket is one endpoint of a two-way communication link between two programs running on the network.

A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent. Socket classes are used to represent the connection between a client program and a server program.

The `java.net` package provides two classes to implement TCP/IP socket programming for the client connection and server connection—

1. `Socket`
2. `ServerSocket`

Upon acceptance, the server gets a new socket bound to a different port. It needs a new socket (and consequently a different port number) so that it can continue to listen to the original socket for connection requests while tending to the needs of the connected client.



Client Application

```

import java.io.*;
import java.net.*;

class Chat{
public static void main(String arg[]) throws IOException{
```

```
InetAddress client=InetAddress.getLocalHost();

Socket s=new Socket(client,5001);

DataOutputStream remoteout=new DataOutputStream(s.getOutputStream());

String str="Hello, how r u?";
byte b[]={str.getBytes()};
remoteout.write(b);

}

}
```

Server Application

```
import java.io.*;
import java.net.*;

class SChat{
public static void main(String arg[]) throws IOException {
    ServerSocket ss=new ServerSocket(5001);

    Socket sock=ss.accept();

    DataInputStream remoteIn=new DataInputStream(sock.getInputStream());
    byte b[]=new byte[50];
    remoteIn.read(b);
    System.out.println(new String(b,0,b.length));
}
}
```

(b) What is a package ? Explain the different access controls for packages in Java. 5

Refer to Chapter-8, Q-1, Page No.-105

(c) What is Unicode? Explain the advantage of using Unicode. 2

Unicode is an entirely new idea in setting up binary codes for text or script characters. Officially called the Unicode Worldwide Character Standard, it is a system for “the interchange, processing, and display of the written texts of

the diverse languages of the modern world.” It also supports many classical and historical texts in a number of languages.

Currently, the Unicode standard contains 34,168 distinct coded characters derived from 24 supported language scripts. These characters cover the principal written languages of the world.

Additional work is underway to add the few modern languages not yet included.

Using unicode has many **advantages** over using a single-byte locale. A minor one is the ability to use any character in file names and on the command line. The main advantage of Unicode, however, is that it allows easier data exchange and better interoperability than any other character set. Unicode is meant to replace ASCII in the future, so at some point “text file” is going to mean “UTF-8 file” just as it means “ASCII file” now. UTF-8 is Unicode file format.

(d) Write a program in Java which finds the sum of the two arrays given below,

A = 1 2 3 4

2 3 4

5 6

B = 4 3 2 1

3 2 1

2 1

Make necessary assumptions, if any.

6

```
class ArrayAddition{
    public static void print(int a[][]){
        for(int i=0; i<a.length;i++){
            for(int j=0;j<a[i].length;j++){
                System.out.print(a[i][j]+"\t");
            }
            System.out.println();
        }
    }
    public static void main(String arg[]){
        int a[][]={{1,2,3,4},{2,3,4},{5,6}};
        int b[][]={{4,3,2,1},{3,2,1},{2,1}};

        int c[][]=new int[3][4];

        for(int i=0;i<3;i++){
            for(int j=0;j<a[i].length;j++){
                www.IgnouOnline.com
            }
        }
    }
}
```

```
        c[i][j]=a[i][j]+b[i][j];  
    }  
}  
  
print(a);  
print(b);  
print(c);  
}  
}
```

**MCS-24 : OBJECT ORIENTED TECHNOLOGIES AND
JAVA PROGRAMMING**
JUNE-2006

Note : Question **one** is **compulsory**. Answer any **three** from the rest.

Q1(a). Explain the concept of Polymorphism. Also, give an example of a Polymorphism. 5

Refer to Chapter-5, Q-5, Page no.-70

(b) What is an object ? How are objects and classes associated with each other ? Also give two advantages of messages passing between objects.

5

Refer to Chapter-1, Q-3, Page no.-5

(c) When can an object be used as a reference of another object ? What care should be taken in such kind of referencing ? Your explanation should be supported by an example. 5

One object can be used as reference to another object provided both are of the same class **type**. Objects should be used as reference very carefully because if there is any change in values of instance variables of one object, values of instance variable of the second object also gets changed.

In java when objects are created using new operator. When objects are passed to a function, by default they are passed by reference. Like a student object is created and passed to multiple classes to set its different attributes.

```
    LibraryCard IssueLibraryCard(Student s){
```

```
    }
```

```
    ...
```

When we create a reference object we must care about the memory leakages.
E.g.

```
    Student s1=new Student();
    Student s2=s1;
```

Now both variable s1 and s2 point to the same object. The object previously referenced by s1 is no longer useful. This object is known as garbage. Garbage is automatically collected by JVM. If you know that an object is no longer needed, you can explicitly assign null to a reference variable for the object. The Java VM will automatically collect the space if the object is not referenced by any variable.

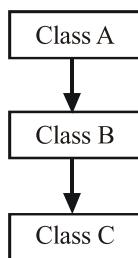
(d) What is multilevel inheritance ? How does it differ from multiple inheritance ? Also give an example of multiple inheritance. 5

Inheritance is a mechanism through which a class is shared by other class or classes. The class, which is shared, by other class is called BASE CLASS or SUPER CLASS and the class which share base class is called DERIVED CLASS or CHILD CLASS.

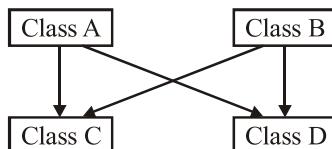
Types of Inheritance

1. Multi-level Inheritance
2. Multiple Inheritance

Multi-level Inheritance



Multiple Inheritance



Java doesn't support the multiple inheritance, so a derived class can only inherit one class, but Java fully support multilevel inheritance e.g., given three classes called A, B, and C, C can be a subclass of B, which is a subclass of A. Here C inherits all the traits of B, and A classes.

The example shows a short program, here class B inherits class A and all members of class A included in class B. That is, object of the class B can access any member of class A as it access its own member.

```

class A
{
    int i;
    void get_A_Val()
  
```

```

{
    System.out.println("Value of i is : " + i);
}
}

class B extends A
{
    int j;
    void get_B_Val()
    {
        get_A_Val(); // get_A_val() is now member of B
        System.out.println("Value of j is : " + j);
    }
}

class mainClass
{
    public static void main(String args[])
    {
        B objB;           //Object of B declared
        objB=new B(); //Object is instantiated

        objB.j=10;

        objB.i=5;      //objB accessing i as its own member

        objB.get_B_Val();
    }
}

```

Even though A is a superclass for B, it is also a completely independent, stand-alone class. Being a superclass for a subclass does not mean that the superclass cannot be used by itself. Further, a subclass can be a superclass for another subclass.

(e) What are the five issues that are to be taken care of while overriding a method ?

5

Refer to Chapter-6, Q-5, Page no.-84

(f) Explain the concept of Multithreading with the help of a diagram. Also, give atleast two advantages of Multithreading.

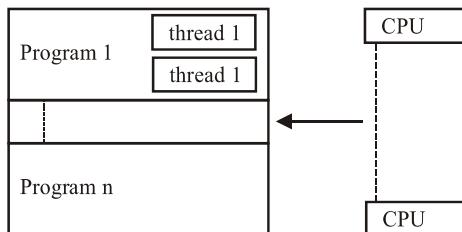
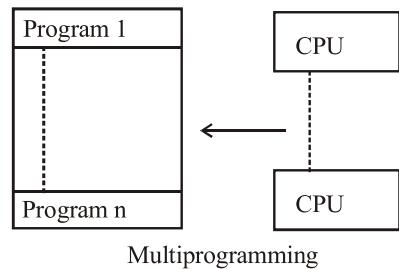
5

#: What are the advantages of Multithreading? Also explain by diagram the difference between multiprogramming and multithreading.

Difference between multiprogramming and multithreading : Multipro-

gramming is the feature that allows you to run two or more programs concurrently whereas Multithreading is dividing a single program into a no of threads(light weight processes) and running those threads simultaneously. OR Multithreading provides concurrency within the content of a single process. But multiprogramming provides concurrency within the content of a single process. But multiprogramming also provides concurrency between processes. Threads are not complete processes in themselves. They are a separate flow of control that occurs within a process.

In figure below shows the difference between the multiprogramming and multithreading.



Advantages of Multithreading

- Concurrency can be used within a process to implement multiple instances of simultaneous services.
- Multithreading requires less processing overhead than multiprogramming because concurrent threads are able to share common resources more efficiently. A multithreading web server is one of the example of multithreaded programming. Multithreaded web servers are able to efficiently handle multiple browser requests. They handle one request per processing thread.
- Multithreading enables programmers to write very efficient programs that make maximum use of the CPU. Unlike most other programming languages, Java provides built-in support for multithreaded programming. The Java runtime system depends on threads for many things. Java uses threads to enable the entire environment to be synchronous.

(g) Give the reasons why the main () method in Java is defined as public and static. Also explain, why the name of Java class containing the main () method has the same name as the name of the file in which it is stored. 5

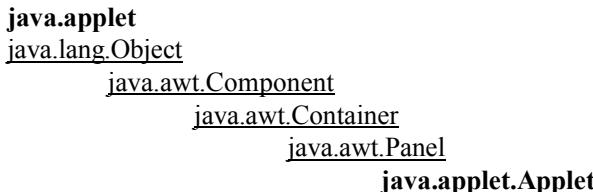
public keyword is an access specifier, which allows the programmer to control the visibility of class members. When a class member is preceded by public, then that member may be accessed by code outside the class in which it is declared. In case of main(), it must be declared as public, since it must be called by code outside of its class when the program is started.

static keyword allow main() to be called without having to instantiate a particular instance of the class. This is important as main() is called by the Java Interpreter before any objects are made.

In java a .java file may contain more than one class, when a code is compiled by java compiler it create .class files depending on the classes in java file. The convention that filenames correspond to that class name, which contains main() function, makes it easier to maintain and organize the programs.

(h) Where does the Applet class appear in the hierarchy of Java classes ? Show this through a class hierarchy diagram. 5

Refer to Chapter-9, Q-1, Page no.-124



Q2(a). Explain the eight basic features of Java. 4

Refer to Chapter-2, Q-5, Page no.-19

(b) What is a Package ? Explain the process of defining our own package, with the help of an example. 5

Refer to Chapter-8, Q-1, Page no.-105

(c) Write the syntax, and explain, any four constructors of the String class. 4

Refer to Chapter-3, Q-4, Page no.-29

(d) Explain the order of constructor calling in multilevel inheritance, with the help of an example. 5

Below is the Order of Constructor Calling in Multilevel Inheritance :

When the object of a subclass is created the constructor of the subclass is called which in turn calls constructor of its immediate superclass. For example, if we taken a case of multilevel inheritance, where class B inherits from class A. and class C inherits from class B. You can see the output of the example program given below, which shows the order of constructor calling.

```
//Program
class A
{
A()
{
System.out.println("Constructor of Class A has been called");
}
}
class B extends A
{
B()
{
super();
System.out.println("Constructor of Class B has been called");
}
}
class C extends B
{
C()
{
super();
System.out.println("Constructor of Class C has been called");
}
}
Class Constructor_Call
{
public static void main(String[] args)
{
System.out.println("-----Welcome to Example of Constructor call-----");
C objc = new C();
}
}
```

Output:

-----Welcome to Constructor call Demo -----
 Constructor of Class A has been called
 Constructor of Class B has been called
 Constructor of Class C has been called

(e) What is this pointer ? Explain its use with the help of an example.

2

Refer to Chapter-6, Q-3, Page no.-81

Q3(a). What is the method to retrieve the colour of the text ? Write a program to retrieve RGB values in a given colour.

7

To retrieve the Color of a text or a String method public int getColor() can be used. It returns a value between 0 and 255 representing the color content.

The Program given bellow is written for getting RGB components of the color

import java.awt.*;

```
import java.applet.Applet;
public class ColorText extends Applet
{
  public void paint(Graphics g)
  {
    int color;
    g.drawString("Our Pink color String",40,50);
    color = getColor(g);
    g.drawString("The Color of the String is "+color, 40,35);
  }
}
```

(b) What is a TCP socket ? Explain how a TCP socket is different from a UDP socket.

5

A **TCP socket** is one end-point of a two-way communication link between two programs running on the network. Socket classes are used to represent the connection between a client program and a server program. The java.net package provides two classes **Socket** and **ServerSocket** that implement the client side of the connection and the server side of the connection, respectively.

First, a server is set up to listen at a given port. The server waits and does nothing until a client attempts to connect that port. If everything goes fine, the connection is successful and both the server and client have an instance of the **Socket** class. From each instance of this class, an input stream and an output stream can be obtained, and all communication is done via these streams.

TCP and UDP socket : Some applications to communicate over the network, will not require the reliable, point-to-point channel provided by TCP. Rather, applications might benefit from a mode of communication that delivers independent packages of information whose arrival and order of arrival are not guaranteed. The UDP protocol provides a mode of network communication whereby applications send packets of data, called datagrams, to one another.

A datagram is an independent, self-contained message sent over the network whose arrival, arrival time, and content are not guaranteed. The DatagramPacket and DatagramSocket classes in the java.net package implement system-independent datagram communication using UDP.

The server side listens to its DatagramSocket and sends a quotation to as many clients as are listening. The client side is a simple program that simply makes a request of the server. Clients and servers that communicate via a reliable channel, such as a URL or a socket, have a dedicated point-to-point channel between themselves, or at least the illusion of one. To communicate, they establish a connection, transmit the data, and then close the connection. All data sent over the channel is received in the same order in which it was sent. This is guaranteed by the channel.

This implies, TCP socket used for a reliable connection oriented service. Data is sent without errors or duplication and is received in the same order as it is sent.

User Datagram Protocol (UDP) socket used for a connectionless service for datagrams, or messages. Datagrams are sent as independent packets. The reliability is not guaranteed, data can be lost or duplicated. However, datagram sockets have improved performance capability over stream sockets and are easier to use.

(c) What is ‘abstraction’ ? Explain two advantages of abstraction with an example. 8

Refer to Chapter-1, Q-9, Page no.-10

Q4(a). Consider the following definition :

```
1. public class exam {  
2.     public int a = 1;  
3.     public int b = 2;  
4.     public void method {final int c}  
5.     { int d = 3;  
6.         class assign {  
7.             private void iMethod (int e) {
```

```

8.          // line 8
9.          }
10.         }
11.      }
```

Write all the variables among a,b,c,d and e that can be referenced at Line 8. 4

local variable d can not be accessed from within inner class; needs to be declared final other variable a,b,c,e can be accessed at Line 8.

(b) Write a program to compute the factorial of a given number in Java. 4

```

class Number
{
    private int i = 0;
    public Number(int i)
    {
        this.i = i;
    }

    public int factorial()
    {
        int fact = 1;

        for(int a = 1; a <= i ; a++)
            fact = fact * a;

        return fact;
    }

    public static void main(String[] args)
    {
        Number myNum = new Number(3);
        System.out.println("Factorial is : " + myNum.factorial());
    }
}
```

(c) Explain the seven steps that are to be followed to establish a connection from a Java program to a database. 7

Following are the seven steps that are to be followed to establish a connection from a Java program to a database :

Step 1: Loading Drivers

First, you have to load the appropriate driver. You can use one driver from the available four. However, the JDBC-ODBC driver is the most preferred among developers. In order to load the driver, you have to give the following syntax:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

Step 2: Making a Connection

The `getConnection()` method of the Driver Manager class is called to obtain the Connection object. The syntax looks like this:

```
Connection conn = DriverManager.getConnection("jdbc:odbc:<DSN NAME>");
```

Here, note that `getConnection()` is a static method, meaning it should be accessed along with the class associated with the method. You have to give the Data Source Name as a parameter to this method.

Step 3: Creating JDBC Statements

A Statement object is what sends your SQL Query to the Database Management System. You simply create a statement object and then execute it. It takes an instance of active connection to create a statement object. We have to use our Connection object "`conn`" here to create the Statement object "`stmt`". The code looks like this:

```
Statement stmt = conn.createStatement();
```

Step 4: Creating JDBC Prepared Statement

`PreparedStatement` object is more convenient and efficient for sending **SQL** statements to the **DBMS**. The main feature, which distinguishes `preparedStatement` object from objects of `Statement` class, is that it gives an **SQL** statement right when it is created. This **SQL** statement is then sent to the **DBMS** right away, where it is compiled. Thus, in effect, a `PreparedStatement` is associated as a channel with a connection and a compiled **SQL** statement. Another advantage offered by `PreparedStatement` object is that if you need to use the same or similar query with different parameters, multiple times, the statement can be compiled and optimized by the **DBMS** just once. While with a normal `Statement`, each user of the same **SQL** statement requires a compilation all over again.

`PreparedStatements` are also created with a `Connection` method. The following code shows how to create a parameterized **SQL** statement with three input parameters:

```
PreparedStatement prepareUpdatePrice  
= con.prepareStatement("UPDATE Employee SET emp_address=? WHERE  
emp_code='1007' AND emp_name=?");
```

You can see two ? Symbol in the above PreparedStatement *prepareUpdatePrice*. This means that you have to provide values for two variables emp_address and emp_name in PreparedStatement before you execute it. Calling one of the set XXX methods defined in the class PreparedStatement can provide values. Most often used methods are setInt, setFloat, setDouble, setString, etc. You can set these values before each execution of the prepared statement.

You can write something like:

```
PrapareUpdatePrice.setInt(1, 3);
PrepareUpdatePrice.setString(2, "Dinesh");
PrepareUpdatePrice.setString(3, "100, Sector-8,Vijay Nagar, Delhi");
```

Step 5: Executing the Statement

In order to execute the query, you have to obtain the Result Set object (similar to Record Set in Visual Basic) and a call to the executeQuery() method of the Statement interface. You have to pass a SQL query like select * from students as a parameter to the executeQuery() method. If your table name is different, you have to substitute that name in place of students. Actually, the RecordSet object contains both the data returned by the query and the methods for data retrieval.

The code for the above step looks like this:

```
ResultSet rs = stmt.executeQuery("select * from students");
```

If you want to select only the name field, you have to issue a SQL command like Select Name from Student. The executeUpdate() method is called whenever there is a delete or an update operation.

Step 6: Looping Through the ResultSet

The ResultSet object contains rows of data that is parsed using the next() method, such as rs.next(). We use the getXXX() method of the appropriate type to retrieve the value in each field. For example, if your first field name is ID, which accepts Number values, then the getInt() method should be used. In the same way, if the second field Name accepts integer String values, then the getString() method should be used, like the code given below:

```
System.out.println(rs.getInt("ID"));
```

Step 7: Closing the Connection and Statement Objects

After performing all the above steps, you have to close the Connection and RecordSet objects appropriately by calling the close() method. For example, in our code above, we will close the object as conn.close()and statement

object as stmt.close().

The code for the sample program is shown below for your reference:

```
import java.sql.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Datas extends JFrame implements ActionListener {

    JTextField id;
    JTextField name;
    JButton next;
    JButton addnew;
    JPanel p;
    static ResultSet res;
    static Connection conn;
    static Statement stat;

    public Datas() {
        super("Our Application");
        Container c = getContentPane();
        c.setLayout(new GridLayout(5,1));
        id = new JTextField(20);
        name = new JTextField(20);
        next = new JButton("Next");
        p = new JPanel();
        c.add(new JLabel("Customer ID",JLabel.CENTER));
        c.add(id);
        c.add(new JLabel("Customer Name",JLabel.CENTER));
        c.add(name);
        c.add(p);
        p.add(next);
        next.addActionListener(this);
        pack();
        setVisible(true);
        addWindowListener(new WIN());
    }

    public static void main(String args[]) {
        Datas d = new Datas();
    }
}
```

www.DoeaccOnline.com

```

try {
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
conn = DriverManager.getConnection("jdbc:odbc:gully");
// gully is the DSN Name

stat = conn.createStatement();
res = stat.executeQuery("Select * from Cutomers");
// Customers is the table name

res.next();

}

catch(Exception e) {
System.out.println("Error" +e);
}
d.showRecord(res);
}

public void actionPerformed(ActionEvent e) {
if(e.getSource() == next) {
try {
res.next();
}
catch(Exception ee) {}
showRecord(res);
}
}

public void showRecord(ResultSet res) {
try {
id.setText(res.getString(1));
name.setText(res.getString(2));
}
catch(Exception e) {}

}

}//end of the main

//Inner class WIN implemented
class WIN extends WindowAdapter {
public void windowClosing(WindowEvent w) {
JOptionPane jop = new JOptionPane();

```

```
jop.showMessageDialog(null,"Database","Thanks",JOptionPane.QUESTION_MESSAGE);
}
} //end of the class
```

(d) Write an Applet in Java that will display the current date.

5

```
import java.applet.*;
import java.awt.*;
import java.util.*;
```

```
/*<applet code="J10" width=400 height=400></applet>*/
```

```
class StringDate extends Date{
    int dd,mm,yy;
    StringDate(){
        dd=super.getDate();
        mm=super.getMonth()+1;
        yy=super.getYear()+1900;
    }
}
```

```
public class DateDemo extends Applet{
    StringDate d;
    public void init(){
        d=new StringDate();
    }
    public void paint(Graphics g){
        g.drawString("Current Date="+d.dd+ "/" +d.mm+ "/" +d.yy,10,10);
    }
}
```

Q5(a). What is a session ? Explain how hidden form fields are used for session tracking.

5

A lasting connection between a user and a {peer}, typically a {server}, usually involving the exchange of many packets between the user's computer and the server. A session is typically implemented as a layer in a network {protocol} (e.g. telnet, FTP). In the case of protocols where there is no concept of a

session layer (e.g. UDP) or where sessions at the session layer are generally very short-lived (e.g. HTTP), virtual sessions are implemented by having each exchange between the user and the remote host include some form of cookie which stores state e.g. a unique session ID, information about the user's preferences or authorisation level, etc.

As it is essential to track the user's requests to perform this task, Java Servlets offers two different ways:

(i) It is possible to save information about user state on the server using Session object.

(ii) It is possible to save information on the client system using cookies.

A session is sequence of HTTP requests, from the same client, over a period of time.

Hidden form fields are HTTP tags that are used to store information that is invisible to the user. In terms of session tracking, the hidden form field would be used to hold a client's unique session id that is passed from the client to the server on each HTTP request. This way the server can extract the session id from the submitted form, like it does for any of form field, and use it to identify which client has made the request and act accordingly.

For example, using servlets you could submit the following search form:

```
<form method="post" action="/servlet/search">
<input type="text" name="searchtext">
<input type="hidden" name="sessionid" value="1211xyz">
...
</form>
```

when it is submitted to the servlet registered with the name **search**, it pulls out the sessionid from the form as follows:

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
...
String theSessionid = request.getParameterValue("sessionid");
If ( isAllowToPerformSearch(theSessionid) )
{
}
...
}
```

In this approach' the search servlet gets the session id from the hidden form field and uses it to determine whether it allows performing any more searches.

Hidden form fields implement the required **anonymous** session tracking fea-

tures the client needs but not without cost. For hidden fields to work the client must send a hidden form field to the server and the server must always return that same hidden form field. This tightly coupled dependency between client requests and server responses requires sessions involving hidden form fields to be an unbreakable chain of dynamically generated web pages. If at any point during the session the client accesses a static page that is not part of the chain, the hidden form field is lost, and with it the session is also lost.

(b) Write a program in Java using the String Buffer class which reverses the string “IGNOU MCA STUDENT”. 5

Class Reversing String

```

{
public String reverseMe(String source)
{
    int i, len = source.length();
    StringBuffer dest = new StringBuffer(len);
    System.out.println("Length of dest: "+dest.length());
    System.out.println("Capacity of dest: "+dest.capacity());
    for (i = (len - 1); i >= 0; i--)
        dest.append(source.charAt(i));
    System.out.println("Capacity of dest: "+dest.capacity());
    System.out.println("Length of dest: "+dest.length());
    return dest.toString();
}
}

public class ReverseString
{
    public static void main ( String args[])
    {
        ReversingString R = new RString();
        String myName = new String("IGNOU MCA STUDENT");
        System.out.println("Length of myName:"+myName.length());
        System.out.println("myName: "+myName);
        System.out.println("reverseMe call: "+R.reverseMe(myName));
    }
}
```

Output:

Length of myName:17

MyName:IGNOU MCA STUDENT

Length of dest:17

Capacity of dest:17

Capacity of dest:17

Length of dest:7

Reverse call:TNEDUTS ACM UONGI

Note:

- i. In the reverseMe() method of above StringBuffer's to String() method is used to convert the StringBuffer to a String object before returning the String.
- ii. You should initialize a StringBuffer's capacity to a reasonable first guess, because it minimizes the number of times memory to be allocated for the situation when the appended character causes the size of the StringBuffer to grow beyond its current capacity. Because memory allocation is a relatively expensive operation, you can make your code more efficient by initializing a StringBuffer's capacity to a reasonable first guess size.

(c) Write five differences between an interface and an abstract class.

5

Refer to Chapter-8, Q-4, Page no.-111

(d) What is method overloading ? Explain how a method is overloaded in Java, with an example.

5

Refer to Chapter-5, Q-3, Page no.-67

**MCS-024 : OBJECT ORIENTED TECHNOLOGIES
AND JAVA PROGRAMMING**
December, 2006

Note : Question number 1 is **compulsory**. Attempt any **three** questions from the rest.

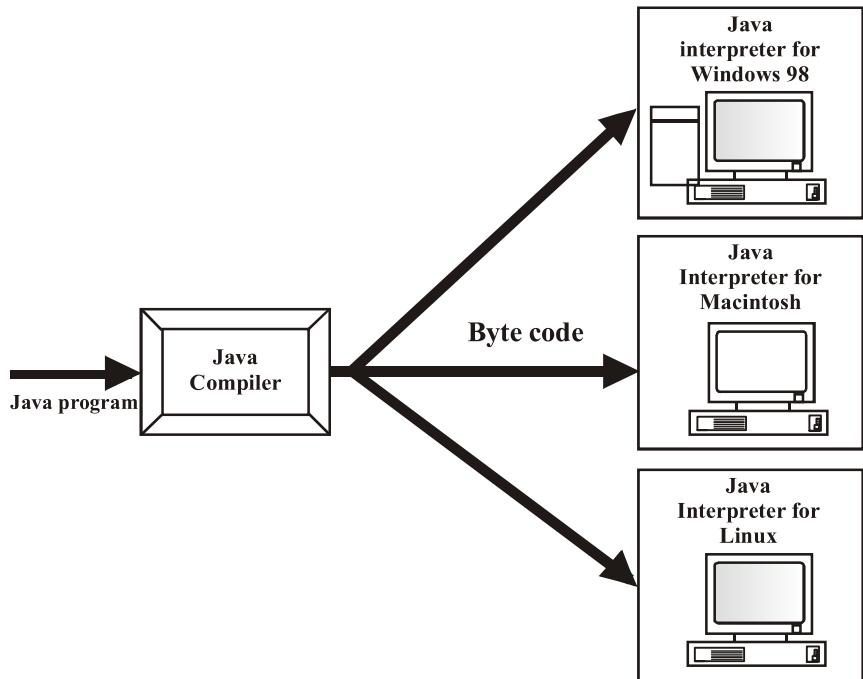
1. (a) What is the Java Virtual Machine ? How does it make Java machine independent ?

5

Ans. First : Refer to Chapter-2, Q.No.-3, Page No.-18

Java is platform independent as it is not dependent upon a particular architecture for execution of its networked programs. Java compiler converts the source code into highly optimized set of instructions called byte code. This is different from normally formed executable code by a compiler.

The specification of bytecode remains the same for any environment (architecture). This bytecode can be run at any machine having Java runtime system called JVM (Java Virtual Machine). It is an interpreter that executes the bytecode. Details of JVM may vary from system to system but bytecode remains same.



Java is Machine Independent and Architecture Neutral

(b) What is Garbage Collection ? Can the programmer explicitly invoke the garbage collector ? List the method with the help of an example which invokes the garbage collector. 6

Ans. Garbage collection : A software routine that searches memory for areas of inactive data and instructions in order to reclaim that space for the general memory pool (the heap). Operating systems may or may not provide this feature. For example, Windows does not do automatic garbage collection which requires that the programmer specifically deallocates memory in order to release it. If a program continues to allocate memory for data buffers and eventually exceeds the physical memory capacity, the operating system then has to place parts of the program in virtual memory (on disk) in order to continue, which slows down processing. Deallocating memory after a routine no longer needs it is a tedious task and programmers often forget to do it or do not do it properly. Java performs automatic garbage collection without programmer intervention, which eliminates this coding headache.

The Java Virtual Machine performs garbage collection on its own so no need to worry about it. No need to set objects to null manually for them to be garbage collected. In fact, if these null objects are still in scope, the garbage collector won't get them anyway. The garbage collector will automatically deallocate memory for objects as they go out of scope naturally. You can always run `System.gc()`.

Garbage collector can be invoked explicitly by the programmer when needed. Sometimes it is required to take independent resources from some object before the garbage collector takes the object. To perform this operation, a method named `finalize()` is used. `Finalize()` is called by the garbage collector when it determines no more references to the object exist.

Your class can provide for its finalization simply by defining and implementing a `finalize()` method in your class. Your `finalize()` method must be declared as follows:

```
protected void finalize() throws throwable
```

This class opens a file when it is constructed:

```
Class openAfile
{
    FileInputStream aFile=null;
    openAfile(String filename)
    {
        try
        {
```

```
aFile = new FileInputStream(filename);
}
catch(Java.io.FileNotFoundException e)
{
System.err.println("could not open file"+filename);
}
}
}
```

To avoid accidental modification or other related problem the openAfile class should close the file when it is finalized. Implementation of finalize() method for the class is as follows:

```
protected void finalize() throws throwable
{
if(aFile != null)
{
aFile.close();
aFile=null;
}
}
```

(c) Explain the following keywords with examples:

8

(i) this

Ans. Refer to Chapter-6, Q.No.-3, Page No.-81

(ii) abstract

Ans. Refer to Chapter-6, Q.No.-6, Page No.-86

(iii) final

Ans. Final variable : A variable can be declared as final. Doing so prevents its contents from being modified. A final variable has to be initialized when it is declared (in this usage, final is similar to const in C / C++ / C#).

final int size= 350 ;

Variables declared as final do not occupy memory on a per instance basis. Thus a final variable is essentially a constant.

Final method: While method overriding is one of Java's most powerful features, there will be times when you will want to prevent it from occurring. To disallow a method from being overridden, specify final as a modifier at the start of declaration. Methods declared as final cannot be overridden.

```
class A {
```

```

final void math() {
    System.out.println("This is a final method.");
}

class B extends A {
    void meth() { // ERROR! Can't override.
        System.out.println("ILLEGAL");
    }
}

```

Final class : When you want to prevent a class from being inherited, and then proceed the class declaration with final. Declaring a class as final implicitly declares all of its method as final.

```

final class A {
    //...
}

// the following code is illegal.
class B extends A {
    //ERROR! Can't subclass A
    //....
}

```

The above code implies it is illegal for B to inherit A since A is declared as final.

(iv) super

Ans. Refer to Chapter-6, Q.No.-4, Page No.-82

(d) What is object serialization ? Write a Java program that writes the state of an object to a file which includes user name and password. Also ensure that the password is not stored in the file. 6

Ans. Object serialization : It takes all the data attributes, writes them out as an object, and reads them back in as an object. For an object to be saved to a disk file it needs to be converted to a serial form. An object can be used with streams by implementing the serializable interface. The serialization is used to indicate that object of that class can be saved and retrieved in serial form. Object serialization is quite useful when you need to use object persistence. By object persistence, the stored object continues to serve the purpose even when no java program is running and stored information can be retrieved in a program so it can resume functioning unlike the other objects that cease to exist when

object stops running.

DataOutputStream and DataInputStream are used to write each attribute out individually, and then can read them back in on the other end. But to deal with the entire object, not its individual attributes, store away an object or send it over a stream of objects. Object serialization takes the data members of an object and stores them away or retrieves them, or send them over a stream. ObjectInput interface is used for input, which extends the DataInput interface, and ObjectOutput interface is used for output, which extends DataOutput. You are still going to have the methods readInt(), and so forth. ObjectInputStream, which implements ObjectInput, is going to read what ObjectOutputStream produces.

Working of object serialization : For ObjectInput and ObjectOutput interface, the class must be serializable. The serializable characteristic is assigned when a class is first defined. Your class must implement the serializable interface. This marker is an interface that says to the java virtual machine that you want to allow this class to be serializable.

```
//Program
import Java.io.*;
import Java.util.*;
public class SerialDemo implements Serializable
{
    private Date date=new Date();
    private String username;
    private transient String password;
    SerialDemo(String name, String pwd)
    {
        username = name;
        password = pwd;
    }
    public String toString()
    {
        String pwd = (password == null) ? "(n/a)" : password;
        return "Logon info: \n " + " Username:" + username +
        "\n Data: " + date + "\n Password:" +pwd;
    }
    public static void main(String[] args)
        throws IOException, ClassNotFoundException
    {
        SerialDemo a = new SerialDemo("Java", "sun");
        System.out.println("Login is = " + a);
        ObjectOutputStream o = new ObjectOutputStream(new
```

```

FileOutputStream("Login.out"));
o.writeObject(a);
o.close();
// Delay:
int seconds = 10;
long t = System.currentTimeMillis() + seconds * 1000;
while(System.currentTimeMillis() < t)
;
// Now get them back:
ObjectInputStream in = new ObjectInputStream(new
FileInputStream("Login.out"));
System.out.println("Recovering object at " + new Date());
a = (SerialDemo)in.readObject();
System.out.println("Login a = " + a);
}
}

```

Output:

```

Login is = Login info:
Username: Java
Date: Thu Feb 03 04:06:22 GMT+05:30 2005
Password: sun
Recovering object at Thu Feb 03 03:06:32 GMT+05:30 2007
Login a = Logon info:
Username: Java
Date: Thu Feb 03 04:06:22 GMT+05:30 2007
Password : (n\ a)

```

(e) What is an applet ? What are its life cycle methods ?

5

Ans. An applet is an application designed to be transmitted over the internet and executed by a java compatible web browser. An applet is actually a tiny java program, dynamically downloaded across the network, just like an image, sound file, or video clip.

When an applet begins, the AWT calls the following methods in this sequence:

- 1 init()
- 2 start()
- 3 paint()

When an applet is terminated, the following methods in this sequence are called:

- 1 stop()
- 2 destroy()

init()

The **init()** method is the first method to be called. This is where you should initialize variables. This method is called only once during the run time of your applet.

start()

The **start()** method is called after **init()**. It is also called to restart an applet after it has been stopped. Whereas **init()** is called once—the first time an applet is loaded—**start()** is called each time an applet's HTML document is displayed onscreen. So, if a user leaves a web page and comes back, the applet resumes execution at **start()**.

paint()

The **paint()** method is called each time your applet's output must be redrawn. This situation can occur for several reasons. For example, the window in which the applet is running may be overwritten by another window and then uncovered. Or the applet window may be minimized and then restored. **paint()** is also called when the applet begins execution. Whatever the cause, whenever the applet must redraw its output, **paint()** is called. The **paint()** method has one parameter of type **Graphics**. This parameter will contain the graphics context, which describes the graphics environment in which the applet is running. This context is used whenever output to the applet is required.

stop()

The **stop()** method is called when a web browser leaves the HTML document containing the applet—when it goes to another page, for example. When **stop()** is called, the applet is probably running. You should use **stop()** to suspend threads that don't need to run when the applet is not visible. You can restart them when **start()** is called if the user returns to the page.

destroy()

The **destroy()** method is called when the environment determines that your applet needs to be removed completely from memory. At this point, you should free up any resources the applet may be using. The **stop()** method is always called before **destroy()**.

(f) What is an interface ? What is the nature of methods and variables defined in an interface ? How is an interface different from an abstract class ?

5

Ans. Refer to Chapter-8, Q.No.-3&4, Page No.-110-111

(g) What is JDBC ? List the classes in Java.SQL package which are used to execute any SQL Query, with an explanation. 5

Ans. JDBC is an API for the Java programming language that defines how a client may access a database. It provides methods for querying and updating data in a database. JDBC is oriented towards relational databases. JDBC has been part of the Java Standard Edition since the release of JDK 1.1. The JDBC classes are contained in the Java package `java.sql`.

JDBC allows multiple implementations to exist and be used by the same application. The API provides a mechanism for dynamically loading the correct Java packages and registering them with the JDBC Driver Manager. The Driver Manager is used as a connection factory for creating JDBC connections.

JDBC connections support creating and executing statements. These statements may be update statements such as SQL CREATE, INSERT, UPDATE and DELETE or they may be query statements using the SELECT statement. Additionally, stored procedures may be invoked through a statement. Statements are one of the following types:

Statement – the statement is sent to the database server each and every time.

PreparedStatement – the statement is cached and then the execution path is pre determined on the database server allowing it to be executed multiple times in an efficient manner.

CallableStatement – used for executing stored procedures on the database. Update statements such as INSERT, UPDATE and DELETE return an update count that indicates how many rows were affected in the database. These statements do not return any other information.

Query statements return a JDBC row result set. The row result set is used to walk over the result set. Individual columns in a row are retrieved either by name or by column number. There may be any number of rows in the result set. The row result set has metadata that describes the names of the columns and their types.

There is an extension to the basic JDBC API in the `javax.sql` package that allows for scrollable result sets and cursor support among other things.

The `java.sql` package contains API for the following:

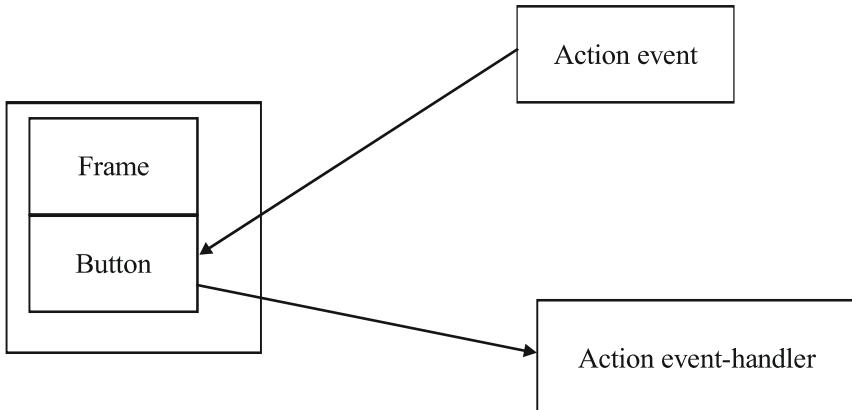
- Making a connection with a database via the `DriverManager` facility
 - `DriverManager` class — makes a connection with a driver
 - `SQLPermission` class — provides permission when code running within a Security Manager, such as an applet, attempts to set up a logging stream through the `DriverManager`
 - `Driver` interface — provides the API for registering and connecting drivers based on JDBC technology (“JDBC drivers”); generally used only by the `DriverManager` class

- Sending SQL statements to a database
 - Statement — used to send basic SQL statements
 - PreparedStatement — used to send prepared statements or basic SQL statements (derived from Statement)
 - CallableStatement — used to call database stored procedures (derived from PreparedStatement)
 - Connection interface — provides methods for creating statements and managing connections and their properties
 - Savepoint — provides savepoints in a transaction
- Retrieving and updating the results of a query
 - ResultSet interface
- Standard mappings for SQL types to classes and interfaces in the Java programming language
 - Array interface — mapping for SQL ARRAY
 - Blob interface — mapping for SQL BLOB
 - Date class — mapping for SQL DATE
 - Ref interface — mapping for SQL REF
 - RowId interface — mapping for SQL ROWID
 - Struct interface — mapping for SQL STRUCT
 - Time class — mapping for SQL TIME
 - Timestamp class — mapping for SQL TIMESTAMP
 - Types class — provides constants for SQL types
- Custom mapping an SQL user-defined type (UDT) to a class in the Java programming language
 - SQLData interface — specifies the mapping of a UDT to an instance of this class
 - SQLInput interface — provides methods for reading UDT attributes from a stream
 - SQLOutput interface — provides methods for writing UDT attributes back to a stream
- Exceptions
 - SQLException — thrown by most methods when there is a problem accessing data and by some methods for other reasons
 - SQLWarning — thrown to indicate a warning
 - BatchUpdateException — thrown to indicate that not all commands in a batch update executed successfully

2 (a) Explain Event Delegation Model with Event Classes and Listener Interfaces commonly used. 8

Ans. The Event Delegation Model came into existence with JDK1.1. In this model, events are sent to the component from where the events originated. The component registers a listener object with the program. The listener object contains appropriate event-handlers that receive and process the events. By registering a listener object with the program, the component enables the delegation of events to the listener object for processing.

The following diagram illustrates the JDK1.2 event model:



Every event has a corresponding listener interface that specifies the methods that are required to handle the event. Event objects are reported to registered listeners. To enable a component to handle events, you must register an appropriate listener for that component. The `java.awt.event` package contains definitions of all the event classes and listener interfaces.

Event types are encapsulated in a class hierarchy rooted at `java.util.EventObject`. An event is propagated from a “Source” object to a “Listener” object by invoking a method on the listener and passing in the instance of the event subclass which defines the event type generated.

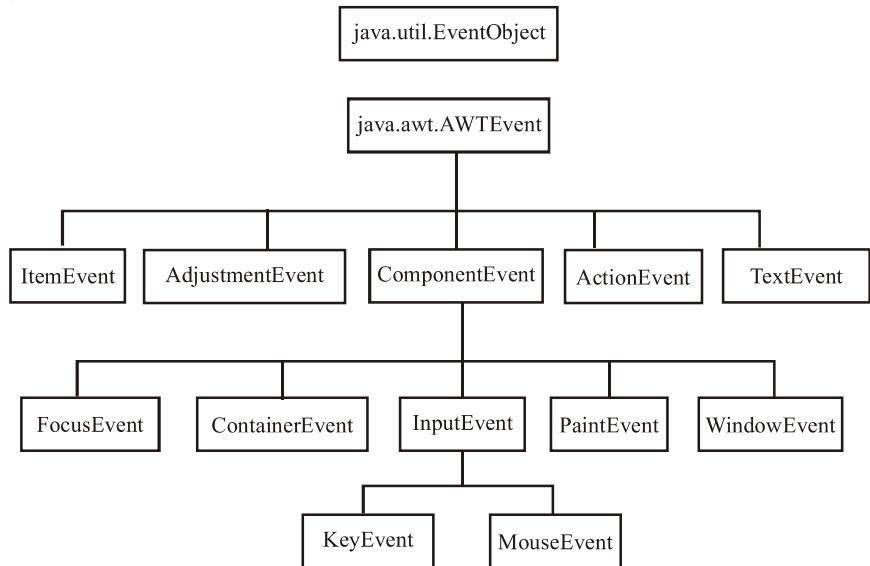
A Listener is an object that implements a specific `EventListener` interface extended from the generic `java.util.EventListener`. An `EventListener` interface defines one or more methods which are to be invoked by the event source in response to each specific event type handled by the interface.

An Event Source is an object which originates or “fires” events. The source defines the set of events it emits by providing a set of `set<EventType>Listener` (for single-cast) and/or `add<EventType>Listener` (for multi-cast) methods which are used to register specific listeners for those events.

In an AWT program, the event source is typically a GUI component and the

listener is commonly an “adapter” object which implements the appropriate listener (or set of listeners) in order for an application to control the flow/handling of events. The listener object could also be another AWT component which implements one or more listener interfaces for the purpose of hooking GUI objects up to each other.

Event Classes : The EventObject class is at the top of the event class hierarchy. It belongs to the java.util package. Most other event classes are present in the java.awt event package. The hierarchy of the event classes is shown below. The java.util.EventObject and the java.awt.AWTEvent classes do not belong to the java.awt.event package. The hierarchy of the JDK 1.2 event classes is given below.



The getSource () method of the EventObject class returns the object that initiated the event. The getID () method returns the event ID that represents the nature of the event. For example, if a mouse event occurs, you can find out whether the event was a click, a drag, a move, a press, or a release from the event object.

Let us see when various events are generated:

1. An ActionEvent object is generated when a component is activated.
2. An AdjustmentEvent object is generated when scrollbars and other adjustable elements are used.
3. A ContainerEvent object is generated when components are added to or removed from a container.
4. A FocusEvent object is generated when a component receives focus for input.

5. An ItemEvent object is generated when an item from a list, a choice, or a check box is selected.
6. A KeyEvent object is generated when a key on the keyboard is pressed.
7. A WindowEvent object is generated when a window activity, like minimizing, occurs.
8. A MouseEvent object is generated when the mouse is used.
9. A PaintEvent object is generated when a component is painted.
10. A TextEvent object is generated when the text of a text component is modified.

(b) Create a Java Applet with two text boxes and a button. Write a program so that on entering two numbers in the text box and clicking of a button it displays the greater of the two numbers. 7

Ans.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
public class userin extends Applet implements ActionListener
{
    TextField text1, text2;
    public void init()
    {
        text1=new TextField(8);
        text2=new TextField(8);
        add(text1);
        add(text2);
        text1.setText("0");
        text2.setText("0");

        text1.addActionListener(this);
        text2.addActionListener(this);
    }
    public void paint(Graphics g)
    {
        int x=0,y=0,z=0;
        String s1, s2, s;
        g.drawString("Input a number in each box",10, 30);
        try
        {
            s1=text1.getText ();
            x=Integer.parseInt(s1);
            s2 =text2.getText ();

```

```
y=Integer.parseInt (s2);

}

catch(Exception ex)
{
}

if(x>y)
{
z=x;
}
else
{
z=y;
}

s=String.valueOf(z);
g.drawString("The larger no is :" ,10,75);
g.drawString(s,100,75);
}

public void actionPerformed(ActionEvent ae)
{
repaint();

}

}
```

The html document is as follows:

```
<html>
<applet
code = userin.class
width=400
height=400>
</applet>
</html>
```

(c) What is a layout Manager ? Create a frame that has four buttons on all four sides, i.e. north, east, south and west. 5

Ans. Layout managers, used in Widget toolkits, are software components which have the ability to layout widgets by their relative positions without using distance units.

A lot of popular Widget toolkits have this ability by default, making it often more natural to use than by defining their absolute (on the screen) or relative

(to the parent) position in pixels or common distance units. Widget toolkits that provide this possibility can be separated in two groups:

- Those where the layout behavior is coded in special Graphic Containers. This is the case in XUL or the .NET Framework Widget toolkit (called Windows.Forms).
- Those where the layout behavior is coded in layout managers, that can be applied to any Graphic Container. This is the case in the Swing widget toolkit that is part of the Java API.

```

import java.awt.*;
import javax.swing.*;
public class test extends JFrame
{
public test()
{
this.setTitle("Borderlayout");
Container contentpane=this.getContentPane();
contentpane.setLayout(new BorderLayout(2,3));
this.add("South",new Button("start"));
this.add("North",new Button("stop"));
this.add("East",new Button("Play"));
this.add("West",new Button("Pause"));
}

public static void main(String[] args)
{
test example = new test();
example.setVisible(true);
}
}

```

3. (a) What are a class and object ? How is an object related to and different from a class ? 6

Ans. Refer to Chapter-1, Q.No.-3, Page No.-5

(b) Write a program that displays all the prime numbers between 2 and 100. 6

Ans. class prime

{

```
public static void main(String args[])

```

```

{
int a,num,i;
for(a=2;a<=100;a++)
{
num=a;
i=2;
while(i<num)
{
if(num%i==0)
{
break;
}
i++;
}//while
if(i==num)
{
System.out.println("no is prime"+num);
}
}//for

}//psvm
}//class

```

(c) Explain the concept of polymorphism. Create an Exception class that is thrown whenever the string entered is not a palindrome. Write a program that accepts the command line argument and uses the custom defined exception.

8

Ans. First : Refer to Chapter-5, Q.No.-5, Page No.-70

```

class myException extends Exception
{
myException(String message)
{
super(message);
}

public class palindrome
{
public static void main(String args[])
{

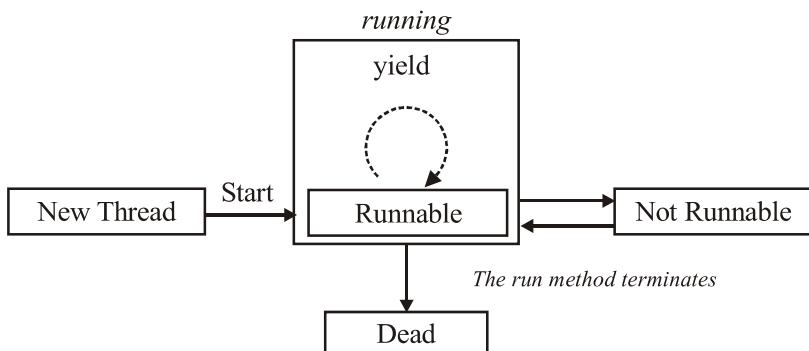
```

```
String st;
int len=0;
st=args[0];
len=st.length();
try
{
for(int i=0;i<len/2+1;i++)
{
if(st.charAt(i)!=st.charAt(len-i-1))
{
throw new myException("String is not a palindrome");
}
else
System.out.println("String is a palindrome");
}
}//try
catch(myException e)
{
System.out.println(e.getMessage());
}
```

4. (a) What is a thread ? Explain the life cycle of a thread.

4

Ans. A thread is a single sequence of execution that can run independently in an application. The following figure shows the states that a thread can be in during its life and illustrates which method calls cause a transition to another state.



Creating a Thread

The application in which an applet is running calls the applet's start method when the user visits the applet's page. The *Clock* applet creates a Thread,

clockThread, in its start method:

```
public void start() {
    if (clockThread == null) {
        clockThread = new Thread(this, "Clock");
        clockThread.start();
    }
}
```

After the above statement has been executed, clockThread is in the New Thread state. A thread in this state is merely an empty Thread object; no system resources have been allocated for it yet. When a thread is in this state, you can only start the thread. Calling any method besides start when a thread is in this state makes no sense and causes an IllegalThreadStateException. In fact, the runtime system throws an IllegalThreadStateException whenever a method is called on a thread and that thread's state does not allow for that method call.

Note that the Clock instance is the first argument to the thread constructor. The first argument to this thread constructor must implement the Runnable interface and becomes the thread's target. The clock thread gets its run method from its target Runnable object — in this case, the Clock instance. The second argument is just a name for the thread.

Starting a Thread

Now consider the next line of code in Clock's start method shown here highlighted:

```
public void start() {
    if (clockThread == null) {
        clockThread = new Thread(this, "Clock");
        clockThread.start();
    }
}
```

The start method creates the system resources necessary to run the thread, schedules the thread to run, and calls the thread's run method. clockThread's run method is defined in the Clock class.

After the start method has returned, the thread is “running.” Here's another look at Clock's run method:

```
public void run() {
    Thread myThread = Thread.currentThread();
    while (clockThread == myThread) {
        repaint();
    }
    try {
        Thread.sleep(1000);
    } catch (InterruptedException e) {
```

```
// The VM doesn't want us to sleep anymore,
// so get back to work
}
}
}
```

Clock's run method loops while the condition `clockThread == myThread` is true.

Within the loop, the applet repaints itself and then tells the thread to sleep for one second (1000 milliseconds). An applet's repaint method ultimately calls the applet's paint method, which does the actual update of the applet's display area. The Clock paint method gets the current time, formats, and displays it:

```
public void paint(Graphics g) {
    //get the time and convert it to a date
    Calendar cal = Calendar.getInstance();
    Date date = cal.getTime();
    //format it and display it
    DateFormat dateFormatter = DateFormat.getTimeInstance();
    g.drawString(dateFormatter.format(date), 5, 10);
}
```

Making a Thread Not Runnable

A thread becomes Not Runnable when one of these events occurs:

- Its sleep method is invoked.
- The thread calls the wait method to wait for a specific condition to be satisfied.
- The thread is blocking on I/O.

The `clockThread` in the Clock applet becomes Not Runnable when the run method calls sleep on the current thread:

```
public void run() {
    Thread myThread = Thread.currentThread();
    while (clockThread == myThread) {
        repaint();
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            // The VM doesn't want us to sleep anymore,
            // so get back to work
        }
    }
}
```

During the second that the `clockThread` is asleep, the thread does not run, even if the processor becomes available. After the second has elapsed, the thread becomes Runnable again; if the processor becomes available, the thread

begins running again.

For each entrance into the Not Runnable state, a specific and distinct exit returns the thread to the Runnable state. An exit works only for its corresponding entrance. For example, if a thread has been put to sleep, the specified number of milliseconds must elapse before the thread becomes Runnable again. The following list describes the exit for every entrance into the Not Runnable state.

- If a thread has been put to sleep, the specified number of milliseconds must elapse.
- If a thread is waiting for a condition, then another object must notify the waiting thread of a change in condition by calling `notify` or `notifyAll`.
- If a thread is blocked on I/O, the I/O must complete.

Stopping a Thread

Although the `Thread` class does contain a `stop` method, this method is deprecated and should not be used to stop a thread because it is unsafe. Rather, a thread should arrange for its own death by having a `run` method that terminates naturally. For example, the while loop in this `run` method is a finite loop: It will iterate 100 times and then exit:

```
public void run() {
    int i = 0;
    while (i < 100) {
        i++;
        System.out.println("i = " + i);
    }
}
```

A thread with this `run` method dies naturally when the loop completes and the `run` method exits.

The Clock applet thread arranges for its own death.:.

```
public void run() {
    Thread myThread = Thread.currentThread();
    while (clockThread == myThread) {
        repaint();
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            //the VM doesn't want us to sleep anymore,
            //so get back to work
        }
    }
}
```

The exit condition for this `run` method is the exit condition for the while loop because there is no code after the while loop:

```
while (clockThread == myThread) {
```

This condition indicates that the loop will exit when the currently executing thread is not equal to clockThread.

When you leave the page, the application in which the applet is running calls the applet's stop method. This method then sets the clockThread to null, thereby telling the main loop in the run method to terminate:

```
public void stop() { // applets' stop method  
    clockThread = null;  
}
```

If you revisit the page, the start method is called again and the clock starts up again with a new thread. Even if you stop and start the applet faster than one iteration of the loop, clockThread will be a different thread from myThread and the loop will still terminate.

(b) Create an applet with a rectangle that is gradually colored after a second, using the sleep method of thread. 6

Ans. import java.awt.*;

```
import java.applet.*;
```

```
public class rectangle extends Applet implements Runnable
```

```
{
```

```
Thread t=null;
```

```
public void start()
```

```
{
```

```
t=new Thread(this);
```

```
t.start();
```

```
}
```

```
public void run()
```

```
{
```

```
}
```

```
public void paint(Graphics g)
```

```
{
```

```
try
```

```
{
```

```
Color c1=new Color(255,0,0);
```

```
g.drawRect(10,10,60,50);
```

```
g.fillRect(10,10,60,50);
```

```
Thread.sleep(1000);
```

```
g.setColor(c1);
```

```
g.drawRect(10,10,60,50);
```

```
g.fillRect(10,10,60,50);
```

```

}
catch(Exception e)
{
}
}
}//class

```

(c) What will be the result of attempting to compile and run the following code :

4

```

(i) public class IfTest{
public static void main (string args [ ]){
if (true)
if (false)
system..out.println ("a");
else
system.out.println ("b");
}
}

```

Ans.

Output:

b

```

(ii) class MyClass {
public static void main (string args [ ]){
boolean b=false;
int i=1 ;
do{
i++;
b=!b;
} while (b);
system.out.println (i);
}
}

```

Ans.

output:

3

(d) Explain the hierarchy of stream classes in the IO package. What is the difference between Binary streams and Text streams ?

6

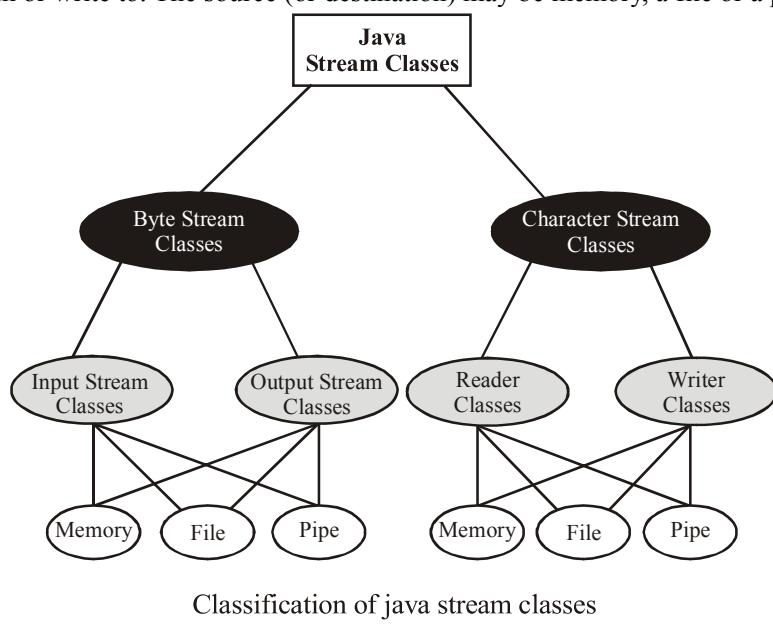
Ans. Stream Classes : The java.io package contains a large number of stream classes that provide capabilities for processing all types of data. These classes

may be categorized into two groups based on the data type on which they operate.

1. Byte stream classes that provide support for handing I/O operations on bytes.

2. Character stream classes that provide support for managing I/O operations on characters.

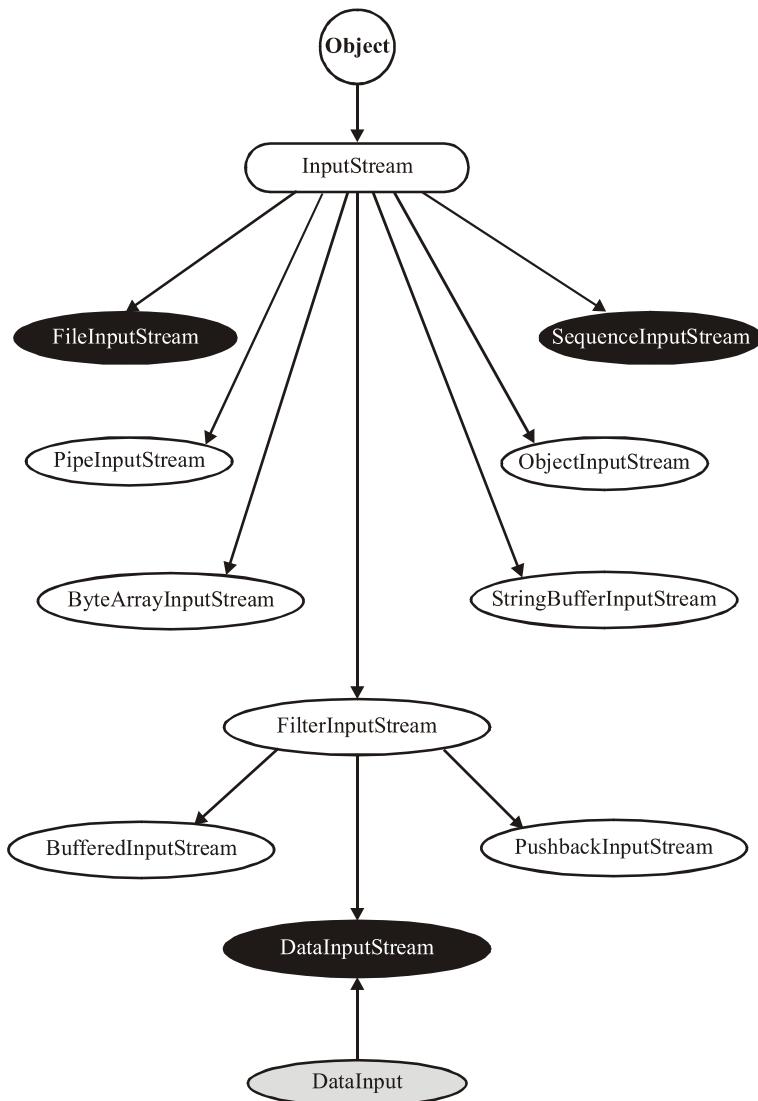
These two groups may further be classified based on their purpose. Figure shows how stream classes are grouped based on their functions. Byte stream and character stream classes contain specialized classes to deal with input and output operations independently on various types of devices. We can also cross-group the streams based on the type of source or destination they read from or write to. The source (or destination) may be memory, a file or a pipe.



Classification of java stream classes

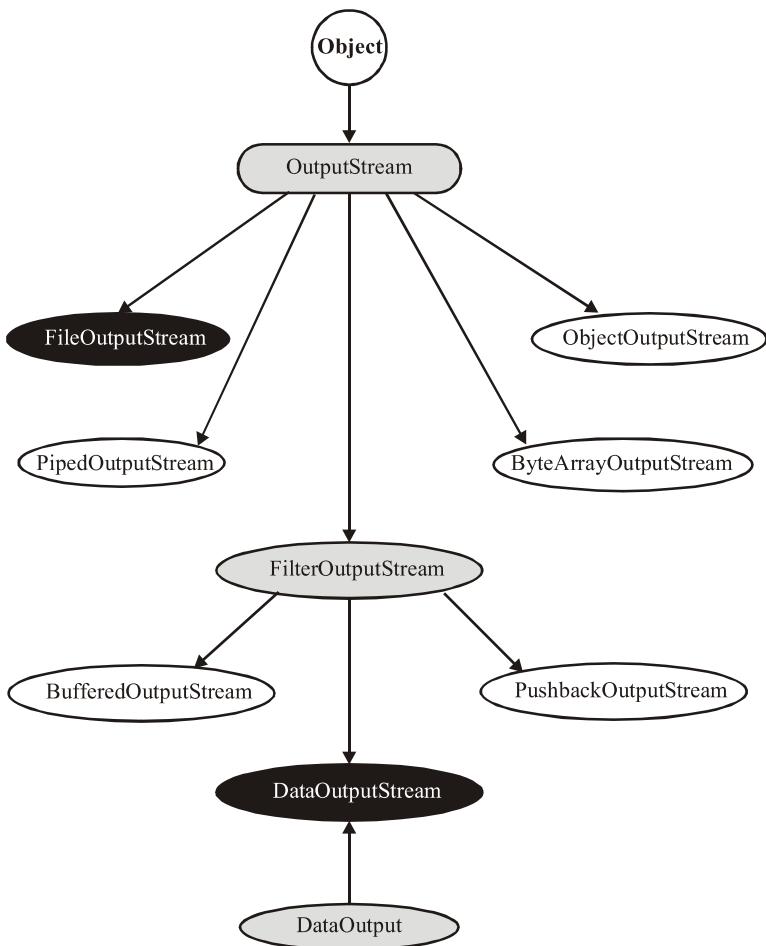
Byte Stream Classes : Byte stream classes have been designed to provide functional features for creating and manipulating streams and files for reading and writing bytes. Since the streams are unidirectional, they can transmit bytes in only one direction and therefore java provides two kinds of byte stream classes: input stream classes and output stream classes.

Input Stream Classes : Input stream classes that are used to read 8-bit include a supper class known as input stream and a number of subclasses for supporting various input-related functions. Figure next shows the class hierarchy of input stream classes.



Hierarchy of input stream classes

Output stream classes : Output stream classes are derived from the base class output stream as shown in Fig. next. Like inputstream, the outputstream is an abstract class and therefore we cannot instantiate it. The several subclasses of the outputstream can be used for performing the output operations.



Hierarchy of output stream classes

- Writing bytes
- Closing
- Flushing stream

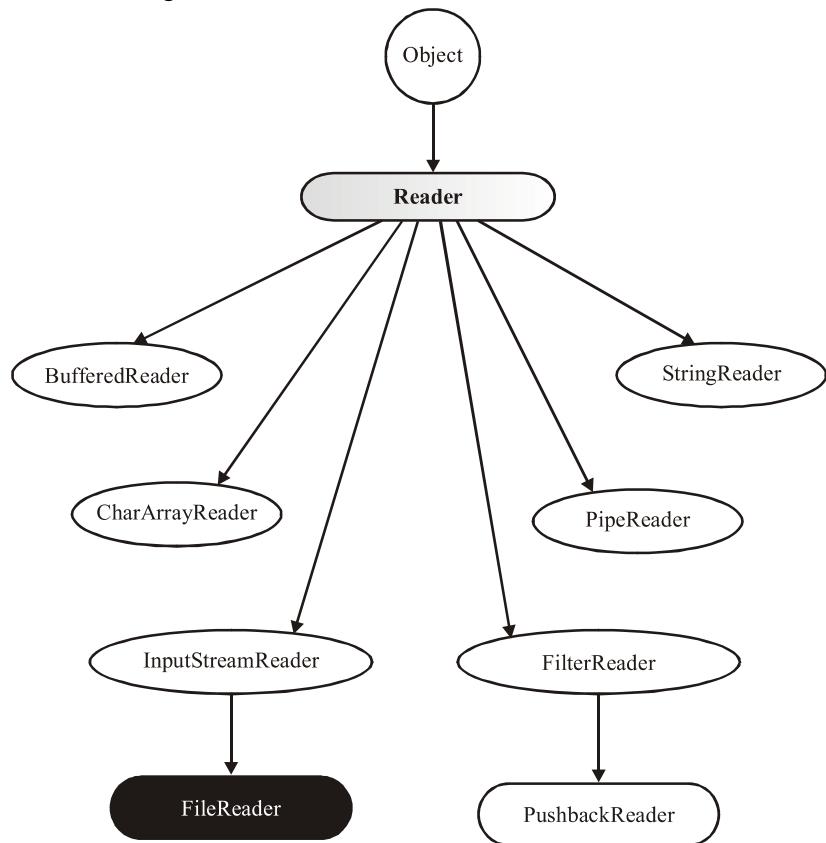
Character Stream Classes : Character stream classes were not a part of the language when it was released in 1995. They were added later when the version 1.1 was announced. Character streams can be used to read and write 16-bit Unicode characters. Link byte streams, there are two kinds of character stream classes, namely, reader stream classes and writer stream classes.

Reader Stream Classes : Reader stream classes are designed to read character from the files. Reader class is the base class for all other classes in this group as shows in Fig. next. These classes are functionally very similar to the input

stream classes, except input stream use bytes as their fundamental unit of information, while reader stream use characters.

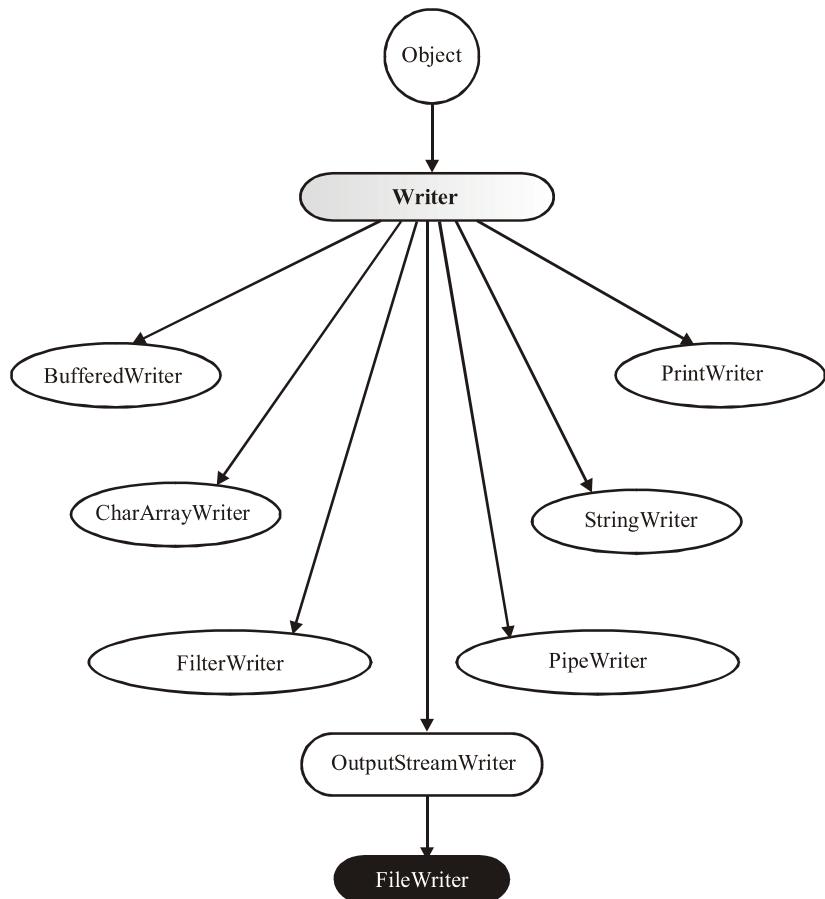
The Reader class contains methods that are identical to those available in the InputStream class, except Reader is designed to handle character. Therefore, reader classes can perform all the functions implemented by the input stream classes.

Writer Stream Classes : Link output stream classes, the writer stream classes are designed to perform all output operations on files. Only difference is that while output stream classes are designed to write bytes, the writer stream classes are designed to write characters.



Hierarchy of reader stream classes

The Writer class is an abstract class which acts as a base class for all the other writer stream defining methods that are identical to those in OutputStream class.



Hierarchy of writer stream classes

Text and Binary Streams : A text stream consists of one or more lines of text that can be written to a text-oriented display so that they can be read. When reading from a text stream, the program reads an NL (newline) at the end of each line. When writing to a text stream, the program writes an NL to signal the end of a line.

Thus, positioning within a text stream is limited. You can obtain the current file-position indicator by calling fgetpos or ftell. You can position a text stream at a position obtained this way, or at the beginning or end of the stream, by calling fsetpos or fseek. Any other change of position might well be not supported.

A **binary stream** consists of one or more bytes of arbitrary information. You can write the value stored in an arbitrary object to a (byte-oriented) binary

stream and read exactly what was stored in the object when you wrote it. The library functions do not alter the bytes you transmit between the program and a binary stream. They can, however, append an arbitrary number of null bytes to the file that you write with a binary stream. The program must deal with these additional null bytes at the end of any binary stream.

Thus, positioning within a binary stream is well defined, except for positioning relative to the end of the stream. You can obtain and alter the current file-position indicator the same as for a text stream. Moreover, the offsets used by ftell and fseek count bytes from the beginning of the stream (which is byte zero), so integer arithmetic on these offsets yields predictable results

5. (a) What is a Servlet ? Create a servlet that accepts the user name from the client and prints “Hello” followed by the actual user name of the client.

8

Ans. First : Refer to Chapter-11, Q.No.-2, Page No.-140

```
import java.io.*;
import javax.servlet.*;
public class HelloServlet extends GenericServlet{
public void service(ServletRequest request,ServletResponse response) throws
ServletException, IOException{
response.setContentType("text/html");
PrintWriter pw=response.getWriter();
pw.println("<B>Hello");
pw.close();
}
}
```

(b) Write short notes on :

6

(i) URL

Ans. Refer to Chapter-10, Q.No.-5, Page No.-130

(ii) Proxy Server

Ans. A proxy server is a server (a computer system or an application program) which services the requests of its clients by forwarding requests to other servers. A client connects to the proxy server, requesting some service, such as a file, connection, web page, or other resource, available from a different server. The proxy server provides the resource by connecting to the specified server and requesting the service on behalf of the client. A proxy server may optionally alter the client's request or the server's response, and sometimes it may serve the request without contacting the specified server.

A proxy server that passes all requests and replies unmodified is usually called a gateway or sometimes *tunnelling proxy*.

A proxy server can be placed in the user's local computer or at specific key points between the user and the destination servers or the Internet.

(iii) Internet Address

Ans. Internet address: Internet address is the address of a device attached to

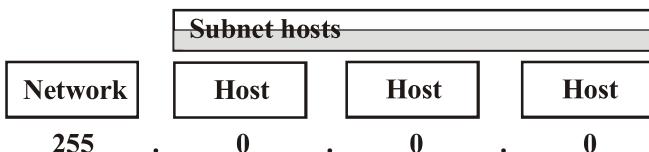
an IP network (TCP/IP network). Every client, server and network device must have a unique IP address for each network connection (network interface). Every IP packet contains a source IP address and a destination IP address. IP addresses are written in “dotted decimal” notation, which is four sets of numbers separated by periods.

Class A, B and C

Class Number	Class Range	Maximum Number of Networks	Maximum Hosts per Network	Number of Bits used in Network/Host ID ID
A	1-126	127	16,777,214	7/24
B	128-191	16,383	65,534	14/16
C	192-223	2,097,151	254	21/8
127 reserved for loopback test				

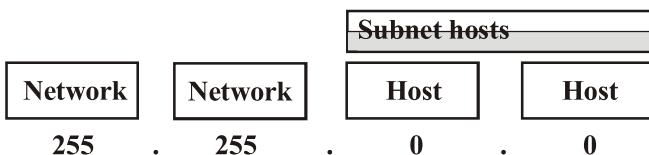
CLASS A (1-126)

Default subnet mask = 255.0.0.0



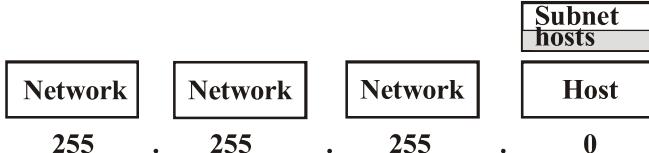
CLASS B (128-191)

Default subnet mask = 255.255.0.0



CLASS C (192-233)

Default subnet mask = 255.255.255.0



Networks, Subnets and Hosts

An IP address is first divided between networks and hosts. The host bits are further divided between subnets and hosts.

(c) Explain RMI Architecture in detail.

6

Ans. Refer to Chapter-11, Q.No.-1, Page No.-137

**MCS-024 : OBJECT ORIENTED TECHNOLOGIES
AND JAVA PROGRAMMING**
June, 2007

Note : Question number 1 is **compulsory**. Attempt any **three** questions from the rest.

1. (a) What is java virtual machine? Explain.

4

Ans. Refer to Chapter-2, Q.No.-3, Page No.-18

(b) What is the difference between method overloading and method overriding? Explain with the help of examples.

3+3

Ans. Refer to Chapter-5, Q.No.-3,4&5, Page No.-67to70

(c) What is data abstraction? Why are classes known as abstract data types?

3+2

Ans. First : Refer to Chapter-1, Q.No.-11, Page No.-12

A major trend in computer science is the object-oriented paradigm, an approach to program design and implementation using collections of interacting entities called objects. Objects incorporate both data and operations. In this way they mimic things in the real world, which have properties (data) and behaviors (operations). Objects that hold the same kind of data and perform the same operations form a class.

Abstract data values are separated from abstract data type operations. If the values in the carrier set of an abstract data type can be reconceptualized to include not only data values but also abstract data type operations, then the elements of the carrier set become entities that incorporate both data and operations, like objects, and the carrier set itself is very much like a class. The object-oriented paradigm can thus be seen as an outgrowth of the use of abstract data types (classes).

(d) When is private specifier used in a java program? Explain with suitable examples.

5

Ans. Classes and packages are both means of encapsulating and containing the name space and scope of variables and methods. Packages act as containers for classes and other subordinate packages. Classes act as containers for data and code. The class is Java's smallest unit of abstraction. Because of the interplay between classes and packages, Java addresses four categories of visibility for class members:

- | Subclasses in the same package
- | Non-subclasses in the same package

- | Subclasses in different packages
- | Classes that are neither in the same package nor subclasses
- Anything declared **private** cannot be seen outside of its class.

Private

Same class : yes

Same package Subclass: No

Same package Non-subclass: No

Different Package Subclass: No

Different Package Non-subclass: No

Now : For example, refer to Chap8, page107

(e) What is the difference between an applet and java application program?

5

Ans. Refer to Chapter-2, Q.No.-1, Page No.-16

(f) What happens if we apply final keyword to

3

- (i) a class
- (ii) a function
- (iii) a variable

Ans. (i) Final class : When you want to prevent a class from being inherited, and then precede the class declaration with final. Declaring a class as final implicitly declares all of its method as final.

```
final class A {
    //...
}
```

// the following code is illegal.

```
class B extends A {
    //ERROR! Can't subclass A
    //....
}
```

The above code implies it is illegal for B to inherit A since A is declared as final

(ii) Final method : While method overriding is one of the Java's most powerful features, there will be times when you will want to prevent it from occurring. To disallow a method from being overridden, specify final as a modifier at the start of declaration. Methods declared as final cannot be overridden.

```
class A {
    final void meth() {
        System.out.println("This is a final method.");
    }
}
```

```

}
class B extends A {
void meth() { // ERROR! Can't override.
    System.out.println("ILLEGAL");
}
}

```

(iii) Final variable : A variable can be declared as final. Doing so prevents its contents from being modified. A final variable has to be initialized when it is declared (in this usage, final is similar to const in C / C++ / C#).

final int size= 350 ;

Variables declared as final do not occupy memory on a per instance basis. Thus a final variable is essentially a constant.

(g) Write the output of the following:

(i) x = -10, y = 1

x << 2

2

x >> y + 2

(ii) String S1 = "Hello";

String S2 = "Hello";

String S3 = "HELLO";

System.out.println (S1 + "equals" + S2 + "→" + S1.equals (S2));

System.out.println (S1 + "equals" + S3 + "→" + S1.equals (S3));

2

(iii) int i = 150;

byte b = byte (i*2);

2

Ans. (i) Output:

-40

-2

(ii) Output:

Hello equals Hello -> true

Hello equals HELLO -> false

(iii) It will show an error.

Byte class is a wrapper class for byte data type. The constructor is Byte(byte num)

(h) Write a program in java to print all the arguments in reverse order given at command line.

6

Ans. class CommandLine

{

```

public static void main(String args[])
{
int a=args.length;
for(int i=a-1; i>=0; i--)
System.out.println("args[" + i + "]: " +args[i]);
}
}

```

java CommandLine a b c d e f g

Ouput:

```

args[6]: g
args[5]: f
args[4]: e
args[3]: d
args[2]: c
args[1]: b
args[0]: a

```

2. (a) What are the differences in abstract classes and interfaces in java? Explain. 6

Ans. Refer to Chapter-8, Q.No.-4, Page No.-111

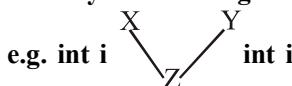
(b) What is finalize() method in java? Explain. 4

Ans. Refer to Chapter-5, Page No.-76

(c) What is a package? How do you create a package in java? Explain with example. 5

Ans. Refer to Chapter-8, Q.No.-1, Page No.-105

(d) Can you have two variables with the same name in two interfaces which you are using to derive a new interface?



Z

Explain your answer.

5

Ans. Yes, an interface may inherit multiple field variables with the same name. This leads to a single, ambiguous variable name. For example:

```

interface A {
    int x = 4;
}

interface B {
    int x = 43;
}

```

```

interface C extends A, B {
    int y = 22;
}
class Z implements C {
    public static void main (String[] argv) {
        System.out.println(x);      // Ambiguous
    }
}

```

In this example, the interface C inherits two variables named x. This is fine, as long as C does not refer to the variable x by its simple name in any of its declarations. If C needs to use x, it must qualify the name with the appropriate interface name (e.g., A.x). Class Z implements interface C, so it also has access to two variables named x. As a result, the use of x in main() is ambiguous. This problem can be resolved by qualifying the variable name with the appropriate interface name (e.g., B.x).

A class that implements multiple interfaces can also inherit multiple field variables with the same name. Again, this leads to a single, ambiguous variable name:

```

interface A {
    int x = 4;
}
interface B {
    int x = 43;
}
class Z implements A, B {
    public static void main (String[] argv) {
        System.out.println(x);      // Ambiguous
    }
}

```

The class Z implements both interface A and interface B, so it inherits two variables named x. As a result, the use of x in main() is ambiguous. This problem can again be resolved by qualifying the variable name with the appropriate interface name (e.g., B.x).

3. (a) What is the difference in throw and throws? Explain with the help of suitable examples. 6

Ans. Refer to Chapter-7, Q.No.-4, Page No.-93

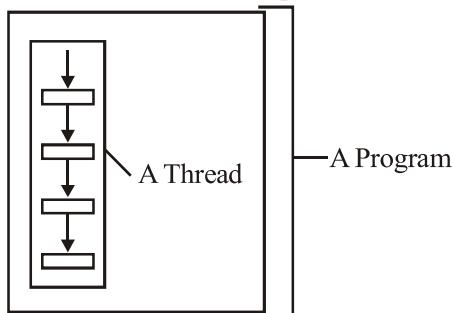
(b) What are the methods of creating threads in java? Explain. 6

Ans. Threads are light weight. They share the same address space and cooperatively share the same heavyweight process.

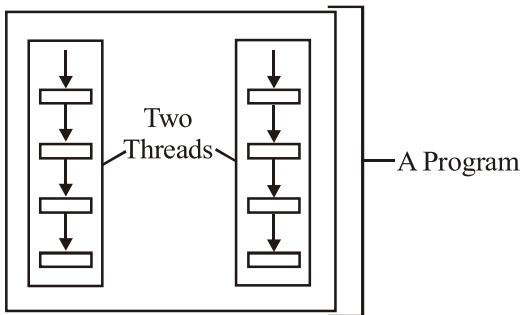
Multithreading enables you to write very efficient programs that make maximum

use of the CPU, because idle time can be kept to a minimum. This is especially important for the interactive, networked environment in which java operates, because idle time is common. .

A thread is similar to the sequential programs. A single thread also has a beginning, a sequence, and an end. At any given time during the runtime of the thread, there is a single point of execution. However, a thread itself is not a program; a thread cannot run on its own. Rather, it runs within a program. The following figure shows this relationship.



Definition: A *thread* is a single sequential flow of control within a program. The real excitement surrounding threads is not about a single sequential thread. Rather, it's about the use of multiple threads running at the same time and performing different tasks in a single program. This use is illustrated in the next figure



A Web browser is an example of a multithreaded application. Within a typical browser, you can scroll a page while it's downloading an applet or an image, play animation and sound concurrently, print a page in the background while you download a new page, or watch three sorting algorithms race to the finish.

An application that creates an instance of Thread must provide the code that will run in that thread. There are two ways to do this:

1. Provide a Runnable object. The Runnable interface defines a single method, run, meant to contain the code executed in the thread. The Runnable object is

passed to the Thread constructor, as in the HelloRunnable example:

```

·public class HelloRunnable implements Runnable {

    public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[]) {
        (new Thread(new HelloRunnable())).start();
    }

}

```

2. Subclass Thread. The Thread class itself implements Runnable, though its run method does nothing. An application can subclass Thread, providing its own implementation of run, as in the HelloThread example:

```

public class HelloThread extends Thread {

    public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[]) {
        (new HelloThread()).start();
    }

}

```

(c) What is the use of super keyword in java? Explain.

5

Ans. Refer to Chapter-6, Q.No.-4, Page No.-82

(d) Why do we need static data in a class?

3

Ans. Refer to Chapter-5, Page No.-74

4. (a) Write an Applet in java to print the directory in which an HTML file containing that applet is stored.

5

```

Ans. import java.awt.*;
import java.applet.*;
import java.net.*;
/*
<applet code="Bases" width=300 height=50>
</applet>
*/
public class Bases extends Applet{
// Display code and document bases.

```

```

public void paint(Graphics g) {
String msg;
URL url = getCodeBase(); // get code base
msg = "Code base: " + url.toString();
g.drawString(msg, 10, 20);
url = getDocumentBase(); // get document base
msg = "Document base: " + url.toString();
g.drawString(msg, 10, 40);
}
}

```

(b) Explain the three different methods used to create an object of InetAddress class. 3

Ans. The **InetAddress** class has no visible constructors. To create an **InetAddress** object, you have to use one of the available factory methods. *Factory methods* are merely a convention whereby static methods in a class return an instance of that class. This is done in lieu of overloading a constructor with various parameter lists when having unique method names makes the results much clearer. Three commonly used **InetAddress** factory methods are shown here.

1. static InetAddress getLocalHost()
throws UnknownHostException
- 2.static InetAddress getByName(String *hostName*)
throws UnknownHostException
- 3.static InetAddress[] getAllByName(String *hostName*)
throws UnknownHostException

The **getLocalHost()** method simply returns the **InetAddress** object that represents the local host. The **getByName()** method returns an **InetAddress** for a host name passed to it. If these methods are unable to resolve the host name, they throw an **UnknownHostException**.

On the Internet, it is common for a single name to be used to represent several machines. In the world of web servers, this is one way to provide some degree of scaling. The **getAllByName()** factory method returns an array of **InetAddress**s that represent all of the addresses that a particular name resolves to. It will also throw an **UnknownHostException** if it can't resolve the name to at least one address.

(c) What are the various colour constructors? 4

Ans. Java supports color in a portable, device-independent fashion. The AWT color system allows you to specify any color you want. It then finds the best match for that color, given the limits of the display hardware currently executing your program or applet.

Color is encapsulated by the **Color** class.

Color defines several constants (for example, **Color.black**) to specify a number of common colors. You can also create your own colors, using one of the color constructors. The most commonly used forms are shown here:

`Color(int red, int green, int blue)`

`Color(int rgvValue)`

`Color(float red, float green, float blue)`

The first constructor takes three integers that specify the color as a mix of red, green, and blue. These values must be between 0 and 255, as in this example:

```
new Color(255, 100, 100); // light red.
```

The second color constructor takes a single integer that contains the mix of red, green, and blue packed into an integer. The integer is organized with red in bits 16 to 23, green in bits 8 to 15, and blue in bits 0 to 7. Here is an example of this constructor:

```
int newRed = (0xff000000 | (0xc0 << 16) | (0x00 << 8) | 0x00);
```

```
Color darkRed = new Color(newRed);
```

The final constructor, **Color(float, float, float)**, takes three float values (between 0.0 and 1.0) that specify the relative mix of red, green, and blue.

(d) Write a program in java to copy one file to another file using command line arguments.

8

```
Ans. import java.io.*;
class copyfile
{
    FileReader fr ;
    BufferedReader br;
    FileWriter fw ;
    BufferedWriter bw;
    String s;

    public static void main(String args[])
    {
        copyfile obj=new copyfile();
        obj.copy(args[0],args[1]);
    }

    void copy(String a, String b)
    {
        try{
            fr= new FileReader(a);

```

```

br = new BufferedReader(fr);
fw= new FileWriter(b);
bw = new BufferedWriter(fw);
while((s = br.readLine()) != null)
{
    bw.write(s);
    bw.flush();
}
}//try
catch(Exception e)
{}
}//copy
}

```

5. (a) Write a program in java to accept an integer number from the user. An exception is thrown if number entered is < 10. 8

Ans.

```

import java.io.*;
class myException extends Exception
{
myException(String message)
{
super(message);
}}
class number
{
public static void main(String args[])
{
try{
DataInputStream in=new DataInputStream(System.in);
String a=in.readLine();

int i=Integer.parseInt(a);
if(i<10)
throw new myException("Number less than 10");
}
catch(Exception e)
{
System.out.println("caught "+e);
}
}
}

```

(b) What are the JAR and Manifest files in java?

4

Ans. JAR Files: A JAR file allows you to efficiently deploy a set of classes and their associated resources. For example, a developer may build a multimedia application that uses various sound and image files. A set of Beans can control how and when this information is presented. All of these pieces can be placed into one JAR file.

JAR technology makes it much easier to deliver and install software. Also, the elements in a JAR file are compressed, which makes downloading a JAR file much faster than separately downloading several uncompressed files. Digital signatures may also be associated with the individual elements in a JAR file. This allows a consumer to be sure that these elements were produced by a specific organization or individual.

The package `java.util.zip` contains classes that read and write JAR files.

Manifest Files: A developer must provide a *manifest file* to indicate which of the components in a JAR file Java Beans are. An example of a manifest file is provided in the following listing. It defines a JAR file that contains four `.gif` files and one `.class` file. The last entry is a Bean.

```
Name: sunw/demo/slides/slide0.gif
Name: sunw/demo/slides/slide1.gif
Name: sunw/demo/slides/slide2.gif
Name: sunw/demo/slides/slide3.gif
Name: sunw/demo/slides/Slides.class
```

Java-Bean: True

A manifest file may reference several `.class` files. If a `.class` file is a Java Bean, its entry must be immediately followed by the line “Java-Bean: True”.

(c) Static functions can only access static data. Explain.

4

Ans. When a number is declared static it can be accessed before any objects of its class are created and without reference to any object. We can declare both methods and variables to be static. The most common example of a static number is a main (). Main () is declared as static because it must be called before any objects exist.

```
class abc
{
static int i;
abc ()
{
i++;
}
}
class static_demo
{
public static void main (string args [])
{
abc obj = new abc ();
System.out.printer (obj. i);
abc1 obj2 = new abc1 ();
```

```

    system.out.printer (obj1. i);
}
}
}

```

Thus static functions can access static data.

(d) Differentiate between TCP client and TCP server socket with the help of an example.

4

Ans. TCP/IP sockets are used to implement reliable, bidirectional, persistent, point-to-point, stream-based connections between hosts on the Internet. A socket can be used to connect Java's I/O system to other programs that may reside either on the local machine or on any other machine on the Internet.

There are two kinds of TCP sockets in Java. One is for servers, and the other is for clients. The **ServerSocket** class is designed to be a "listener," which waits for clients to connect before doing anything. The **Socket** class is designed to connect to server sockets and initiate protocol exchanges.

The creation of a **Socket** object implicitly establishes a connection between the client and server. There are no methods or constructors that explicitly expose the details of establishing that connection. Here are two constructors used to create client sockets:

1. **Socket(String hostName, int port)** :Creates a socket connecting the local host to the named host and port; can throw an **UnknownHostException** or an **IOException**.

2. **Socket(InetAddress ipAddress, int port)** : Creates a socket using a preexisting **InetAddress** object and a port; can throw an **IOException**.

The **ServerSocket** class is used to create servers that listen for either local or remote client programs to connect to them on published ports. Since the Web is driving most of the activity on the Internet, this section develops an operational web (http) server.

ServerSockets are quite different from normal **Sockets**. When you create a **ServerSocket**, it will register itself with the system as having an interest in client connections. The constructors for **ServerSocket** reflect the port number that you wish to accept connections on and, optionally, how long you want the queue for said port to be. The queue length tells the system how many client connections it can leave pending before it should simply refuse connections. The default is 50. The constructors might throw an **IOException** under adverse conditions. Here are the constructors:

1. **ServerSocket(int port)** Creates server socket on the specified port with a queue length of 50.

2. **ServerSocket(int port, int maxQueue)** Creates a server socket on the specified port with a maximum queue length of *maxQueue*.

3. **ServerSocket(int port, int maxQueue, InetAddress localAddress)**: Creates a server socket on the specified port with a maximum queue length of *maxQueue*. On a multihomed host, *localAddress* specifies the IP address to which this socket binds.

ServerSocket has a method called **accept()**, which is a blocking call that will wait for a client to initiate communications, and then return with a normal **Socket** that is then used for communication with the client.

**MCS – 024: OBJECT ORIENTED TECHNOLOGIES AND JAVA
PROGRAMMING
December, 2007**

Note: Question number 1 is compulsory. Attempt any three questions from the rest.

Q1. (a) Explain the life cycle of applet, briefly describing all its methods.
Refer to Dec-2006, Q.No.-1(e)

(b) Explain the different application areas of OOPS.

Different application areas of OOPS are Engineering, CAD/CAM, Multimedia, Scientific Applications etc.

(c) What is a constructor? Write a java program to explain how super class constructors are called in their subclass.

Refer to Chapter-5, Q.No.-2

Now Refer to Chapter-6, Q.No.-4

(d) Compare and contrast multiple inheritance and multilevel inheritance with example.

Refer to Chapter-6, Q.No.-1

(e) Explain the difference between String and String Buffer Class with example.

Refer to Chapter-3, Q.No.-5

(f) Write the output of the following:

(i) protected class Student

```
{ int x = 1;  
static int y;  
public static void main (string arg[ ])  
{  
system.out.println ("The value of y is:" +y);  
}  
}
```

The value of y is: 0

**(ii) string S1 = “Hello”;
string S2 = “Hello”;**

```

string S3 = new string (“Hello”);
System.out.println (“S1 is equal to S2” + S1 equals (S2));
System.out.println (“S1 is equal to S3” + S1 == S2);
compile time error in line no. 3

```

(iii) int X = 5*2 + 10/2 – 3;
System.out.println (“The value of X is:” + X);
The value of X is: 12

(g) Write a program in java to find the length of string “Practice in programming is always”. Find the difference between first and last occurrence of “r” in this string.

Refer to Q.No.-14, Page-102

(h) What are the various data types supported by java? What are their value storage limits?

Refer to Q.No.-1, Page-28

Now Refer to Q.No.-13, Page-14

Q2. (a) Explain the use of final keyword with variable, method and class.

Refer to Q.No.-6, Page-85

(b) Explain the URL connection class. Explain any four methods of this class.

Refer to Q.No.-6, Page-130

(c) What are various access specifiers in java? Write their accessibility.

Refer to Q.No.-2, Page-171

(d) Find the errors in the following java program and correct them.

```

1. public class test_string
2. {
3. public static main(string)
4. {
5. string str = “Hi java”;
6. int x = str . length;
7. system.out.println (“length is” + x);
8. }
9. }

```

Ans. Line 3 must be : public static void main(String arg[]])

Line 5 must be : String str = “Hi java”;

Line 7 must be : system.out.println (“length is” + x);

Q3. (a) Write a program to create an applet which contains 3 buttons having labels “one”, “two” and “three” and one text field. When user clicks any button its label text appears in the test field.

Refer to similar question

(b) What is a bitwise operator? If i is int i = 28, what will be the value of i after

- (i) $i = <<< 2;$
- (ii) $i = i + i <<< 2;$

Refer to Page-33

(c) What are the differences between checked and unchecked exceptions?

Name two classes in each category.

Refer to Q.No.-9, Page-101

Now Refer to Q.No.-2, Page-91

Q4. (a) Write an exception subclass which throws an exception if the string variable passed as an argument to a method, doesn't have first letter in upper case.

Refer to similar question

(b) Compare and contrast “equals()” & “==” operator in context of string class with example.

Refer to Q.No.-2(a)(ii), Page-231

(c) What is meant by Stream and what are the types of Streams and Classes of the Streams?

Refer to Q.No.-10, Page-101

Now Refer to Q.No.-4(d), Page-255

(d) Differentiate between ‘Transient’ and ‘Volatile’ keyword.

Refer to Q.No.-2(a)(i), Page-230

Q5. (a) Why is Pushback Input Stream used?

Ans. A PushbackInputStream adds functionality to another input stream, namely the ability to “push back” or “unread” one byte. This is useful in situations where it is convenient for a fragment of code to read an indefinite number of data bytes that are delimited by a particular byte value; after reading the terminating byte, the code fragment can “unread” it, so that the next read

operation on the input stream will reread the byte that was pushed back. For example, bytes representing the characters constituting an identifier might be terminated by a byte representing an operator character; a method whose job is to read just an identifier can read until it sees the operator and then push the operator back to be re-read.

(b) Write a program which calculates the size of a given file and then renames that file to another name.

Refer to Q.No.-1, Page-147

(c) Explain the concept of ‘Stub’ and ‘Skeleton’ in RMI.

Refer to Q.No.-1, Page-137

(d) What are the differences between Get and Post methods?

Refer to Q.No.-2(a), Page-206

(e) What is the difference between a java bean and instance of a normal java class?

Refer to Q.No.-10, Page-146

**MCS – 024: OBJECT ORIENTED TECHNOLOGIES AND JAVA
PROGRAMMING**
June, 2008

Note: Question number 1 is compulsory. Attempt any three questions from the rest.

Q1. (a) What is bytecode? Explain the difference between compiled code of Java and compiled code of C.

Refer to Q.No.-4, Page-18

Now Refer to Q.No.-2 and 3, Page-17-18

(b) Write a class complex that represents a complex number with suitable constructors and a function that multiplies two complex numbers.

Refer to Q.No.-2(a), Page-244

(c) What is the difference between method overloading and method overriding? Explain with suitable examples.

Refer to Chapter-5, Q.No.-3

(d) What is classpath? Explain its use with suitable example.

Ans. The Classpath is an argument set on the command-line, or through an environment variable, that tells the Java Virtual Machine where to look for user-defined classes and packages in Java programs.

Use : Suppose we have a package structure called org.mypackage containing the following classes : HelloWorld (main class), SupportClass, and UtilClass, the package being physically under the directory D:\myprogram (on Windows).

The corresponding physical file structure is :

D:\myprogram\

|

—> org\

|

—> mypackage\

|

—> HelloWorld.class

—> SupportClass.class

—> UtilClass.class

To launch the program, we should use the following command :

java -classpath D:\myprogram org.mypackage.HelloWorld

where :

- classpath D:\myprogram set the path to the packages used in the program

org.mypackage.HelloWorld is the path of the main class

Setting the path through an environment variable

The Environment variable named CLASSPATH may be alternatively used to set the Classpath. For the above example, we could also use on Windows :

```
set CLASSPATH=D:\myprogram
```

```
java org.mypackage.HelloWorld
```

(e) What is the need of interprocess communication? How does Java support it? Explain with example.

Ans. Unlike most other computer languages, JAVA provides built-in support for multithreading programming. A multithreading program contains two or more parts that can run concurrently. Each point of such program is called thread; specialized form of multitasking.

You are most certainly acquainted with multitasking, because it is supported by virtually all modern Operating Systems. However, there are two distinct types of multitasking; process-based & thread-based. It is important to understand the difference between two. For most readers, process-based multitasking is the more familiar form. A process is, in essence, a program that is executing. Thus process-based multitasking is the feature that allows your computer to run two or more programs concurrently. For example, process-based multitasking enables you to run the Java compiler at the time your text editor is also running. In process-based multitasking, a program is the smallest unit of code that can be dispatched by the scheduler.

In a thread-based multitasking environment, the thread is the smallest unit of dispatchable code. This means that a single program can perform two or more tasks simultaneously. For instance a text editor can format text at the same time that it is printing, as long as these two actions are being performed by two separate threads. Thus, process-based multitasking deals with the “big-picture”, and thread-based multitasking handles the details.

Multitasking threads require less overhead than multitasking process. Processes are heavy weight task that require their own separate address spaces. Inter process communication is expensive and limited. Context switching from one process to another is also costly. Threads, on the other hand, are lightweight. They share the same address space and cooperatively share the same heavyweight process. Inter thread communication is less expensive and context switching from one thread to the next is at low cost. While Java programs make use of process-based multitasking environments, process-based multitasking is not under the control of Java. However, multithreaded multitasking is.

Multithreading enables you to write very efficient programs that make maximum use of the CPU, because idle time can be kept to minimum, this is especially

important for the interactive, networked environment in which Java operates, because idle time is common. For example, the transmission rate of data over a network is much slower than the rate at which the CPU can process it. If you have programmed for OS such as Windows 98 or Window NT, then you are already familiar with multithreaded programming. However, the fact that Java threads manager makes multithreading especially convenient, because many of the details are handled for you.

Now Refer to Q.No.-6, Page-76

Now Refer to Q.No.-4(c), Page-212

(f) Write a program in Java to convert the following string in lower case: “IGNOU”.

Ans. Following is the program which can convert given string in lowercase

```
public class TestProg
{
    public static void main(String args[])
    {
        String s = "IGNOU";
        String slower = s.toLowerCase();
        System.out.println(slower); // prints out: ignou
    }
}
```

Q2. (a) What is an abstract class? How can it be used to implement polymorphism in Java? Explain your answer with an example.

Ans. Abstract classes are classes that contain one or more abstract methods. An abstract method is a method that is declared, but contains no implementation. Abstract classes may not be instantiated, and require subclasses to provide implementations for the abstract methods. Let's look at an example of an abstract class, and an abstract method.

Example of implementation of polymorphism : Multiple polymorphism is when an abstract class uses another abstract class. An OutputDevice abstraction has been introduced for traceable objects to write to. With this design, any traceable object can write to any OutputDevice.

```
public interface OutputDevice
{
    public void Write( String out );
}

public class SystemOutput implements OutputDevice
{
    public void Write( String out )
```

```

    {
        System.out.println( out );
    }
}

public class LogFileOutput implements OutputDevice {/* impl */ }

public interface Traceable
{
    public void Trace( OutputDevice device);
}

public class Foo implements Traceable
{
    public void toString() /* return state of object*/;
    public void Trace( OutputDevice device)
    {
        device.Write( toString );
    }
}

public class TraceManager
{
    public void Register( Traceable item ) { }
    public void TraceAll( OutputDevice device)
    {
        /* iterate all Traceable
         objects and invoke Trace( device )*/
    }
}

```

Using multiple polymorphism leads to a more scalable and extensible design. As new OutputDevice classes are identified and implemented, the TraceManager class remains unaltered — as do the classes that implement Traceable.

(b) What is an interface in Java? What is its role in inheritance? Explain the difference between interface and abstract class.

Refer to Q.No.-3, Page-10

Now Refer to Q.No.-28, Page-193

(c) What is the finally clause? Explain its use in Java with example.

Refer to Q.No.-5, Page-95

Q3. (a) What is multithreading? What is its need in Java? How would

you add threads to a class that already inherits from a class other than the thread class? (Explain with examples)

Refer to Q.No.-4(c), Page-212

(b) What is StreamTokenizer? What are different instance variables defined in StreamTokenizer? Give an example showing the use of StreamTokenizer.

Ans. The StreamTokenizer class takes an input stream and parses it into “tokens”, allowing the tokens to be read one at a time. The parsing process is controlled by a table and a number of flags that can be set to various states. The stream tokenizer can recognize identifiers, numbers, quoted strings, and various comment styles.

Each byte read from the input stream is regarded as a character in the range ‘\u0000’ through ‘\u00FF’. The character value is used to look up five possible attributes of the character: *white space*, *alphabetic*, *numeric*, *string quote*, and *comment character*. Each character can have zero or more of these attributes. In addition, an instance has four flags. These flags indicate:

- Whether line terminators are to be returned as tokens or treated as white space that merely separates tokens.
- Whether C-style comments are to be recognized and skipped.
- Whether C++-style comments are to be recognized and skipped.
- Whether the characters of identifiers are converted to lowercase.

A typical application first constructs an instance of this class, sets up the syntax tables, and then repeatedly loops calling the nextToken method in each iteration of the loop until it returns the value TT_EOF.

Field Summary	
double	nval If the current token is a number, this field contains the value of that number.
String	sval If the current token is a word token, this field contains a string giving the characters of the word token.
static int	TT_EOF A constant indicating that the end of the stream has been read.
static int	TT_EOL A constant indicating that the end of the line has been read.
static int	TT_NUMBER A constant indicating that a number token has been read.
static int	TT_WORD A constant indicating that a word token has been read.
int	ttype After a call to the nextToken method, this field contains the type of the token just read.

```
/* This class reads data from a file and stores it in an array of classes. It uses
a StreamTokenizer to divide the data into separate tokens. */
class WeeksWeather
{
private WeatherData [] temperatures;
private int numberTemps;
final int maxSize = 10;
WeeksWeather ()
{
numberTemps = 0;
temperatures = new WeatherData [maxSize];
} // constructor
public void readTemperatures ()
{
int next;
WeatherData data;
BufferedReader stdin = new BufferedReader (new InputStreamReader
(System.in));
try
{
System.out.print ("File Name: ");
String fileName = stdin.readLine ();
BufferedReader infile = new BufferedReader (new InputStreamReader (new
FileInputStream (fileName)));
StreamTokenizer tokens = new StreamTokenizer (infile);
next = tokens.nextToken ();
while (next != tokens.TT_EOF)
{
data = new WeatherData ();
data.readData (tokens);
data.displayTemps ();
temperatures [numberTemps] = data;
numberTemps++;
next = tokens.nextToken ();
}
} // try block
catch (IOException e) { System.out.println ("IO Exception");}
catch (NumberFormatException ex) { System.out.println ("Number format
error.");}
} // method readTemperatures
private int calculateAverageHigh ()
{
int sum = 0;
```

```
for (int count = 0; count < numberTemps; count++)
    sum = sum + temperatures [count].getHigh ();
    int average = sum / numberTemps;
    return average;
} // method calculateAverage
private int findHighest ()
{
    int highest;
    highest = temperatures [0].getHigh ();
    for (int count = 1; count < numberTemps; count++)
    {
        if (highest < temperatures [count].getHigh ())
            highest = temperatures [count].getHigh ();
    }
    return highest;
} // method findHighest
private int findLowest ()
{
    int lowest;
    lowest = temperatures [0].getLow ();
    for (int count = 1; count < numberTemps; count++)
    {
        if (lowest > temperatures [count].getLow ())
            lowest = temperatures [count].getLow ();
    }
    return lowest;
} // method findLowest
public void displayStatistics ()
{
    int average = calculateAverageHigh ();
    int highest = findHighest ();
    int lowest = findLowest ();
    System.out.println ("The average high was " + average);
    System.out.println ("The highest temperature was " + highest);
    System.out.println ("The lowest temperature was " + lowest);
} // method displayAverage
} // class WeeksWeather
/* Program output:
Sunday High: 71 Low: 62
Monday High: 59 Low: 45
Tuesday High: 66 Low: 49
Wednesday High: 58 Low: 42
Thursday High: 52 Low: 44
```

Friday High: 63 Low: 52

Saturday High: 73 Low: 59

The average high was 63

The highest temperature was 73

The lowest temperature was 42

*/

(c) Explain the life cycle of an applet in brief.

Refer to Dec-2006, Q.No.-1(e), Page-240

Q4. (a) Write a program in Java that reads text from the keyboard and writes it to a .txt file.

Refer to Q.No.-4, Page-149

(b) What do you mean by the statement, “HTTP is a stateless protocol”?

**What are various ways through which you can maintain the state?
(Explain each in brief)**

Ans. HTTP is a stateless protocol. The advantage of a stateless protocol is that hosts do not need to retain information about users between requests, but this forces web developers to use alternative methods for maintaining users' states. For example, when a host would like to customize content for a user while visiting a website, the web application must be written to track the user's progress from page to page. A common method for solving this problem involves sending and requesting cookies. Other methods include server side sessions, hidden variables (when current page is a form), and URL encoded parameters (such as /index.php?session_id=some_unique_session_code).

(c) What is the use of finalize() method?

Refer to Page-76

(d) What is PushbackInputStream?

Refer to Dec-2007, Q.No.-5(a)

Q5. Explain the following with suitable example :

(a) RMI

Refer to Page-137

(b) Swing

Refer to Q.No.-8, Page-126

(c) Datagrams

Refer to Page-136

(d) Servlet

Refer to Q.No.-2, Page-140

Note: Question number 1 is **compulsory**. Attempt any **three** questions from the rest.

Q1. (a) What is the difference between Polymorphism and Encapsulation?

Refer to CS-74, Page No.-60

“Encapsulation is used as a generic term for techniques, which realize data abstraction. Encapsulation therefore implies the provision of mechanisms to support both modularity and information hiding.”

The object diagrams show that the object's variables make up the center or nucleus of the object.

(b) Write a program that displays a diamond using asterisks (*) as follows:



Ans. class Demo

```
{  
public static void main(String args[ ]) {  
{  
int i, j, k, sp1 = 6, s1, f = 0;  
for(i = 1; i <= 6; i++) {  
{  
for(s1 = 1; s1 <= sp1; s1++) {  
System.out.print(" ");  
System.out.print("*");  
for(j = 1; j < 2 * i - 1; j++) {  
{  
System.out.print(" ");  
f = 1;  
}  
if(f == 1) {  
System.out.print("*");  
System.out.print("\n");  
sp1 -= 1;  
}  
for(i = 1; i <= 6; i++) {  
{
```

```

for(s1 = 1; s1 <= i; s1++)
System.out.print(" ");
System.out.print("*");
for(j = 1; j <= 6 - (2 * i - 6); j++)
{
System.out.print(" ");
}
System.out.print("*");
System.out.print("\n");
}
System.out.print("*");
}
}

```

(c) Name three threads that are created automatically by the Java virtual machine and discuss the purpose of each thread.

Refer to Page No.-250, Q.No.-4(a)

(d) Explain the following and their usage using suitable examples:

(i) Extends

Refer to Page No.-79

(ii) Implements

Ans. When a class provides the methods (possibly abstract) necessary to conform to some interface we say that class implements the interface. Implementing an interface is closely related to extending a class. A new class can extend at most one superclass, but it may implement several interfaces. When a class or abstract class implements or extends, those methods are available in all descendants. All descendant classes automatically implement that interface too. You can refer to descendant class objects by the base class name or the interface name.

(iii) Final

Refer to Page No.-85

(e) Find the errors in the following Java programs and correct them:

(i) int g();

```

{
system.out.println ("Inside method g");
int h( );
{
system.out.println ("Inside method h");
}
}

```

Ans. Error in 3rd line

System.out.println("Inside method g");

Error in 6th line

System.out.println("Inside method h");
 In 1st line method g should return a value.
 In 3rd and 6th line s should be capital of system.
 In 4th line method h should return a value.

(ii) int sum (int x, int y)

```
{  
int result;  
result = x+y;  
}
```

Ans. Error in 1st line should be

void sum (int x, int y)

After 4th line there should be 1 line more i.e.

System.out.println(result);

void sum (int x, int y)

```
{  
int result;  
result = x+y;
```

System.out.println(result);

}

(f) What is a package in Java and what are the access criteria? Explain with suitable examples.

Refer to Page No.-105 and Page No.-107

(g) What is explicit casting? Explain with suitable example.

Ans. An explicit cast is a cast that you must specifically invoke, with either the CAST AS keywords or with the cast operator (::). The database server does *not* automatically invoke an explicit cast to resolve data type conversions. The EXPLICIT keyword is optional; by default, the CREATE CAST statement creates an explicit cast.

The following CREATE CAST statement defines an explicit cast from the **rate_of_return** opaque data type to the **percent** distinct data type:

CREATE EXPLICIT CAST (rate_of_return AS percent
 WITH rate_to_prct)

The following SELECT statement explicitly invokes this explicit cast in its WHERE clause to compare the **bond_rate** column (of type **rate_of_return**) to the **initial(APR)** column (of type **percent**):

SELECT bond_rate FROM bond
 WHERE bond_rate::percent > initial_APY

(h) What is the difference between declaring a variable and defining a variable? Explain with the help of example.

Refer to Chapter-3, Q.No.-7, Page No.-36

Q2. (a) What is the difference between the boolean operator & and the && operator? Explain with suitable example.

Ans.

Operator	Use	Description
&	A & B	Boolean AND: If both A and B are true, the result is true. If either A or B are false, the result is false and both A and B are evaluated before the test.
&&	A && B	Conditional AND: If both A and B are true, result is true. If either A or B are false, the result is false. But if A is false, B will not be evaluated. For example, (A > 4 && B <= 10) will evaluate to true if A is greater than 4 and B is less than or equal to 10.

(b) Find the errors in the following Java program and correct them:

Class Examination;

```
{
public void main (args[ ])
{
system.out.println ("java is an OOL")
int i = 10;
for (i > 0; i —)
system.out.println ("i = " i);
}
}
}
```

Ans. Line 3: Public static void main (args[])

Line 5: System.out.println("java is an OOL");

Line 7: for(i = 0; i < 10; i++)

Line 8: System.out.println("i = " + i);

Line 11: No braces expected

Class examination

```
{
Public static void main (String args[ ])
{
int i = 10;
for(i = 0; i < 10; i++)
System.out.println ("Java is an OOL");
}
}
```

(c) Write a program to implement Quick Sort technique.

Ans. QuickSort Recursive (a, lb, ub)

Here is an array of size n(>1) in memory. This algorithm sorts this array in ascending order using quick sort technique. The variable loc is used to hold the index of splitting element.

begin

```
    if (lb < ub) then
        Call Split Array(a, lb, ub, loc)
        Call QuickSort Recursive (a, lb, loc – 1)
        Call QuickSort Recursive (a, loc + 1, ub)
```

endif

End.

SplitArray(a, beg, end, loc)

Here, is an array in memory, and beg and end represents the lower bound and upper bound of the sub array in question.

begin

```
    Set left = be, right = end, loc = beg
    Set done = false
    while (not done) do
        while(a[loc] £ a[right]) and [loc ^ right)) do
            Set right = right – 1
```

endwhile

if (loc = right) then

 Set done = true

else if (a[loc]>a[right]) then

 Interchange a[loc] and a[right]

 Set loc = right

endif

if (not done) then

 while (a[loc] ^ a[left]) and (loc ^ left) do

 Set left = left + 1

 endwhile

 if(loc = left) then

 Set done = true

else if (a[loc] < a[left]) then

 Interchange a[loc] and a[left]

 Set loc = left

 endif

endif

endwhile

End.

(d) Find and correct the error in each of the following segments:

(i) final int A_size = 5;

A_size = 10;

Ans. Value size can't be change after giving the size with final keyword.

(ii) Assume

int a[] [] = {{1, 2}, {3, 4}},

a[1, 1] = 5;

Ans. a[1, 1] = 5; displaying the wrong value. Value 5 is not initialized.

Q3. (a) Write a Java applet to show a sine wave with variable frequency on a web page.

```
Ans. import Java.awt.*;
import Java.applet.*;
public class Demo extends Applet
{
    public void point (Graphics g)
    {
        g.setColor(Color.RED);
        g.drawArc(10, 20, 30, 100, 10, 100);
        g.setColor(Color.GREEN);
        g.drawArc(10, 20, 30, 100, 50, 100);
    }
}
```

(b) What do you mean by "this" keyword? Explain with example.

Refer to Page No.-81, Q.No.-3

(c) What is an event? Explain different components of an event.

Refer to Page No.-204, Q.No.-1(4)

(d) What gives Java it's "write once and run anywhere" nature?

Ans. Java is platform independence it gives "write once and run anywhere" nature. This is provided by the two concepts (i) Byte code and (ii) Java virtual machine. The source code of java is compiled into byte code and the byte code will be interpreted and executed by the JVM on the target machine.

Q4. (a)What is multithreading? Explain two advantages of multithreaded programs. Write a program in Java to explain how different priorities can be assigned to different threads.

Refer to Page No.-212, Q.No.-4(c)

(b) What is a servlet? Explain the use of GET and POST methods.

Refer to Page No.-206, Q.No.-2(a)

(c) What is the use of 'synchronize' keyword?

Ans. — The synchronized keyword may be applied to statement block or to a

method.

- The synchronized keyword provides the protection for the crucial sections that are required only to be executed by a single thread once at a time.
- The synchronized keyword avoids a critical code from being executed by more than one thread at a time. Its restricts other threads to concurrently access a resource. Example:

```
public class Counter {
```

```
    private int count = 0;
```

```
    public synchronized void count() {
        int limit = count + 100;
        while (count++ != limit) System.out.println(count);
    }
```

```
}
```

(d) What is a datagram socket? Explain the use of datagram socket through an example of a program written in Java.

Refer to Page No.-129, Q.No.-4(ii)

A datagram socket is the sending or receiving point for a packet delivery service. Each packet sent or received on a datagram socket is individually addressed and routed. Multiple packets sent from one machine to another may be routed differently, and may arrive in any order.

UDP broadcasts sends are always enabled on a DatagramSocket. In order to receive broadcast packets a DatagramSocket should be bound to the wildcard address. In some implementations, broadcast packets may also be received when a DatagramSocket is bound to a more specific address.

Q5. (a) Compare and contrast break and continue statement.

Refer to Chapter-4, Page No.-49-50

(b) What happens when a return type, even void is specified for a constructor? What do you understand by parameterized constructors? Explain with one example.

Ans. When a return type is specified for a constructor then there will be a compile time (syntax error) shown. Parameterized constructor is a constructor which is having arguments. By using this type of constructor we can specify the specific values for the some of members or all the data members.

Ex. class box

```
{
```

```
Double width, height ,depth;
box(double w, double h, double d)
```

```

{
Width = w;
Height = h;
Depth = d;
}
}

Class box demo{
Public = static void main(String args[])
{
Box mybox1= new box(10,20,15);
Box mybox2 = new box(3,6,9)
}
}

```

(c) Find the errors in the following Java program and correct them:

(i) `i = 1;`
`while (i <= 10);`
`i++; }`

Ans. 1. Declaration of data type because variable i can't be declare.

2. Looping statement can't be terminated by semicolon.

3. While loop should be open by curly braces.

(ii) `for (k = 0.1; k! = 1.0; k += 0.1)`

`System.out.println (k);`

Ans. Refer to www.gullybaba.com/ignou/mcs24.htm

(d) Write a program that calculates the product of the odd integers from 1 to 15.

Ans. Class Prod

```

{
void cal ( )
{
int i = 1;
int num;
int Prod = 1;
for (i = 1; i <= 15; i++)
{
num = i;
if(num % 2! = 0)
Prod = Prod * num;
}
System.out.println("The product of odd nos. is = " + Prod);
}
```

Note: Q. No. 1 is **compulsory**. Attempt **any three** questions from the rest.

Q1. (a) What are the major characteristics of the following:

(i) procedure oriented approach

(ii) object oriented approach

Ans. The **procedure oriented programming (POP)** approach focuses on creating and ordering procedures or a block of code keeping in mind to accomplish a specific job. The key features of this kind of approach are: use of procedures, sequencing of procedures and sharing global data. **The use of a procedure-oriented language for process-oriented simulation:**

Now Refer to Page No.-5, 2nd Para

(b) Compare and contrast abstraction and encapsulation techniques?

Also, give an example for comparison.

Refer to Page No.-12, Q.No.-11

(c) Write steps to create a java applet displaying “Global Warming”.

Ans. import.java.awt.*;

import.java.awt.*;

import.java.net.*;

/*

<applet code=”msg” width=300 height=50>

</applet>

*/

public class MSG extends Applet{

public void show(Graphics g){

g.drawString(“Global Warming”,10,20);

}

}

(d) Write the description, use and operations used for “Right Shift” and “Left Shift”.

Refer to Page No.-34, Last Para

(e) Differentiate between “Final”, “Finally” and “Finalize” statements.

Refer to Page No.-85, 95 and 76

(f) What is RMI? How can you create stub and skeleton? Write the steps involved.

Refer to Chapter-11, Q.No.-1, Page No.-137

Q2. (a) Describe any five constructors used to create string objects and any five methods used for handling string objects.

Ans. Five string constructors are:

public String(): Used to create a String object that represents an empty character sequence.

public String(String value): Used to create a String object that represents the same sequence of characters as the argument; in other words, the newly created string is a copy of the string passed as an argument.

public String(char value[]): New object of string is created using the sequence of characters currently contained in the character array argument.

public String(char value[], int offset, int count): A new string object created contains characters from a sub-array of the character array argument.

public String(byte, bytes[], String enc) throws Unsupported Encoding Exception: Used to construct a new String by converting the specified array of bytes using the specified character encoding.

Five methods for handling string object are:

Methods	Return
<code>public int length()</code>	Return the length of string.
<code>public int indexOf(int ch)</code>	Return location of first occurrence of ch in string, if ch don't exist in string return -1.
<code>public int indexOf(int ch, int fromIndex)</code>	Return location of occurrence of ch in string after fromIndex, if ch don't exist in string return -1.
<code>public int lastIndexOf(int ch)</code>	Return location of last occurrence of ch in string, if ch location does not exist in string return -1.
<code>public int lastIndexOf(int ch, int fromIndex)</code>	Return last of occurrence of ch in string after location fromIndex, if ch does not exist in string after location fromIndex return -1.

(b) Describe the “java io” hierarchy and classify the input stream class.
Refer to Page No.-255 to 258, Q.No.-4(d)

(c) Write the difference between throw and throws.

Refer to Chapter-7, Q.No-4, Page No.-93

Q3. (a) What is listener? Write a program to implement mouse motion listener.

Ans. Refer to Page No.-244, 4th Para and;

```
import Java.Applet.*;
import Jawa.awt.*;
import Java.awt.event.*;
public class TestMouse1
{
    public static void main (String[ ] args)
    {
        Frame f = new Frame("TestMouseListener");
        f.setSize(500,500);
        f.setVisible(true);
        f.addMouseListener(new MouseAdapter( ))
        {
            public void mouseClicked(MouseEvent e)
            {
                System.out.println("Mouse clicked:( "+e.getX( )+", "+e.getY( )+" )");
            }
        });
    }
}
```

(b) Write a java program to show inter thread communication.

Ans. //program

```
class WaitNotifyTest implements Runnable
{
    WaitNotifyTest( )
    {
        Thread th = new Thread (this);
        th.start( );
    }
    synchronized void notifyThat ( )
    {
        System.out.println("Notify the threads waiting");
        this.notify( );
    }
    synchronized public void run( )
    {
        try
        {
```

```

System.out.println("Thread is waiting...");  

this.wait( );  

}  

catch (InterruptedException e) { }  

System.out.println("Waiting thread notified");  

}
}
}

class runWeightNotify
{
public static void main (String args[ ])  

{
WaitNotifyTest wait_not = new WaitNotifyTest( );  

Thread.yield ( );  

wait_not.notifyThat( );  

}
}
}

Output:
```

Thread is waiting...
Notify the threads waiting
Waiting thread notified

Q4. (a) Discuss Layout and Layout Manager along five classes in java packages which implement it.

Ans. When we add a component to an applet or a container, the container uses its *layout manager* to decide where to put the component.

java.awt.LayoutManager is an interface. Five classes in the java packages implement it:

- FlowLayout
- BorderLayout
- CardLayout
- GridLayout
- GridBagLayout

Now Refer to Page No.-185 and 207

Card Layout: A CardLayout breaks the applet into a deck of cards, each of which has its own Layout Manager. Only one card appears on the screen at a time. The user flips between cards, each of which shows a different set of components. The common analogy is with HyperCard on the Mac and Tool book on Windows. In Java this might be used for a series of data input screens, where more input is needed than will comfortably fit on a single screen.

Grid Layout: A GridLayout divides an applet into a specified number of rows and columns, which form a grid of cells, *each equally sized and spaced*.

We will find the GridLayout is great for arranging Panels. A GridLayout specifies the number of rows and columns into which components will be placed. The applet is broken up into a table of equal sized cells.

GridLayout is useful when we want to place a number of similarly sized objects. It is great for putting together lists of checkboxes and radio buttons as we did in the Ingredients applet.

Grid Bag Layout: GridBagLayout is the most precise of the five AWT Layout Managers. It is similar to the GridLayout, but components do not need to be of the same size. Each component can occupy one or more cells of the layout. Furthermore, components are not necessarily placed in the cells beginning at the upper left-hand corner and moving to the right and down.

In simple applets with just a few components we often need only one layout manager. In more complicated applets, however, we will often split our applet into panels, lay out the panels according to a layout manager and give each panel its own layout manager that arranges the components inside it.

The GridBagLayout constructor is trivial, GridBagLayout() with no arguments.

```
GridBagLayout gbl = new GridBagLayout();
```

(b) Explain the significance of CONTAINER for Java GUI program.

Refer to Page No.-115, 2nd Para

(c) Write a program to show how a class implements two interfaces.

Ans. //program to implement two interfaces in a single class

interface First

```
{  
void MethodOne( );  
int MyValue1 = 100;  
}
```

interface Second

```
{  
public void MethodTwo( );  
int MyValue2 = 200;  
}
```

class interfaceIn implements First,Second

```
{  
public void MethodOne( )  
{
```

```
System.out.println("Method of interface First is implemented with  
value:"+MyValue1);
```

```
}
```

```
public void MethodTwo( )
```

```
{
System.out.println("Method of interface Second is implemented with
value:"+MyValue2);
}
}
class TwoInterfTest
{
public static void main(String args[ ])
{
interfaceIn object = new interfaceIn( );
object.MethodOne( );
object.MethodTwo( );
}
}
```

Output:

Method of interface First is implemented with value:100
 Method of interface Second is implemented with value:200

Q5. (a) Write programs to send and receive UDP datagrams. (Make separate sender and receiver program)

Ans. Sending UDP Datagrams: To send data to a particular server, you first must convert the data into byte array. Next you pass this byte array, the length of the data in the array (most of the time this will be the length of the array) and the InetAddress and port to which you wish to send it into the DatagramPacket() constructor.

For example, first we create DatagramPacket object

```
try
{
InetAddress m = new InetAddress("http://mail.yahoo.com");
int chargen = 19;
String s = "My second UDP Packet";
byte[ ] b = s.getBytes( );
DatagramPacket dp = new DatagramPacket(b, b.length, m, chargen);
}
catch (UnknownHostException ex)
{
System.err.println(ex);
}
```

Now create a Datagram Socket object and pass the packet to its send() method as try

```
{
```

```

DatagramSocket sender = new DatagramSocket( );
sender.send(dp);
}
catch(IOException ex)
{
System.err.println(ex);
}

```

Receiving UDP Datagrams: To receive data sent to you, you construct a DatagramSocket object on the port on which you want to listen. Then you pass an empty DatagramPacket object to the DatagramSocket's receive() method.

public void receive(DatagramPacket dp) throws IOException

The calling thread blocks until the a datagram is received. Then dp is filled with the data from that datagram. You can then use getPort() and getAddress() to tell where the packet came from, getData() to retrieve the data, and getLength() to see how many bytes were in the data. If the received packet was too long for the buffer, then it's truncated to the length of the buffer. For example,

```

try
{
byte buffer = new byte[65536]; //maximum size of an IP packet
DatagramPacket incoming = new DatagramPacket(buffer, buffer.length);
DatagramSocket ds = new DatagramSocket(2134);
ds.receive(dp);
byte[ ] data = dp.getData( );
String s = new String(data, 0, data.getLength( ));
System.out.println("Port" + dp.getPort( ) + "on" + dp.getAddress( )
+ "sent this message:");
System.out.println(s);
}
catch(IOException ex)
{
System.err.println(ex);
}

```

(b) Explain the life cycle of a servlet along with program which creates a simple servlet generating HTML to write “Welcome to IGNOU”?

Ans. Refer to Page No.-141, Q.No.-4

```

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

```

```
public class Welcome to IGNOU extends HttpServlet
{
    public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0" +
"Transitional//EN">\n" +
"<HTML>\n" +
"<HEAD><TITLE>Welcome to IGNOU</TITLE></HEAD>\n" +
"<BODY>\n" +
"<H1>Welcome to IGNOU</H1>\n" +
"</BODY></HTML>");
    }
}
```

Note : Q No. 1 is compulsory. Attempt any three questions from the rest.

Q1. (a) How does java handle event? Write a program in Java to capture an event generated by keyboard.

Ans. In Java, events represent all activity that goes on between the user and the application. Java's Abstract Windowing Toolkit (AWT) communicates these actions to the programs using events. Here, this is done through the *java.awt.**; package of java. Events are the integral part of the java platform. For example: to handle events in *java.awt.**; package. AwtEvent is the main class of the program which extends from the **Frame** class implements the ActionListener interface.

This program starts to run from the main method in which the object for the AwtEvent class has been created. The constructor of the AwtEvent class creates two buttons with adding the addActionListener() method to it. The constructor also initializes a label with text “Gullybaba.com”.

When you click on the button then the actionPerformed() method is called which receives the generated event. This method shows the text of the source of the event on the label.

Here is the code of the program :

```
import java.awt.*;
import java.awt.event.*;
public class AwtEvent extends Frame implements ActionListener
Label lbl
public static void main(String argv[]){
    AwtEvent t = new AwtEvent();
}
public AwtEvent(){
    super("Event in Java awt");
    setLayout(new BorderLayout());
    try{
        Button button = new Button("INSERT_AN_URL_HERE");
        button.addActionListener(this);
        add(button, BorderLayout.NORTH);
        Button button1 = new Button("INSERT_A_FILENAME_HERE");
        button1.addActionListener(this);
        add(button1, BorderLayout.SOUTH);
        lbl = new Label("Gullybaba.com");
    }
}
```

```
add(lbl, BorderLayout.CENTER);
addWindowListener(new WindowAdapter(){
public void windowClosing(WindowEvent we){
    System.exit(0);
}
});
}
catch (Exception e){}
setSize(400,400);
setVisible(true);
}

public void actionPerformed(ActionEvent e){
Button bt = (Button)e.getSource();
String str = bt.getLabel();
lbl.setText(str);
}
}

// program to capture an event
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.KeyAdapter;
import org.eclipse.swt.events.KeyEvent;
import org.eclipse.swt.layout.FillLayout;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;

public class SWTKeyEvent
{
public static void main (String [] args) {
Display display = new Display ();
Shell shell = new Shell(display);
shell.setText("SWT KeyEvent Example");

shell.setLayout(new FillLayout());

Button button = new Button(shell, SWT.CENTER);

button.setText("Type Something");
button.addKeyListener(new KeyAdapter()
{
```

```

public void keyPressed(KeyEvent e)
{
String string = "";
//check click together?
if((e.stateMask & SWT.ALT) != 0) string += "ALT - keyCode = " + e.keyCode;
if((e.stateMask & SWT.CTRL) != 0) string += „CTRL - keyCode = „ + e.keyCode;
if((e.stateMask & SWT.SHIFT) != 0) string += "SHIFT - keyCode = " + e.keyCode;

if(e.keyCode == SWT.BS)
{ string += "BACKSPACE - keyCode = " + e.keyCode;
}

if(e.keyCode == SWT.ESC)
{ string += "ESCAPE - keyCode = " + e.keyCode;
}

//check characters
if(e.keyCode >=97 && e.keyCode <=122)
{ string += " " + e.character + " - keyCode = " + e.keyCode;
}

//check digit
if(e.keyCode >=48 && e.keyCode <=57)
{ string += " " + e.character + " - keyCode = " + e.keyCode;
}
if(!string.equals(""))
System.out.println (string);
} );
shell.open();
while (!shell.isDisposed ()) {
if (!display.readAndDispatch ()) display.sleep ();
display.dispose ();
}
}

```

(b) Write a program to generate a line in an applet?

Ans. The drawLine() method has been used in the program to draw the line in the applet. Here is the syntax for the drawLine() method :

drawLine(int X_from_coordinate, int Y_from_coordinate, int X_to_coordinate, int Y_to_coordinate);

Here is the java code of the program :

```
import java.applet.*;
import java.awt.*;

public class Line extends Applet{
    int x=300,y=100,r=50;

    public void paint(Graphics g)
    {
        g.drawLine(3,300,200,10);
        g.drawString("Line",100,100);
    }
}
```

Here is the HTML code of the program:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<div align="center">
<APPLET CODE="Line.class" WIDTH="800" HEIGHT="500"><
APPLET>
</div>
</BODY>
</HTML>
```

(c) Explain Java Thread Model along with priorities.

Ans. Java environment has been built around the multithreading model. In fact all Java class libraries have been designed keeping multithreading in mind. If a thread goes off to sleep for some time, the rest of the program does not get affected by this. Similarly, an animation loop can be fired that will not stop the working of rest of the system.

At a point of time a thread can be in any one of the following states – new, ready, running, inactive and finished. A thread enters the new state as soon as it is created. When it is started (by invoking start() method), it is ready to run. The start() method in turn calls the run() method which makes the thread enter the running state. While running, a thread might get blocked because some resource that it requires is not available, or it could be suspended on purpose for some reason (like put off to sleep by the programmer). In such a case the thread enters the state of being inactive. A thread can also be stopped purposely because its time has expired, then it enters the state of ready to run once again.

A thread that is in running state can be stopped once its job has finished. A thread that is ready to run can also be stopped. A thread that is stopped enters

the finished state. A thread that is in inactive state can either be resumed, in which case it enters the ready state again, or it can be stopped in which case it enters the finished state.

Thread Priorities

In multithreading environment, one thread might require the attention of the CPU more quickly than other. In such a case that thread is said to be of high priority. Priority of a thread determines the switching from one thread to another. In other words, priority determines how a thread should behave with respect to the other threads.

Thread priorities are integers that specify the relative priority of one thread to another.

The word priority should not be confused with the faster running of a thread. A high priority thread does not run any faster than the low priority thread. A thread can voluntarily leave the control by explicitly stopping, sleeping or blocking on pending I/O or it can pre-empted by the system to do so. In the first case the processor examines all threads and assigns the control to the thread having the highest priority. In the second case, a low priority thread that is not ready to leave the control is simply pre-empted by the higher priority thread no matter what it is doing. This is known as pre-emptive multi-tasking. It is advisable that in case two threads have the same priority, they must explicitly surrender the control to their peers.

(d) Write any four methods involved in output stream class.

Ans. Four methods in output stream class are :

`void close()` : Closes this output stream and releases any system resources associated with this stream.

`void flush()` : Flushes this output stream and forces any buffered output bytes to be written out.

`void write(byte[] b)` : Writes `b.length` bytes from the specified byte array to this output stream.

`void write(byte[] b, int off, int len)` : Writes `len` bytes from the specified byte array starting at offset `off` to this output stream.

(e) Discuss the throwable class hierarchy for handling exceptions in Java

Refer Chapter-7, Q 2, Page No- 91

(f) Differentiate between superclass and subclass. Also write a program to show the reusability concept.

Ans. Difference between super class and subclass

The terms Super Class and Sub Class comes from the OOPS property named INHERITANCE.

- 1) A subclass inherits all the properties of a Super class.
- 2) A subclass can access all the methods, variables of super class and not vice versa.
- 3) A subclass extends a super class.
- 4) A super class is a class that is inherited whereas subclass is a class that does the inheriting.

IF you want to reuse the code of one class then your new class should “extend” the Older One. So that you will have all the code present in the Older class in the new One.

The Older class is called the Parent/SuperClass

The newer one is called the Child/SubClass.

Program to show the reusability concept using method overriding:

Class A

```
{  
int a,b;  
A(int i,int j)  
{  
a=i;  
b=j;  
}  
Void show()  
{  
System.out.println("a and b" +a +" "+j);  
}  
}
```

Class B extends A

```
{  
Int k;  
B(int x,int y,int z)  
{  
Super(x,y);  
K=c;  
}  
Void show()  
{  
System.out.println("k" +k)  
}  
}
```

Class reuse

```
{
```

```

Public static void main(String args[])
{
B sub=new B(1,2,3);
Sub.show();
}
    }

```

Q2. (a) Explain the differences between Swing and AWT. Also, write the advantages of swing based GUI.

Refer Chapter-9, Q.No.-9, Page No- 126

(b) Write a program describing the usage of multidimensional array.

Refer Chapter-3, Q.No.-12, Page No.-42 and Q.No.-15, Page No.-44

(c) Explain the usage of iterative statements (for, do-while) using programs.

Refer Chapter-4, Page No- 47-49

Q3. (a) Why the public and static keywords are used in “public static void main()”? Explain.

Refer to Chapter-2, Q.No.-7, Page No.-23

(b) What are the benefits of OOPS?

Ans. Oops offers several benefits to both the program developer and the user. The major benefits are:

- **Reduce complexity:** software complexity can be easily managed.
- **Provide extensibility:** object oriented system can be easily upgraded from small to large system.
- **Eliminate redundancy:** through inheritance we can eliminate redundant code and extend the use of existing classes.
- **Allow building secure program:** data hiding principle helps programmer to build secure program that can be accessed by code in other parts of the program.
- **Increase productivity:** instead of writing code from scratch, solution can be built by using standard working modules. So saves the development time.

(c) Explain Applet life cycle along with methods used with it. What are those components of an event used with AWT.

Refer to Dec-2006, Q.No.-1(e), Page No.-240 and; Refer to Page No.-245

Q4. (a) Explain the usage of Abstract classes.

Refer Chapter-6, Q 6, Page No- 85

(b) What is the relation in interface and inheritance? Also, explain how multiple inheritance can be implemented using interface.

Ans. One interface can inherit another interface using extends keyword. This is syntactically the same as inheriting classes. Let us say one interface A is inheriting interface B. At the same time a class C is implementing interface A. Then class C will implement methods of both the interface A and B. E.g.
//program to show relation between interface and inheritance

Interface A

```
{  
Public Void MYMethod_A();  
}
```

Interface B extends A

```
{  
Public Void MYMethod_B();  
}
```

Class Interface_Inheritance implements B

```
{  
Public void MYMethod_A()  
{  
}
```

```
System.output.println("Method of Interface A is implemented")  
}
```

```
Public void MYMethod_B()  
{  
System.output.println("Method of Interface B is implemented")  
}
```

```
}
```

Class Test

```
{  
Public static void main(String args[])  
{  
}
```

```
Interface_Inheritance ob= new Interface_Inheritance();
```

```
Ob. MYMethod_A();  
Ob. MYMethod_B();  
}  
}
```

Java does not support Multiple inheritance however multiple inheritance can be implemented using interface. A class which is inherited more than one class is called multiple inheritance. So in Interface that classes within classes mechanism is there. This mechanism is behind the keyword Interface. The keyword Interface means it is classes within classes. So by implicitly we are using multiple inheritance by the use of interface..

The multiple inheritance is supported by using more implements keyword example :

```
class a implements b, implements c, implements d
{
//stmt
}
```

(c) Write a Java program to copy the text contents of one file into another file.

Ans. Java program to copy the text contents of one file into another file:

```
import java.io.*;
```

```
public class CopyFile{
    private static void copyfile(String srFile, String dtFile){
        try{
            File f1 = new File(srFile);
            File f2 = new File(dtFile);
            InputStream in = new FileInputStream(f1);

            //to Append the file.
            OutputStream out = new FileOutputStream(f2,true);

            //to Overwrite the file.
            OutputStream out = new FileOutputStream(f2);

            byte[] buf = new byte[1024];
            int len;
            while ((len = in.read(buf)) > 0){
                out.write(buf, 0, len);
            }
            in.close();
            out.close();
            System.out.println("File copied.");
        }
        catch(FileNotFoundException ex){
            System.out.println(ex.getMessage() + " in the specified directory.");
            System.exit(0);
        }
        catch(IOException e){
            System.out.println(e.getMessage());
        }
    }
}
```

```

}
public static void main(String[] args){
switch(args.length){
case 0: System.out.println("File has not mentioned.");
System.exit(0);
case 1: System.out.println("Destination file has not mentioned.");
System.exit(0);
case 2: copyfile(args[0],args[1]);
System.exit(0);
default : System.out.println("Multiple files are not allow.");
System.exit(0);
} } }

```

Q5. (a) Explain access control used in Java with all types of specifiers characteristics. Also, give an example for each.

Refer to Page No.-106 to 109

(b) Write client and server programs in Java to show the TCP connection establishment and data transfer.

Ans. We start by importing the java.net package, which contains a set of pre-written networking routines (including InetAddress).

```
import java.net.*;
```

Next, we declare a new variable of type InetAddress, which we assign the value of the local host machine (for machines not connected to a network, this should represent 127.0.0.1). Due to the fact that InetAddresses can generate exceptions, we must place this code between a try .. catch UnknownHostException block.

```
// Obtain the InetAddress of the computer on which this program is running
InetAddress localaddr = InetAddress.getLocalHost();
```

The InetAddress class has methods that return the IP address as an array of bytes (which can be easily converted into a string), as well as a string representation of its domain name (e.g. mydomain.org). We can print out the InternetAddress, as well as the domain name of the local address.

```
System.out.println ("Local IP Address : " + localaddr );
System.out.println ("Local hostname : " + localaddr.getHostName());
```

```
// Program to show TCP connection
public class MyFirstInternetAddress
{
public static void main(String args[])
{
```

```
try
{
InetAddress localaddr = InetAddress.getLocalHost();

System.out.println ("Local IP Address : " + localaddr );
System.out.println ("Local hostname : " + localaddr.getHostName());
}
catch (UnknownHostException e)
{
System.err.println ("Can't detect localhost : " + e);
}

}

/** Converts a byte_array of octets into a string */
public static String byteToStr( byte[] byte_arr )
{
StringBuffer internal_buffer = new StringBuffer();

// Keep looping, and adding octets to the IP Address
for (int index = 0; index < byte_arr.length -1; index++)
{
internal_buffer.append ( String.valueOf(byte_arr[index]) + ".");
}

// Add the final octet, but no trailing '.'
internal_buffer.append ( String.valueOf (byte_arr.length) );

return internal_buffer.toString();
}
```

Note : Q. No. 1 is compulsory. Attempt any three questions from the rest.

Q1. (a) Consider a class that stores a Bank account holder's account number, ATM card number, account balance and ATM PIN. Write a program to store the data onto a disk file, except for the account balance and ATM PIN. Use serialization and transient variables.

Ans.

```
import Java.io.*;
import java.util.*;
public class UserAccount implements Serializable
{
    Private transient int atmPin_no;
    Private String atmCard_no;
    Private String account_number;
    Private transient float balance_amount;
    UserAccount(int pin,String cardno,String acc_no,float balance)//constructor
    {
        atmPin_no= pin;
        atmCard_no= cardno;
        account_number= acc_no;
        balance_amount= balance;
    }
    public String toString()
    {
        String pin=( atmPin_no==null)?"(n/a)": atmPin_no;
        Return "Logon info:\n "+"Account No." + account_number + "\n Balance:" +
        balance_amount;
    }
    Public static void main(String[] args)
    Throws IOException,ClassNotFoundException
    {
        UserAccount acc=new
        UserAccount (1234,"A983423115","23456789076543",5000.34);
        System.out.println("Login is="+ acc);
        ObjectOutputStream obj=new ObjectOutputStream(new
                FileOutputStream("Login.out"));
        obj.writeObject(acc);
    }
}
```

```

obj.close;
//now get them back
ObjectInputStream in=new ObjectInputStream (new
    FileInputStream("Login.out"));
Acc=(UserAccount)in.readObject();
System.out.println("login="+ acc);
}
}

```

(b) What makes compiled JAVA programs platform independent? Explain.
Refer to Page No.-20 to 21

(c) Explain the significance of PATH and CLASS PATH.

Ans. We keep all “executable files”(like .exe files) and “batch files”(like .bat) in PATH variable. And we keep all jar files and class files in CLASSPATH variables. So path is set according to the .exe files & .bat files and classpath is set according to the .jar files & .class files.

Set the PATH variable if you want to be able to conveniently run the executables (javac.exe, java.exe, javadoc.exe, and so on) from any directory without having to type the full path of the command. If you do not set the PATH variable, you need to specify the full path to the executable every time you run it, such as:
C:\Program Files\Java\jdk1.6.0\bin\javac MyClass.java

The CLASSPATH variable is one way to tell applications, including the JDK tools, where to look for user classes. (Classes that are part of the JRE, JDK platform, and extensions should be defined through other means, such as the bootstrap class path or the extensions directory.)

Operating system tries to find .exe and .bat files in path and JVM tries to find all classes in classpath.

(d) Is this a complete and legal comment?

```
/* */
```

Justify.

Ans. Yes it is complete and legal comment.

In java we can use Multiline comment and single line comment.

For multiline comment we can use /* */

```
/* stmt1
   Stmt2
   Stmt3 */
```

For single line we can use

```
// stmt
```

So /* */ is a legal comment.

(e) Are inner class secure and useful?

Ans. Inner classes are a major addition to the language. Java started out with the goal of being simpler than C++. But inner classes are anything but simple. The syntax is complex. (It will get more complex as we study anonymous inner classes later in this chapter.) It is not obvious how inner classes interact with other features of the language, such as access control and security. Must be carefully implemented to fix the security hole of the current implementation. Little additional overhead (regarding both code size and execution time). Implemented a byte code rewriter to incorporate the changes by transforming the byte code. Can be implemented in the compiler. Can extend this idea to have friend classes like C++.

(f) What are the two fundamental mechanisms for building a new classes of a existing class?

Ans. There are two fundamental mechanisms for building new classes from existing ones: inheritance and aggregation.

Inheritance extend the functionality of a class by creating a subclass. Override superclass members in the subclasses to provide new functionality. Make methods abstract/virtual to force subclasses to “fill-in-the-blanks” when the superclass wants a particular interface but is agnostic about its implementation. Aggregation create new functionality by taking other classes and combining them into a new class. Attach a common interface to this new class for interoperability with other code.

(g) The members of which classes can access members of a class in a particular package?

Ans. Only public package members are accessible outside the package in which they are defined. To use a public package member from outside its package, one or more of the following things to be done:

1. Refer to the member by its long(qualified) name.
2. Import the package member.
3. Import an entire package.

(h) Explain the significance of each argument in the function main (string args []).

Ans. If this is in reference to the “String[] args” passed as a parameter to the “**public static void main()**” method, then it refers to the list of arguments passed to the class when it is called. Eg: When the following command is executed “java abc xxx yyy” then in the main method of the abc.java, the first parameter of args i.e. args[0] would have “xxx” and the second parameter of args i.e. args[1] would have “yyy”. Referring to the 3rd or any other higher element would throw a “IndexOutOfBoundsException” exception

(i) Why cannot the “abstract” method modifier be used together with the “native” modifier.

Ans. A method that is native is implemented in platform-dependent code, typically written in another programming language such as C, C++, FORTRAN, or assembly language. The body of a native method is given as a semicolon only, indicating that the implementation is omitted, instead of a block.

A compile-time error occurs if a native method is declared abstract. This is logical because the native modifier implies that an implementation exists, and the abstract modifier insists that there is no implementation.

Q2. (a) Differentiate the following and support with example.

(i) Final and static member

Ans. We do not have to assign a value to a final variable when you declare it. The “final” keyword just means that you can (and must) assign its value exactly once. Where you are allowed to assign the value depends on if it is static or not (as well as its access modifier).

`private final int j;`

The value MUST be assigned ONCE, either here in the declaration or in each constructor. It is NOT automatically assigned a value of 0 (zero). Also, since it is not static, there is a distinct value for each instance object, so it makes sense that it must be assigned in the constructor.

`private static int k;`

The value may be assigned from anywhere in this class. Since it is not final, the value may be assigned any number of times, but since it is static, this value is stored at the class-level and universally available to all instances. If left unassigned, it is assumed to be 0 (zero).

`private static final int i;`

The value MUST be assigned ONCE, either here in the declaration or in a “static” class-level block. It is NOT automatically assigned a value of 0 (zero).

So, to review:

“final”: per-instance value, must be assigned once per instance.

“final” + “static”: per-class value, must be assigned once per class.

“static”: per-class value, can be assigned any number of times per class.

[neither modifier]: per-instance value, can be assigned any number of times per instance.

Before Java can create an instance of a class, it must first load and initialize that class. One step of class initialization is assigning values to all its static fields. If they’re static AND final, those values can’t change once they’re assigned.

(ii) Inheritance and aggregation:

Refer to Q.No.1(f) in same paper June-2010

iii) Abstract class and Interface

Refer to Chapter-13, Q.NO.-28, Page No.-193

iv) String and String Buffer

Refer to Chapter-3, Q.No.-5, Page No.-30

(b) Write a program to implement the following operations on a string :**(i) Convert to uppercase**

Ans. Public class ChangeUpper

```
{  
Public static void main(String args[])  
{  
String str="gullybaba"  
String strUpper=str.toUpperCase();  
System.out.println(strUpper);//displays GULLYBABA  
}  
}
```

(ii) Convert to lowercase

Ans. Public class ChangeLower

```
{  
Public static void main(String args[])  
{  
String str="GULLYBABA"  
String strLower=str.toLowerCase();  
System.out.println(strLower);//displays gullybaba  
}  
}
```

(c) What are user – defined exception?

Refer to Chapter-7, Q.No.-6, Page No.-97

Q3. (a) What is the difference between checked and unchecked exceptions?

Refer to Chapter-7, Q.No.-9, Page No.-101

(b) Create a class account with the following data members :

(i) account id

(ii) name

(iii) current balance

and implement the member functions :

(i) Withdraw**(ii) Deposit**

Ans: // AccountDemo.java

```

class Account
{
    protected double balance;
    protected String account_id;
    protected String name;
    // Constructor to initialize balance
    public Account( double amount )
    {
        balance = amount;
    }

    // Overloaded constructor for empty balance
    public Account(String id,String nm)
    {
        balance = 0.0;
        account_id=id;
        name=nm;
    }

    public void deposit(double amount )
    {
        balance += amount;
    }

    public double withdraw( double amount )
    {
        // See if amount can be withdrawn
        if (balance >= amount)
        {
            balance -= amount;
            return amount;
        }
        else
        // Withdrawal not allowed
        return 0.0;
    }

    public double getbalance()
    {
        return balance;
    }
}

```

```

        }
    }
class AccountDemo
{
    public static void main(String args[])
    {
        // Create an account
        Account my_account = new Account("34561234567","Gullybaba");
        System.out.println ("Account Id:" +my_account.account_id);
        System.out.println ("Name:" +my_account.name);
        System.out.println ("Initial Balance:" +my_account.balance);
        // Deposit money
        my_account.deposit(25000.00);
        // Print current balance
        System.out.println ("After Deposit Current balance "
+my_account.getbalance());
        // Withdraw money
        my_account.withdraw(5000.00);
        // Print remaining balance
        System.out.println ("After withdraw Remaining balance "
+my_account.getbalance());
    }
}

```

(c) How multiple inheritance is achieved in Java? Explain with an example.
Refer to Dec-2009, Q.No.04(b)

(d) Write an applet that prints “Here” at the current cursor position whenever the mouse left button is clicked.

Refer to Dec-2009, Q.No.-1(b)

Q4. (a) Write short notes on following :

(i) Interthread communication

Ans. Interthread communication:

Java provides a very efficient way through which multiple-threads can communicate with each-other. This way reduces the CPU's idle time i.e. A process where, a thread is paused running in its critical region and another thread is allowed to enter (or lock) in the same critical section to be executed. This technique is known as **Interthread communication** which is implemented by some methods. These methods are defined in “**java.lang**” package and can only be called within synchronized code shown as:

Method	Description
Wait()	It indicates the calling thread to give up the monitor and go to sleep until some other thread enters the same monitor and calls method notify() or notifyAll() .
Notify()	It wakes up the first thread that called wait() on the same object.
Notify All()	Wakes up (Unlocks) all the threads that called wait() on the same object. The highest priority thread will run first.

(ii) Event Handling in Java

Refer to Dec-2009, Q.No.-1(a)

(b) What is a layout managers? Explain layouts with example.

Refer to Q.No.-2(c), Page No.-247

(c) Write a Java program to set up JDBC and execute the following SQL statement on a database table employee-t with the fields emp-id, emp name, emp-department, emp-basic “SELECT * FROM employee-t where emp-basic < 10000;

Ans. // MyJDBC.java: Query an MSQl database using JDBC.

```
import java.sql.*;
class MyJDBC
{
    public static void main (String[] args)
    {
        try
        {
            String url = "jdbc:mysql://localhost/MyDb";
            Connection conn = DriverManager.getConnection(url,"","");
            Statement stmt = conn.createStatement();
            ResultSet rs;
            rs = stmt.executeQuery("SELECT * FROM employee_t WHERE
            emp_basic<10000");
            while ( rs.next() )
            {
                String Eid = rs.getString("emp_id");
                String EmployeeName = rs.getString("emp_name");
```

```
String EmpDepartment = rs.getString("emp_department");
float basic = rs.getFloat("emp_basic");
System.out.println(Eid);
System.out.println(EmployeeName);
System.out.println(EmpDepartment);
System.out.println(basic);
}
conn.close();
}
catch (Exception e) {
    System.err.println("Got an exception! ");
    System.err.println(e.getMessage());
}
}
```

Q5. (a) Write a program to sort the list given as command line arguments using bubble sort.

Ans. //Bubble sort of list using command line argument

```
import java.io.*;
public class Sort
{
    public static void main(String[] args)
    {
        /* Retrieves the filename of dataset from the command-line arguments */
        if (args.length != 1) {
            System.err.println("Usage:java Sort filename.txt");
            System.exit(1);
        }
        /* Reading data from the file system */
        try {
            int[] data = null;
            String line;
            FileReader input = new FileReader(args[0]);
            BufferedReader bufRead = new BufferedReader(input);
            line = bufRead.readLine();
            while (line != null) {
                if (line.length() > 0) {
                    String[] tokens = line.split(" ");
                    data = concat(data, str2int(tokens));
                }
            }
        } catch (Exception e) {
            System.out.println("An error occurred while reading the file: " + e.getMessage());
        }
    }
}
```

```

        }
        line = bufRead.readLine();
    }
    bufRead.close();
/* Create an instance of Sort class */
Sort s = new Sort();
/* Perform bubble sort on dataset and display the result */
int bubblesorted[] = s.bubble(data);
s.dispList(bubblesorted);
} catch (IOException e) {
    e.printStackTrace();
} // end of try ... catch
} // end of main

public static int[] bubble(int[] list) {
    int[] sortedList = null;
    System.out.println("Run bubble sort");
    int i = list.length ;
    for (int pass=1; pass < i; pass++) {
        // count how many passes through the array
        for (int j=0; i < i-pass; i++) {
            if (list[j] > list[j+1]) {
                // This swaps the elements in the array around
                int temp = list[j];
                list[j] = list[j+1];
                list[j+1] = temp;
            }
        }
    }
    sortedList = list;
    return sortedList;
}
/* The dispList will display all elements in the list */
public static void dispList(int[] list) {
    for (int i = 0 ; i < list.length ; i++) {
        System.out.print(list[i] + " ");
    }
    System.out.print("\n");
} // end of dispList

public static int[] concat(int[] A, int[] B) {
}

```

```

if (A != null && B != null) {
    int[] C = new int[A.length + B.length];
    System.arraycopy(A, 0, C, 0, A.length);
    System.arraycopy(B, 0, C, A.length, B.length);
    return C;
} else if (A != null) {
    return A;
} else if (B != null) {
    return B;
}
return null;
}
public static int[] str2int(String[] A) {
    int[] B = new int[A.length];
    for (int i=0 ; i < A.length ; i++) {
        B[i] = Integer.parseInt(A[i]);
    }
    return B;
}
}

} // end of Sort class

```

(b) What is Java Bean? What are its advantages?

Refer to Q.No.-5 and Q.No.-7, Page No.-142

(c) What is Introspection in Java Beans? Illustrate with a code fragment.

Refer to Q.No.-9, Page No.-145 and;

```

public class BeanDemo
{
    private final String name = "BeanDemo";
    private int size;

    public String getName()
    {
        return this.name;
    }

    public int getSize()
    {
        return this.size;
    }
}

```

```
}

public void setSize( int size )
{
    this.size = size;
}

public static void main( String[] args )
    throws IntrospectionException
{
    BeanInfo info = Introspector.getBeanInfo( BeanDemo.class );
    for ( PropertyDescriptor pd : info.getPropertyDescriptors() )
        System.out.println( pd.getName() );
}
```

Note: Q. No. 1 is compulsory. Attempt any three questions from the rest.

Q1. (a) Write a Java program to read the contents of binary file as integers and print out their values on the standard output.

Ans.

```
import java.io.*;
public class PrintConsol
{
    public static void main(String[] args)
    {
        try
        {
            FileInputStream FIS = new FileInputStream("Intro1.txt");
            int n;
            While ((n = FIS.available()) > 0)
            {
                int b;
                int result = FIS.read(b);
                if (result == -1) break;
                System.out.print(s);
            } // end while
            FIS.close();
        } // end try
        catch (IOException e)
        {
            System.err.println(e);
        }
        System.out.println();
    }
}
```

(b) What are anonymous classes? Explain with a code fragment.

Ans. Anonymous class is a class whose name is not assigned. It is also called as nameless class. It is generally used within another class.

e.g. Class demo

```
{Public static void main [string args [ ]]
Class {Public: interval a, interval, b} x;
x.a=5; x.b=7;
System.out.println ("sum="-(a+b));}
```

(c) What is the purpose of serialization of an object? How does the volatile modifier to a data type affect serialization?

Ans. Serialization is the process of writing the state of an object to a byte stream. This is useful to save the state of the programme to a persistent storage area, such as file. It is also needed to implement RMI. The volatile modifier tells the compiler

that the variable modified by volatile can be changed unexpectedly by other parts of the programme. Hence, it affects the serialization.

(d) Justify “Java is a secure” and “architectural neutral” Language.

Ans. Java is a secure: As you are likely aware, every time that you download a “normal” program, you are risking a viral infection. Prior to Java, most users did not download executable programs frequently, and those who did scanned them for viruses prior to execution. Even so, most users still worried about the possibility of infecting their systems with a virus. In addition to viruses, another type of malicious program exists that must be guarded against. This type of program can gather private information, such as credit card numbers, bank account balances, and passwords, by searching the contents of your computer’s local file system.

When you use a Java-compatible Web browser, you can safely download Java applets without fear of viral infection or malicious intent. Java achieves this protection by confining a Java program to the Java execution environment and not allowing it access to other parts of the computer. The ability to download applets with confidence that no harm will be done and that no security will be breached is considered by many to be the single most important aspect of Java.

Java is architecture-neutral : Java applications will run anywhere on the network on just about any kind of platform. I think this is one of the greatest strengths of Java. The Java compiler generates “byte-codes” from the source file, which are not processor-specific. Rather, these byte-codes run on the Java Virtual Machine which converts the code to the ones and zeros which a specific processor, like an Intel or a Motorola, understands. Thus Java programs can run on any processor and code does not need to be re-written for different processors. Hence, Sun’s Java slogan: “write once, run anywhere.” But when Microsoft starts tampering with the Java classes so they can only run on Windows, then the universality of the Java standard is compromised.

(e) Write a program to generate a Fibonaaci series.

Refer to Chapter-12, Q.No.-23, Page No.-166

(f) What is a synchronized block?

Ans. A synchronized block is used to implement synchronization. It ensures that a call to a method that is a member of object occurs only after the current thread has successfully entered object’s monitor.

(g) What is the difference between the notify() and notifyAll() methods?

Ans. Notify () function resumes execution of a thread waiting in the involving object whereas notify all resumes execution of all threads waiting on the invoking objects.

Q2. (a) Explain the purpose of Inet Address class and list three of its methods.

Refer to June-2007, Q.No.-4(b), Page No.-270

(b) Explain the difference between (a & b) and (a && b). Where a and b are logical expressions.

Refer to Dec-2008, Q.No.-2(a), Page No.-290

(c) Explain with example the use of the ternary operator.

Ans. Relational Operators compare two quantities to determine if they are equal or if one is greater than the other. Java has kept C’s question/colon, or conditional, operator that takes three operands (it is ternary operator). The first operand is

boolean, and the two other operands may be of any type. If the boolean operand is true, the result is the second operand; if it is false, the result is the third operand:

```
boolean b;
int c;
```

```
b = true;
c = (b ? 1 : 2); //1 will be assigned to c because b is true
```

```
b = false;
```

```
c = (b ? 1 : 2); //2 will be assigned to c because b is false.
```

These comparison operators are usually used in conjunction with conditional statements.

(d) Java is architecture neutral, secure and distributed language. Justify.

Ans. Refer to Q.No.-1(d)

Distributed language-Java is designed for the distributed environment of the Internet, because it handles TCP/IP protocols. In fact, accessing a resource using a URL is not much different from accessing a file. The original version of Java (Oak) included features for intra-address-spaces messaging. This allowed objects on two different computers to execute procedure remotely. Java has recently revived these interfaces in a package called *Remote Method Invocation (RMI)*. This feature brings an unparalleled level of abstraction to client/server programming.

Q3. (a) Develop an applet to display a solid circle that fits perfectly inside the outline of a square.

```
Ans. import java.awt.*;
import java.applet.*;
/* <applet code="Lines" width=300 height=300>
</applet>
*/
public class graphic extends Applet
{
public void paint (Graphics g)
{
g.drawRect(0,0, 100,100); g.fillOval(1,1, 99, 99); }}
```

(b) What is the difference between a process and a thread? Also mention any three methods of thread class and describe briefly.

Ans. A process is a programme in execution, whereas a thread is the smallest unit of despatchable code.

Methods of thread class are:

(1) join — Wait for a thread to terminate.

(2) run — Entry point for the thread.

(3) start — Start a thread by calling its run method.

(c) What is grid layout?

Refer to June-2009, Q.No.-4(a)[Grid Layout], Page No.-298

Q4. (a) Write a Java applet to present pull down list of the seven colours (violet, indigo, blue, green, yellow, orange, red) of a rainbow and allow the user to select one of the colours. Print out the index number of the selected choice.

Ans. import java.awt.*;

```

import java.awt.event.*;
import java.applet.*;
/* <applet code="color" width=300 height=200>
</applet>
*/
public class color extends Applet implements ItemListener
{
    Choice color;
    int I;
    public void init()
    {
        color=newChoice();
        color.add("violet");
        color.add("indigo")
        color.add("blue");
        color.add("green");
        color.add("yellow");
        color.add("red");
        add(color);
        color.addItemListener(this);
    }
    public void itemStateChanged(ItemEvent ie)
    {
        repaint();
    }
    public void paint(Graphics g)
    {
        color.drawString(color.getSelectedIndex(), 6,120);
    }
}

```

(b) What are the principles of event delegation model? What are the sources of event and event listener interface?

Refer to Dec-2006, Q.No.-2(a), Page No.-244

(c) Explain briefly the concept of RMI.

Refer to Chapter-11, Q.No.-1, Page No.-137

Q5. (a) Write a Java program that draws a line, a triangle or a circle depending on whether 2,3 or 1 point parameters are given to it:

Line between the two points, Triangle joining the three points, Circle of default radius at that point

```

Ans. import java.awt.*;
import java.applet.*;
/* <applet code="draw" width=300 height=300>
</applet>
*/
public class graphics extends Applet {
    public void paint(Graphics g)

```

```

g.drawOval(0,0,100,100);
g.drawLine(10,10, 10,100);
int x[]={10, 30, 50};
int y[]={25, 10, 25];
g.drawPolygon(x,y,3);
}
}
}

```

(b) What are checked and unchecked exceptions? Explain with example.

Refer to Dec-2007, Q.No.-3(c), Page No.-277

(c) What is wrong with the following program fragment? Make the needed changes to correct it.

```

double d;
d = 4.75;
switch (d)
{
case 0:
case1:
case2: { println ("small");
           break};
case 3:
case 4:
case 5: { println ("medium");
           break };
default:
           println ("large");
}

```

Ans. double d;
d = 4.75;
switch (d)
{
case 0:
case1:
case2: { System.out.println ("small");
 break};
case 3:
case 4:
case 5: { System.out.println ("medium");
 break };
default:
 System.out.println ("large");
}

Note: Q. No. 1 is compulsory. Attempt any three questions from the rest.

Q1. (a) Explain, with suitable examples, the advantage of object oriented language over structured programming language.

Refer to Chapter-1, Q.No.-2, Page No.-2

(b) Differentiate between method overloading and method overriding with an example.

Refer to June-2007, Q.No.-1(b), Page No.-263

(c) What is the use of class path? How it helps in the execution of a java program?

Refer to June-2008, Q.No.-1(d) Page No.-279

(d) Explain the advantages and disadvantages of garbage collection.

Refer to Chapter-2, Q.No.-13, Page No.-27

Disadvantages:

(1) Increase overhead in compiler (2) Degrades the throughput

(e) What is the use of Interface? How can you define and implement it using a program?

Refer to Chapter-8, Q.No.-3 & Q.No.-4, Page No.-110

(f) Differentiate checked and unchecked exception.

Refer to Chapter-7, Q.No.-9, Page No.-101

(g) Write a program in JAVA to convert given Full Name, e.g. “Ajay Singh” into First Name, i.e. “Ajay” and Last Name, i.e. “Singh”. Also sort the given list of names in ascending order of First Name?

Ans. class StringSort

```
{  
    public static void main(string args[])  
    string a[]={"Ram Gopal", "Ajay Singh", "Ram Singh" "Ashok Kumar");  
    for(int j=0; j<a.length; j++)  
    {  
        string sj=a[j].substring(0,a[j], indexOf(" "));  
        for(int i=j+1; i<a.length; i++)  
        {  
            string si=a[i].substring(0,a[i].indexOf(" "));  
            if(si.compareto(sj)<0)  
            {  
                String t=a[j];  
                a[j]=a[i];  
                a[i]=t;  
            }  
        }  
    }  
}
```

Q2. (a) How does java implements the model of interprocess synchronization using Threads? Explain with the help of a program.

Refer to June-2008, Q.No.-1(e), Page No.-280

(b) Write a program in Java which reads the content of a given file and writes it to the console.

Refer to Chapter-7, Q.No.-11, Page No.-101

Q3. (a) What is layout manager? Which is the default layout manager? Differentiate among the way of managing components by 5 different layouts.

Ans. Layout manager: Every Container, by default, has a layout manager (an object conforming to the LayoutManager protocol). If a Container's default layout manager doesn't suit your needs, you can easily replace it with another one. A layout managers range from the very simple (Container's default layout manager doesn't suit your needs, you can easily replace it with another one. A layout managers range from the very simple (FlowLayout and GridLayout) to the special purpose (BorderLayout and CardLayout) to the ultra-flexible (GridBagLayout).

Each applet brings up a window that you can resize to see how resizing affects the layout.

For a `java.applet.Applet`, the default layout manager is FlowLayout.

For the content pane of a `javax.swing.JApplet`, the default layout manager is a BorderLayout.

Various types of layout manager in Java and how to use them :

How to Use BorderLayout: BorderLayout is the default layout manager for all Windows, such as Frames and Dialogs. It uses five areas to hold components: north, south, east, west, and center. All extra space is placed in the center area. Here's an applet that puts one button in each area.

How to Use CardLayout : Use the CardLayout class when you have an area that can contain different Components at different times. CardLayouts are commonly tied to Choices, with the state of the Choice determining which Panel (group of Components) the CardLayout displays.

How to Use FlowLayout: FlowLayout is the default layout manager for all Panels. It simply lays out components from left to right, centering them within their row and, if necessary, starting new rows.

How to Use GridLayout: GridLayouts simply make a bunch of Components have equal size, displaying them in the requested number of rows and columns. Here's a picture of a program with a Panel that uses a GridLayout to control the display of 5 buttons.

How to Use GridBagLayout: GridBagLayout is the most sophisticated, flexible layout manager we provide. It aligns components vertically and horizontally, without requiring that the components be the same size.

(b) Write a program for reading from and writing to a URL connection.

Refer to Chapter-10, Q.No.-6, Page No.-130

Q4. (a) Explain advantage of exception. How exception sub classes are created?

Create your own exception subclass "My Ex class" which takes an string as input and throws a message "string too large" when the size of string is more than ten characters.

Ans. Advantages:

- (1) Automatic error checking & handling
- (2) Run-time error-management

```

Class MyExClass extends Exception
{
private String detail;
MyExClass (String a)
{
detail=a;
}
}
Class ExceptionDemo
{
static void compute(String a) throws MyExclass
{
System.out.println("Called compute(" + a + ")");
if (a.length()>10)
throw new MyExClass(a);
System.out.println("Normal Exit")
public static void main(string args[ ])
{
try
{
compute("hello");
compute("good.morning");
}
catch(MyExClass e)
{
System.out.println("caught" + e + "string too large");
}
}
}
}

```

(b) Differentiate between Datagram and stream communication.

Refer to June-2007, Q.No.-5(d), Page No.-274 & Dec-2008, Q.No.-4(d), Page No.-293

(c) Explain two uses of “final” keyword with the help of example.

Refer to Chapter-6, Q.No.-6, Page No.-85

Q5. (a) Write the process of implementing cookies. What is the role of session in handling a cookie.

Ans. A Cookie is created by calling the Cookie constructor, which takes two strings: the cookie name and the cookie value. Neither the name nor the value should contain whitespace or any of: [] () = , “ / ? @ : ;

The following are the attributes of the cookie:

getComment/setComment: Gets/sets a comment associated with this cookie.

getDomain/setDomain: Gets/sets the domain to which cookie applies. Normally, cookies are returned only to the exact hostname that sent them. You can use this method to instruct the browser to return them to other hosts within the same domain. Note that the domain should start with a dot (e.g. .prenhall.com), and must contain two dots for non-country domains like .com, .edu, and .gov, and three dots for country domains like .co.uk and .edu.es.

getMaxAge/setMaxAge: Gets/sets how much time (in seconds) should elapse before the cookie expires. If you don't set this, the cookie will last only for the current session (i.e. until the user quits the browser), and will not be stored on disk. See the LongLivedCookie class below, which defines a subclass of Cookie with a maximum age automatically set one year in the future.

getName/setName: Gets/sets the name of the cookie. The name and the value are the two pieces you virtually always care about. Since the getCookies method of HttpServletRequest returns an array of Cookie objects, it is common to loop down this array until you have a particular name, then check the value with getValue. See the getCookieValue method shown below.

getPath/setPath: Gets/sets the path to which this cookie applies. If you don't specify a path, the cookie is returned for all URLs in the same directory as the current page as well as all subdirectories. This method can be used to specify something more general. For example, someCookie.setPath("/") specifies that all pages on the server should receive the cookie. Note that the path specified must include the current directory.

getSecure/setSecure: Gets/sets the boolean value indicating whether the cookie should only be sent over encrypted (i.e. SSL) connections.

getValue/setValue: Gets/sets the value associated with the cookie. Again, the name and the value are the two parts of a cookie that you almost always care about, although in a few cases a name is used as a boolean flag, and its value is ignored (i.e. the existence of the name means true).

getVersion/setVersion: Gets/sets the cookie protocol version this cookie complies with. Version 0, the default, adheres to the original Netscape specification. Version 1, not yet widely supported, adheres to RFC 2109.

Webpages have no memories. A user going from page to page will be treated by the website as a completely new visitor. Session cookies enable the website you are visiting to keep track of your movement from page to page so you don't get asked for the same information you've already given to the site. Cookies allow you to proceed through many pages of a site quickly and easily without having to authenticate or reprocess each new area you visit.

Session cookies allow users to be recognized within a website so any page changes or item or data selection you do is remembered from page to page. The most common example of this functionality is the shopping cart feature of any e-commerce site. When you visit one page of a catalog and select some items, the session cookie remembers your selection so your shopping cart will have the items you selected when you are ready to check out. Without session cookies, if you click CHECKOUT, the new page does not recognize your past activities on prior pages and your shopping cart will always be empty.

(b) What is Inheritance? Explain its advantages. Also explain with example how a subclass is derived from a super class in Java.

Refer to Chapter-13, Q.No.-31, Page-196

Note: Q. No. 1 is compulsory. Attempt any three questions from the rest.

Q1. (a) Compare and contrast object-based programming language and object-oriented programming language.**Ans.**

Object based languages	Object oriented language
If the language supports only 3 features i.e. (data encapsulation, data abstraction & polymorphism). Then it is said to be object based programming language.	If the language supports all the 4 features i.e. (encapsulation, abstraction, polymorphism & also inheritance) then it is said to be object oriented programming language.
Doesn't support Inheritance	Support Inheritance
Example - Java Script, VB	Example - Java, C#, VB.Net
In Object Based Language, we can write program without object (eg. C++) .	In Object Oriented Language we can't write the program without object (eg. Java).

(b) What is inheritance? Differentiate between multilevel and multiple inheritance using an example.

Refer to Chapter-1, Q.No.-9, Page No.-10 & Chapter-6, Q.No.-1, Page No.-77

(c) Write a program in Java to calculate area and circumference of a circle. Show the use of 'final' keyword in your program.

Ans.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class AreaCircumferenceCircle {
    final static float pi = 22/7f;
    public static void main(String[] args) throws NumberFormatException, IOException
    {
        System.out.println("Enter the radius of the circle: ");
        InputStreamReader io= new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(io);
        float radius = Float.parseFloat(br.readLine());
        System.out.println("Area of the Circle is: "+radius*radius*pi);
        System.out.println("Circumference of the Circle is: "+2*radius*pi);
    }
}
```

(d)Write a program in Java for creating thread by inheriting the thread class.

Ans. We will see how to create thread by inheriting the thread class.

The steps for the same would be trivial:

(1) Create a class which inherits the Thread class.

(2) Override the run() method from the Thread class. (or else default run method from Thread, which is actually blank will be executed).

(3) Call the start() from or outside the class when you feel the thread is ready to shoot.

Here the code:

the client class (usage class) goes like this:

```
public class CounterTest2Client {
public static void main(String[] args) {
System.out.println("Started Thread: "+Thread.currentThread().getName());
CounterTest2 counterA = new CounterTest2("counterA");
CounterTest2 counterB = new CounterTest2("counterB");
System.out.println("Quitting from: "+Thread.currentThread().getName());
}
}
```

the Thread class goes like this:

```
public class CounterTest2 extends Thread {
private int counter;
public CounterTest2(String name) {
super(name);
counter = 0;
System.out.println("Init Thread: " + this);
// setDaemon(true);
start();
}
@Override
public void run() {
try {
while (counter < 5) {
System.out.println(getName() + ": " + counter++);
Thread.sleep(100);
}
} catch (Exception e) {
e.printStackTrace();
}
}
}
```

the result of the above will be:

Started Thread: main

Init Thread: Thread[counterA,5,main]

Init Thread: Thread[counterB,5,main]

counterA: 0

Quitting from: main

counterB: 0

counterB: 1

counterA: 1

counterA: 2

counterB: 2

counterA: 3

counterB: 3

counterB: 4

counterA: 4

(e) What is an interface in Java? Explain how an interface is created with the help of an example?

Refer to Chapter-8, Page No.-110, Q.No.-3 & Q.No.-4

(f) Write a program to find the length of a given string. Replace all the occurrences of a given character by a substitute character and print the resultant string.

Ans. /**

```
*****
* String replace Program |
*****|
```

* Takes three string input from the user
* Replaces all the occurrences of the second string
* with the third string from the first string
* @author Gullybaba
*/
/** Include Libraries */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
/** Define the max char length */
#define MAX_L 4096
/** Prototypes */
void replace (char *, char *, char *);
int main(void) {
 char o_string[MAX_L], s_string[MAX_L], r_string[MAX_L]; //String storing
variables
 printf("Please enter the original string (max length %d characters): ", MAX_L);
fflush(stdin);
gets(o_string);
printf("\nPlease enter the string to search (max length %d characters): ", MAX_L);
fflush(stdin);
gets(s_string);

```

printf("\nPlease enter the replace string (max length %d characters): ", MAX_L);
fflush(stdin);
gets(r_string);
printf("\n\nThe Original
string\n*****\n");
puts(o_string);
replace(o_string, s_string, r_string);
printf("\n\nThe replaced
string\n*****\n");
puts(o_string);
return 0;
}
/**/
* The replace function
*
* Searches all of the occurrences using recursion
* and replaces with the given string
* @param char * o_string The original string
* @param char * s_string The string to search for
* @param char * r_string The replace string
* @return void The o_string passed is modified
*/
void replace(char * o_string, char * s_string, char * r_string) {
//a buffer variable to do all replace things
char buffer[MAX_L];
//to store the pointer returned from strstr
char * ch;
//first exit condition
if(!(ch = strstr(o_string, s_string)))
return;
//copy all the content to buffer before the first occurrence of the search string
strncpy(buffer, o_string, ch-o_string);
//prepare the buffer for appending by adding a null to the end of it
buffer[ch-o_string] = 0;
//append using sprintf function
sprintf(buffer+(ch - o_string), "%s%s", r_string, ch + strlen(s_string));
//empty o_string for copying
o_string[0] = 0;
strcpy(o_string, buffer);
//pass recursively to replace other occurrences
return replace(o_string, s_string, r_string);
(g) Write a program to set the font of given string in text area as font name as "Arial", font size as 10 and font style as FONT.BOLD and FONT.ITALIC.

```

```

Ans. import java.applet.*;
import java.awt.*;
//<applet code=“Appf1.class” width=200 height=200></applet>
public class SetFont extends Applet
{
    Font a;
    public void init()
    {
        setBackground(Colour.red);
        a=new Font(“Arial”,Font.Bold+Font.Italic,10);
    }
    public void paint(Graphics g)
    {
        g.setFont(a);
        g.drawString(“Java”,10,80);
    }
}

```

Q2. (a) Explain the term Datagram. Describe the Datagram packet constructors along with their specific usage. Write a program for sending UDP Datagram’s?

Ans. A datagram is, to quote the Internet’s Request for Comments 1594, “a self-contained, independent entity of data carrying sufficient information to be routed from the source to the destination computer without reliance on earlier exchanges between this source and destination computer and the transporting network.” The term is used in several well-known communication protocols, including the User Datagram Protocol and AppleTalk.

A very similar term, packet, is used in the Internet Protocol and other protocols related to the Internet.

A datagram or packet needs to be self-contained without reliance on earlier exchanges because there is no connection of fixed duration between the two communicating points as there is, for example, in most voice telephone conversations. (This kind of protocol is referred to as connectionless.)

Actually the DatagramPacket class is a wrapper for an array of bytes from which data will be sent or into which data will be received. It also contains the address and port to which the packet will be sent.

Datagram Packet constructors:

```

public DatagramPacket(byte[] data, int length)
public DatagramPacket(byte[] data, int length, InetAddress host, int port)

```

You can construct a DatagramPacket object by passing an array of bytes and the number of those bytes to the DatagramPacket() constructor:

```

String s = “My first UDP Packet”
byte[] b = s.getBytes();
DatagramPacket dp = new DatagramPacket(b, b.length());

```

Normally the object of DatagramPacket is created by passing in the host and port to which you want to send the packet with data and its length. For example, object m in the code given below:

```

try
{
InetAddress m = new InetAddress("http://mail.yahoo.com");
int chargen = 19;
String s = "My second UDP Packet"
byte[] b = s.getBytes();
DatagramPacket dp = new DatagramPacket(b, b.length, m, chargen);
}
catch (UnknownHostException ex)
{
System.err.println(ex);
}

```

The byte array that's passed to the constructor is stored by reference, not by value. If you change its contents elsewhere, the contents of the DatagramPacket change as well.

DatagramPackets themselves are not immutable. You can change the data, the length of the data, the port, or the address at any time using the following four methods:

```

public void setAddress(InetAddress host)
public void setPort(int port)
public void setData(byte buffer[])
public void setLength(int length)

```

Now Refer to June-2009, Q.No.-5(a), Page No.-300

(b) What is JDBC? Explain how JDBC connectivity is established? Give an example of preparing and executing SQL statements using JDBC.

Refer to Dec-2006, Q.No.-1(g), Page No.-242

Q3. (a) What is Java beans ? Explain its features. Also, illustrate the difference between a Java bean and an instance of a normal Java class.

Refer to Chapter-11, Page No.-142, Q.No.-5 & Q.No.-7 & Q.No.-10, Page No.-146

(b) What is File class? Explain its use with an example program.

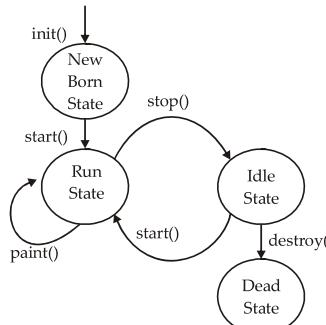
Refer to Chapter-13, Q.No.-29, Page No.-194

Q4. (a) Discuss the Input stream and output stream hierarchy. Write a program to read the binary file.

Refer to Dec-2006, Q.No.-4(d), Page No.-255 & Dec-2010, Q.No.-1(a), Page No.-326

(b) Write a brief description of Applet lifecycle. Discuss four low level events and their respective event listeners.

Refer to Dec-2006, Q.No.-1(e), Page No.-240



Low-Level Events: These events are those that represent a low-level input or windows-system occurrence on a visual component on the screen.

The various low-level event classes and what they generate are as follows:

- **A Container Event** Object is generated when components are added or removed from container.
- **A Component Event** object is generated when a component is resized moved etc.
- **A Focus Event** object is generated when component receives focus for input.
- **A Key Event** object is generated when key on keyboard is pressed, released etc.
- **A Window Event** object is generated when a window activity, like maximizing or close occurs.
- **A Mouse Event** object is generated when a mouse is used.
- **A Paint Event** object is generated when component is painted.

Event Listeners: An object delegates the task of handling an event to an **event listener**. When an event occurs, an event object of the appropriate type (as explained below) is created. This object is passed to a **Listener**. The listener must **implement the interface** that has the method for event handling. A component can have multiple listeners, and a listener can be removed using **remove Action Listener ()** method. You can understand a listener as a person who has listened to your command and is doing the work which you commanded him to do so.

The **low-level event listeners** of above low-level events are as follows:

- ComponentListener
- ContainerListener
- FocusListener
- KeyListener
- MouseListener
- MouseMotionListener
- WindowsListener.

Q5. (a) Discuss servlet life cycle. Write a servlet program to generate a dynamic HTML page for welcome of the user who logged in.

Ans. Refer to Chapter-11, Q.No.-4 , Page No.-141

13b.html

<html>

```
<head><title>13b</title></head>
<form action="http://localhost:8080/13b.jsp">
<p><center><br><br><h1>Verification of a particular User login</h1><br><br>
User Name : <input type="text name="uname" size=10><br>
password : <input type="password name="pwd" size=10><br><br>
<input type="submit value=Submit">
</form>
</html>
13b.jsp
<%! String username=null,password=null; %>
<% username=request.getParameter("uname");
password=request.getParameter("pwd");
%>
<%
if(username.equals("Gullybaba") && password.equals("12345"))
{
response.sendRedirect("http://localhost:8080/welcome.jsp");
}
else
out.println("<center><h2>Invalid username or password");
%>
welcome.jsp
<%
out.println("<center><h2>Welcome Gullybaba<br>");
out.println("You are now Logged in");
%>
```

Store the 13b.html, 13b.jsp and welcome.jsp in
C:\Program Files\Apache Group\Tomcat 4.1\webapps\ROOT
Start Tomcat

Open the Browser and give path as: <http://localhost:8080/13b.html>

(b) Discuss any four layouts. Also explain the interfaces used for the implementation of these layouts.

Refer to June-2009, Q.No.-4(a), Page No.-298

Refuse to accept anything but happiness
from others as well as from the world.

Note: Q. No. 1 is compulsory. Attempt any three questions from the rest.

Q1. (a) What is multithreading? Explain how does it help Java in its performance?

Refer to Dec-2005, Q.No.-4(c), Page No.-212

(b) Differentiate between throw and throws?

Refer to Chapter-7, Q.No.-4, Page No.-93

(c) What is Inheritance? Briefly explain importance of super keyword in Java?

Refer to Chapter-1, Q.No.-9, Page No.-10 & Chapter-6, Q.No.-4, Page No.-82

(d) What are shift operators? How many types of shift operators are available in Java?

Refer to June-2009, Q.No.-1(d), Page No.-295

(e) Briefly explain benefits of stream classes.

Ans. Importance of Java Stream Classes

The java.io package contains classes for performing input and output. The basic abstraction of the Java I/O system is a sequential stream of bytes or characters. The majority of the classes in this package represent and manipulate streams of bytes or characters. Java streams gain much of their power from their ability to be flexibly combined. For example, FileInputStream is a low-level class that reads bytes from a file (assuming, of course, that the program is not subject to security restrictions that prevent it from reading files). An InputStreamReader reads bytes from a byte stream, such as a FileInputStream, and converts the stream of bytes to a stream of characters. A BufferedReader reads characters in bulk from a character stream, such as an InputStreamReader, and buffers them up for more efficient use later. A Java program can combine these three modular stream classes to read text from a file in an efficient manner.

The java.io package contains numerous stream classes. There are classes that read data from files and write data to files. There are also classes that allow character stream and byte stream input to and output from strings and arrays. Other classes support pipes that transfer data to and from other threads in the same program.

Some of the most exciting capabilities of streams occur in conjunction with the networking capabilities of the java.net package (covered in the next section). The Java stream abstraction allows streams of bytes and characters to be read from and written to the network as easily as they can be read from and written to files on the local hard disk.

Another important feature of the java.io package is the ability to *serialize* a Java object into a stream of bytes that can be stored in a file or transferred across a network. Serialization is supported by the ObjectInputStream and ObjectOutputStream classes. The ObjectOutputStream serializes a Java object, while the ObjectInputStream does the opposite, reading a stream of bytes from a file, network, or other source and converting the data back into the Java object that it represents. This is a tremendously powerful capability in any object-oriented system, and is particularly useful for networked, object-oriented enterprise software.

The input/output capabilities of Java are not restricted entirely to streams of bytes and characters, however. The File class represents files and directories in the local file system and allows manipulation of those files and directories. The RandomAccessFile class allows random access to files, as an alternative to the sequential access allowed by the stream classes.

(f) Explain URL rewriting with an example.

Refer to Page No.-210 [URL Rewriting]

(g) Explain how exception handling is done in Java, with the help of an example.

Refer to Chapter-7, Q.No.-6, Page No.-97

(h) What is a package in Java? Explain how package is created in Java.

Refer to Chapter-8, Q.No.-1, Page No.-105

Q2. (a) Distinguish between the following terms with examples:

(i) Exception and Error

Refer to Chapter-7, Q.No.-1 & Q.No.-3, Page No.-90

(ii) Method overloading and overriding

Refer to Chapter-13, Q.No.-3, Page No.-172

(iii) Final and Finally

Refer to Chapter-6, Q.No.-6, Page No.-85 & Chapter-7, Q.No.-5, Page No.-95

(iv) Instance variables and class variables.

Refer to Chapter-1, Q.No.-8, Page No.-10 & Q.No.-6, Page No.-8

(b) What is constructor? Explain constructor overloading in Java with an example.

Refer to Chapter-5, Q.No.-2, Page No.-66

Q3. (a) What is the common usage of serialization?

Refer to Dec-2010, Q.No.-1(c), Page No.-326

(b) What is the result of compiling and running the following program?

```
{
Public static void main (string args [ ])
{
int i=-1;
I=i>>1;
System.out.println (i);
}
}
```

Ans. -1

(c) Explain use of keyword this with the help of program?

Refer to Chapter-6, Q.No.-3, Page No.-81

(d) Create a class with in this package “Amount Inwords” to convert amount into words. (Consider amount not to be more than 100000)

Ans. class constNumtoLetter

```
{
String[] unitdo = {"", " One", " Two", " Three", " Four", " Five",
" Six", " Seven", " Eight", " Nine", " Ten", " Eleven", " Twelve",
```

```
"Thirteen", "Fourteen", "Fifteen", "Sixteen", "Seventeen",
"Eighteen", "Nineteen"};
String[] tens = {"", "Ten", "Twenty", "Thirty", "Forty", "Fifty",
"Sixty", "Seventy", "Eighty", "Ninety"};
String[] digit = {"", "Hundred", "Thousand", "Lakh", "Crore"};
int r;
//Count the number of digits in the input number
int numberCount(int num)
{
int cnt=0;
while (num>0)
{
r=num%10;
cnt++;
num = num / 10;
}
return cnt;
}
//Function for Conversion of two digit
String twonum(int numq)
{
int numr, nq;
String ltr="";
nq = numq / 10;
numr = numq % 10;
if(numq>19)
{
ltr=ltr+tens[nq]+unitdo[numr];
}
else
{
ltr = ltr+unitdo[numq];
}
return ltr;
}
//Function for Conversion of three digit
String threenum(int numq)
{
int numr, nq;
String ltr="";
nq = numq / 100;
```

```
numr = numq % 100;  
  
if(numr == 0)  
{  
    ltr = ltr + unitdo[nq]+digit[1];  
}  
else  
{  
    ltr = ltr +unitdo[nq]+digit[1]+” and”+twonum(numr);  
}  
return ltr;  
}  
}  
}  
class originalNumToLetter  
{  
public static void main(String[] args) throws Exception  
{  
    //Defining variables q is quotient, r is remainder  
    int len, q=0, r=0;  
    String ltr = “”;  
    String Str = “Rupees”;  
    constNumtoLetter n = new constNumtoLetter();  
    int num = Integer.parseInt(args[0]);  
    if (num <= 0) System.out.println(“Zero or Negative number not for conversion”);  
    while (num>0)  
    {  
        len = n.numberCount(num);  
        //Take the length of the number and do letter conversion  
        switch (len)  
        {  
            case 8:  
                q=num/10000000;  
                r=num%10000000;  
                ltr = n.twonum(q);  
                Str = Str+ltr+n.digit[4];  
                num = r;  
                break;  
            case 7:  
            case 6:  
                q=num/100000;  
                r=num%100000;  
                ltr = n.twonum(q);  
                Str = Str+ltr+n.digit[3];  
        }  
    }  
}
```

```
num = r;
break;
case 5:
case 4:
q=num/1000;
r=num%1000;
ltr = n.twonum(q);
Str= Str+ltr+n.digit[2];
num = r;
break;
case 3:
if(len==3)
r = num;
ltr = n.threenum(r);
Str = Str + ltr;
num = 0;
break;
case 2:
ltr = n.twonum(num);
Str = Str + ltr;
num=0;
break;
case 1:
Str = Str + n.unitdo[num];
num=0;
break;
default:
num=0;
System.out.println("Exceeding Crore....No conversion");
System.exit(1);
}
if(num==0)
System.out.println(Str+" Only");
}
}
}
```

Q4. (a) What is URL? Explain two constructors for URL, in Java.net package.

Explain how you may connect to a URL in Java.

Refer to Chapter-10, Q.No.-5 & Q.No.- 6, Page No.-130

(b) Write an applet that draws circle, a line, and a polygon inside the applet's visible area.

Ans. import java.awt.*;
import java.applet.*;

```

/* <applet code="Draw" width=300 height=300>
</applet>
*/
public class graphic extends Applet
{
public void paint (Graphics g)
{
g.drawCircle(0,0, 100,100); g.drawLine(1,1, 99,99); g.drawPolygon(2,2, 98, 98); }}

```

(c) Compare the different layout managers in brief.

Refer to June-2011, Q.No.-3(a), Page No.-332

Q5. (a) Write a servlet program that fetches all data from client and stores it in a database successfully.

```

Ans. import java.io.*;
import java.util.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class DataServlet extends HttpServlet{
    private ServletConfig config;
    //Setting JSP page
    String page="DataPage.jsp";
    public void init(ServletConfig config)
        throws ServletException{
            this.config=config;
    }
    public void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException,IOException{
        PrintWriter out = response.getWriter();
        //Establish connection to MySQL database
        String connectionURL = "jdbc:mysql://192.168.10.59/
messagepaging";
        Connection connection=null;
        ResultSet rs;
        response.setContentType("text/html");
        List dataList=new ArrayList();
        try {
            // Load the database driver
            Class.forName("com.mysql.jdbc.Driver");
            // Get a Connection to the database
            connection = DriverManager.getConnection
(connectionURL, "root", "root");
            //Select the data from the database

```

```
String sql = "select * from message";
Statement s = connection.createStatement();
s.executeQuery(sql);
rs = s.getResultSet();
while(rs.next()){
    //Add records into data list
    dataList.add(rs.getInt("id"));
    dataList.add(rs.getString("message"));
}
rs.close();
s.close();
}catch(Exception e){
System.out.println("Exception is ;"+e);
}
request.setAttribute("data",dataList);
//Disptching request
RequestDispatcher dispatcher = request.getRequestDispatcher
Dispatcher(page);
if(dispatcher != null){
    dispatcher.forward(request, response);
}
}
```

(b) What do you mean by an event? Explain different components of an event.

Refer to Dec-2005, Q.No.-1(h), Page No.-204

The trick to problem solving is to get to the root of a problem before it even shows up.

Note: Q. No. 1 is compulsory. Attempt any three questions from the rest.

Q1. (a) Why java is known as architectural neutral?

Refer to Page No.-327[Architecture]

(b) What is method overloading? Explain your answer with suitable example.

Refer to Chapter-5, Q.No.-3, Page No.-67

(c) Explain static variable is used in Java?

Ans. Static Variable

- Static variables are declared with the static keyword in a class, but outside a method, constructor or a block.
- There would only be one copy of each class variable per class, regardless of how many objects are created from it.
- Static variables are rarely used other than being declared as constants. Constants are variables that are declared as public/private, final and static. Constant variable never change from their initial value.
- Static variable are stored in static memory. It is rare to use static variables other than declared final and used as either public or private constant.
- Static variables are created when the program start and destroyed when the program stop.
- Visibility is similar to instance variables. However, most static variables are declared public since they must be available for users of the class.
- Default values are same as instance variables. For numbers, the default value is 0; for Booleans, it is false; and for object references, it is null. Values can be assigned during the declaration or within the constructor. Additionally values can be assigned in special static initialise blocks.
- Static variables can be accessed by calling with the class name.
- When declaring class variables as public static final, then variables names (constants) are all in upper case. If the static variables are not public and final the naming syntax is the same as instance and local variables.

Example:

```
import java.io.*;
public class Employee
{
    // salary variable is a private static variable
    private static double salary;
    // DEPARTMENT is a constant
    public static final String DEPARTMENT = "Development";
    public static void main(String args[])
    {
```

```

        salary = 1000;
        System.out.println(DEPARTMENT+"average
        salary:"+salary);
    }
}

```

(d) What are the functions of the dot (.) operator? Explain with suitable example.

Ans. The dot (.) operator links the name of the object with the name of an instance variable. For example, to assign the width variable of mybox the value 100. We should use the following statement:

```
mybox.width=100;
```

This statement tells the compiler to assign the copy of width that is contained within the mybox object the value of 100. In general , we use the dot operator to access both the instance variables and the methods within an object.

Example:

```

class Box
{
    double width;
    double height;
    double depth;
}

class BoxDemo
{
    public static void main(String args[])
    {
        Box mybox=new Box();
        double vol;
        mybox.width=10;
        mybox.height=20;
        mybox.depth=15;
        vol=mybox.width*mybox.height*mybox.depth;
        System.out.println("Volume is "+vol);
    }
}

```

(e) Differentiate between Inheritance and Abstract class?

Refer to Chapter-1, Q.No.-9, Page No.-10 & June-2008, Q.No.-2(a), Page No.-281

(f) What is Interface in Java? Explain with an example how interfaces are created in Java.

Ans. Java does not support multiple inheritance, that class in java cannot have more than one super class.

```
class A extends B extends C
```

```
{
```

.....

.....

}

is not permitted in java. Java provides an alternate approach known as interface to support the concept of multiple inheritance. Although a java class cannot be a subclass of more than one super class, it can implement more than one interface. An interface is basically a kind of class. Like class, interface contains methods and variable but with major difference. The difference is that interfaces define only abstract methods and final fields.

The general form of an interface definition is:

Interface InterfaceName

{

```
    return_type method1(parameters);
    return_type method2(parameters);
    return_type method3(parameters);
    .....
    return_type method(parameters);
    type variable_Name1;
    type variable_Name2;
```

}

Implementing Interfaces: Interfaces are used as “superclasses” whose properties are inherited by classes. It is therefore necessary to create a class that inherits the given interface. This is done as follows:

class classname **implements** interfacename

{

body of classname

}

Example:

interface Area

{

```
    final static float pi=3.14F;
    float compute (float x, float y);
```

}

class Rectangle implements Area

{

```
    public float compute(float x, float y)
    {
        return(x*y);
    }
```

}

class Circle implements Area

{

```
    public float compute(float x, float )
    {
```

```

        return(pi*x*x);
    }
}

class InterfaceTest
{
    public static void main(String args[])
    {
        Rectangle rect =new Rectangle();
        Circle cir=new Circle();
        Arear area;
        area=rect;
        System.out.println("Area of Rectangle="+area.compute(10,20));
        area=cir;
        System.out.println("Area of Circle="+area.compute(10,0));
    }
}

```

(g) What is multithreading? Explain Different stages of a thread.

Ans. Multithreading is a conceptual programming paradigm where a program (process) is divided into two or more subprograms (processes), which can be implemented at the same time in parallel. For example, one program can display an animation on the screen while another may build the next animation to be displayed. This is something similar to dividing a task into subtasks and assigning them to different people for execution independently and simultaneously.

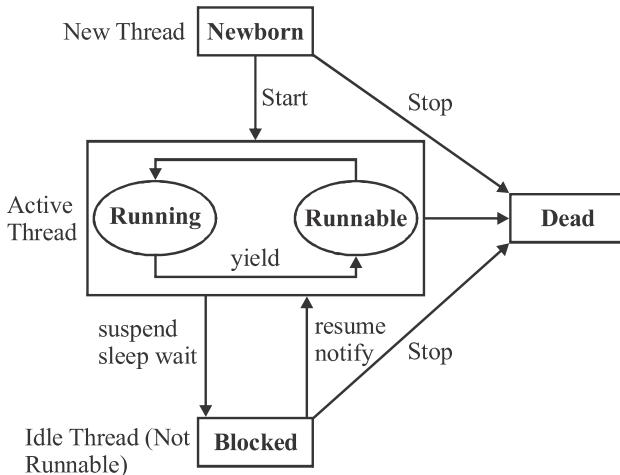
In most of our computers, we have only a single processor and therefore, in reality, the processor is doing only one thing at a time. However, the processor switches between the processes so fast that it appears to human beings that all of them are being simultaneously.

Multithreading is a powerful programming tool that makes java distinctly different from its fellow programming languages. Multithreading is useful in a number of ways. It enables programmers to do multiple things at one time. They can divide a long program (containing operations that are conceptually concurrent) into threads and execute them in parallel. For example, we can send tasks such as printing into the background and continue to perform some other task in the foreground.

Life Cycle of a Thread: During the life time of a thread, there are many states it can enter. They include:

- (1) Newborn state
- (2) Runnable state
- (3) Running state
- (4) Blocked state
- (5) Dead state

A thread is always in one of these five states. It can move from one state to another via a variety of ways as shown in figure.



Newborn State: When we create a thread object, the thread is born and is said to be in newborn state. The thread is not yet scheduled for running. At this state, we can do only of the following things with it:

- Schedule it for running using `start()` method.
- Kill it using `stop()` method.

If scheduled, it moves to the runnable state. If we attempt to use any other method at this stage, an exception will be thrown.

Runnable State: The runnable state means that the thread is ready for execution and is waiting for the availability of the processor. That is, the thread has joined the queue of threads that are waiting for execution. If all threads have equal priority, then they are given time slots for execution in round robin fashion, i.e. first-come, first-served manner. The thread that relinquishes control joins the queue at the end and again waits for its turn. This process of assigning time to threads is known as time-slicing.

Running State: Running means that the processor has given its time to the thread for its execution. The thread runs until it relinquishes control on its own or it is preempted by a higher priority thread.

Blocked State: A thread is said to be blocked when it is prevented from entering into the runnable state and subsequently the running state. This happens when the thread is suspended, sleeping, or waiting in order to satisfy certain requirements. A blocked thread is considered “runnable” but not dead and therefore fully qualified to run again.

Dead State: Every thread has a life cycle. A running thread ends its life when it has completed executing its `run()` method. It is a natural death. However, we can kill it by sending the `stop` message to it at any state thus causing a premature death. A thread can be killed as soon it is born., or while it is running, or even when it is in “not runnable” condition.

(h) What is Garbage collection? Explain advantages and disadvantage of Garbage collection in Java.

Ans. One of the key features of java is its garbage-collected heap, which takes care of freeing dynamically allocated memory that is no longer referenced. It shields the substantial complexity of memory allocation and garbage collection from the developer. Because the heap is garbage-collected, java programmers don't have to explicitly free the allocated memory.

The JVM's heap stores all the objects created by an executing java programs. We know Java's "new" operator allocate memory to object at the time of creation on the heap at run time. Garbage collection is the process of automatically freeing objects that are no longer referenced by the program. This frees the programmer from having to keep track of when to free allocated memory, thereby preventing many potential bugs and thus reduces the programmer's botheration.

The name "garbage collection" implies that objects that are no longer needed by the program are "garbage" and can be thrown away and collects back into the heap. We see this process as "memory recycling". When the program no longer references an object, the heap space it occupies must be recycled so that space is available for subsequent new objects. The garbage collector must somehow determine which objects are no longer referenced by the program and make available the heap space occupied by such unreferenced objects.

Advantages and Disadvantages of garbage collection

Giving the job of garbage collection to the JVM has several advantages. First, it can make programmers more productive. Programming in non-garbage-collected languages, the programmer has to spend a lot of time in keeping track of the memory de-allocation problem. In java, the programmer can use that time more advantageously in doing other work.

A second advantage of garbage collection is that it ensures program integrity. Garbage collection is an important part of java's security strategy. Java programmers feel free from the fear of accidental crash of the system because JVM is there to take care of memory allocation and de-allocation.

The major disadvantages of a garbage-collected heap are that it adds an overhead that can adversely affect program performance. The JVM has to keep track of objects that are being referenced by the executing program, and free unreferenced objects on the fly. This activity will likely take more CPU time than would have been required if the program explicitly free unrefined memory. The second disadvantage is the programmers in a garbage-collected environment have less control over the scheduling of CPU time devoted to freeing objects that are no longer needed.

**Q2. (a) Briefly explain role of following classes in Java Network programming
(i) Socket**

Ans. The socket class is designed to connect to server sockets and initiate protocol exchanges. The creation of a socket object implicitly establishes a connection between the client and server. There are no methods or constructors that explicitly

expose the details of establishing that connection. Here are two constructor used to create client sockets:

Socket(String hostname, int port)	Creates a socket connecting the local host to the named host and port; can throw an UnknownHostException or an IOException.
Socket(InetAddress ipAddress, int port)	Creates a socket using a preexisting InetAddress object and a port; can throw an IOException.

A socket can be examined at any time for the address and port information associated with it, by use of the following methods:

InetAddress getInetAddress()	Return the InetAddress associated with the socket object.
Int getPort()	Returns the remote port to which this Socket object is connected.
Int getLocalPort()	Returns the local port to which this socket object is connected.

(ii) Datagrams Packet

Ans. A datagram is an independent, self contained message sent over the network whose arrival, arrival time, and content are not guaranteed. Datagram runs over UDP protocols.

The UDP protocol provides a mode of network communication where packets sent by applications are called datagrams. A datagram is an indeoendent, self-contained message sent over the network whose arrival, arrival time, and content are not guaranteed. The datagram packet and datagram socket classes in java.net package implement system independent datagram communication using UDP.

DatagramPacket class is a wrapper for an array of bytes from which data will be sent or into which data will be received. It also contains the address and port to which the packet will be sent.

DatagramPacket constructors;

public DatagramPacket(byte[] data, int length)

public DatagramPacket(byte[] data, int length, InetAddress host, int port)

ou can construct a DatagramPacket object by passing in the host and port to which you want to send the packet with data and its length. For example,object m in the codegiven below:

try

{

InetAddress m=new InetAddress("http://mail.yahoo.com");

int chargen=19;

String s="My second UDPPacket";

byte[] b=s.getBytes();

```

        DatagramPacket dp=new DatagramPacket(b,b.length,m,chargen);
    }
    Catch(UnknownHostException ex)
    {
        System.err.println(ex);
    }
}

```

(b) Explain the difference between application and applet?

Ans.

Applet	Application
Small Program	Large Program
Used to run a program on client browser	Can be executed on standalone computer system
Applet is portable and can be executed by any JAVA supported browser	Need JDK, JRE, JVM installed on client machine
Applet applications are executed in a restricted environment	Application can access all the resources of the computer
Applets are created by extending the java.applet Applet	Applications are created by writing public static void main(String[] s) method
Applet application has 5 method which will be automatically invoked on occurrence of specific event	Application has a single start point which is main method
Example: <pre> import java.awt.*; import java.applet.*; public class My class extends Applet { public void init() {} public void start() {} public void stop() {} public void destroy() {} public void paint(Graphics g) {} }</pre>	<pre> public class My class { public static void main(String args[]) { } }</pre>

(c) What is object oriented programming? Explain basic concepts of object oriented programming, briefly.

Ans. The major objective of object-oriented approach is to eliminate some of the flaws encountered in the procedural approach. OOP treats data as a critical element in the program development and does not allow it to closely to the functions that operate on it and protects it from unintentional modification by other functions. OOP allows us to decompose a problem into a number of entities called Objects and then build data and functions around these entities. The combination of data and methods make up an object.

The data of an object can be accessed only by the methods associated with that object. However, methods of one object can access the methods of other objects. However, methods of one object can access the methods of other objects.

Some of the features of object-oriented paradigm are:

- Emphasis is on data rather than procedure.
- Programs are divided into what are known as Objects.
- Data Structures are designed such that they characterise the objects.
- Methods that operate on the data of an object are tied together in the data structure.
- Data is hidden and cannot be accessed by external functions.
- Objects may communicate with each other through methods.
- New data and methods can be easily added whenever necessary.
- Follows bottom-up approach in program design.

Basic Concepts of Object-Oriented Programming: Object-oriented is a term, which is interpreted differently by different people. It is therefore necessary to understand some of the concepts used extensively in object-oriented programming.

Object and Classes: Objects are basic runtime entities in an object-oriented system. They may represent a person, a place, a bank account, a table of data or any item that the program may handle.

When a program is executed, the objects interact by sending messages to one another. For example, ‘customer’ and ‘account’ are two objects in a banking program, then the customer object may send a message to the account object requesting for the balance.

A class may be thought of as a ‘data type’ and an object as a ‘variable’ of that data type. Once a class has been defined, we can create any number of objects belonging to that class. Each object is associated with the data of type class with which they are created. A class is thus a collection of objects of similar type. For example, mango, apple and orange are members of the class fruit. Classes are user-defined data types and behave like the built-in types of a programming language.

Data Abstraction and Encapsulation: The wrapping up of data and methods into a single unit (called class) is known as encapsulation. Data encapsulation is the most striking feature of a class. The data is not accessible to the outside world and only those methods, which are wrapped in the class, can access it. These methods provide the interface between the object’s data and the program. This insulation of the data from direct access by the program is called data hiding. Encapsulation makes it possible for objects to be treated like ‘black boxes’, each performing a specific task without any concern for internal implementation.

Abstraction refers to the act of representing essential features without including the background details or explanations. Classes use the concept of abstraction and are defined as a list of abstract attributes such as size, weight and cost, and methods that operate on these attributes. They encapsulate all the essential properties of the objects that are to be created.

Inheritance: Inheritance is a process by which objects of one class acquire the properties of objects of another class. This concept provides the idea of reusability.

This means that we can add the additional features to an existing class without modifying it. It is possible of driving a new class from the existing one. The new class will have the combining features of both the classes. In this example when the class child inherits the class parent, the child class is referred as drive class and the class parent is called base class.

Polymorphism: The meaning of polymorphism is same operation may behave differently at different places according to the data passed to it. Example, we have a function add, at one place this function is used for adding two integer values where it will work like the addition. At a second place for adding two strings, the result produced at this place will be in the form of concatenation.

Dynamic Binding: Dynamic binding (also called late binding) means that the code associated with a given procedure all is not known until its call at run-time.

Message Communication: An object-oriented program consists of asset of objects that communicate with each other. The process of programming in an object-oriented language, therefore, involves the following basic steps:

- Creating classes that define objects and their behaviour.
- Creating objects from class definitions.
- Establishing communication among objects.

Objects communicate with one another by sending and receiving information much the same way as people pass messages to one another. The concept of message passing makes it easier to talk about building systems that directly model or simulate their real-world counterparts.

Q3. (a) Explain the two types of exceptions available in java, with an example of each.

Ans. An exception is a condition that is caused by a run-time error in the program. If the exception object is not caught and handled properly, the interpreter will display an error message and will terminate the program. If we want the program to continue with the execution of remaining code, then we should try to catch the exception object thrown by the error condition and then display an appropriate message for taking corrective actions. This task is known as exception handling.

Types of Exceptions

Exception in java can be categorised into two types:

Checked Exception: These exceptions are explicitly handled in the code itself with the help of try-catch blocks. Checked exceptions are extended from the **java.lang.Exception** class.

For Example: **ClassNotFoundException**, **IllegalAccessException**, **NoSuchMethodException** etc.

```
class InvalidConfigFileNotFoundException extends Exception
{
    InvalidConfigFileNotFoundException(String errorMsg)
    {
        super(errorMsg);
    }
}
```

Unchecked Exception: These exceptions are not essentially handled in the program code; instead the JVM handles such exceptions. Unchecked exceptions are extended from the **java.lang.RuntimeException**.

For example: NullPointerException, ArithmeticException, IndexOutOfBoundsException etc.

```
class InvalidUsernamePasswordException extends RuntimeException
```

```
{
```

```
    InvalidUsernamePasswordException(String errorMsg)
    {
        super(errorMsg);
    }
```

```
}
```

(b) Explain when should you extend the thread class for creating a thread.

Ans. A thread can be created in java by extending Thread class, where you must override run() method.

- Call start() method to start executing the thread object.

Example:

```
package com.myjava.threads;
class MySmpThread extends Thread
{
    public static int myCount = 0;
    public void run()
    {
        while(MySmpThread.myCount <= 10)
        {
            try
            {
                System.out.println("Expl Thread:
                    "+(++MySmpThread.myCount));
                Thread.sleep(100);
            }
            catch (InterruptedException iex)
            {
                System.out.println("Exception in thread:
                    "+iex.getMessage());
            }
        }
    }
}
public class RunThread
{
    public static void main(String a[])
}
```

```

{
    System.out.println("Starting Main Thread...");  

    MySmpThread mst = new MySmpThread();  

    mst.start();  

    while(MySmpThread.myCount <= 10)  

    {
        try
        {
            System.out.println("Main Thread:  

                "+(++MySmpThread.myCount));  

            Thread.sleep(100);
        }
        catch (InterruptedException iex)
        {
            System.out.println("Exception in main thread:  

                "+iex.getMessage());
        }
    }
    System.out.println("End of Main Thread...");  

}
}

```

(c) Write a program to read the contents of file into a character array and write it into another file. Get names of the file from the user through standard input.

Ans. public class CopyFile

```

{
    private static void copyfile(String srFile, String dtFile){
        try
        {
            File f1=new File(srFile);
            File f2=new File(dtFile);
            InputStream in=new FileInputStream(f1);
            OutputStream out=new FileOutputStream(f2,true);
            OutputStream out=new FileOutputStream(f2);
            Byte[] buf=new byte[1024];
            int len;
            while((len=in.read(buf))>0)
            {
                out.write(buf,0,len);
            }
            in.close();
            out.close();
            System.out.println("File copied");
        }
    }
}
```

```

        catch(FileNotFoundException ex)
        {
            System.out.println(ex.getMessage() + "in the specified directory");
            System.exit(0);
        }
        catch(IOException e)
        {
            System.out.println(e.getMessage());
        }
    }
    public static void main(String[] args)
    {
        switch(args.length)
        {
            case 0: System.out.println("File has not mentioned");
            System.exit(0);
            case 2: copyfile(args[0],args[1]);
            System.exit(0);
            default: System.out.println("Multiple files are not allowed.");
            System.exit(0);
        }
    }
}

```

Q4. (a) Can we import the same package class twice? Will the JUM load the package twice at runtime? Justify your answer.

Ans. One can import the same package or same class multiple times. Neither compiler nor JVM complains about it. And the JVM will internally load the class only once no matter how many times you import the same class.

JVM is the main concept of java architecture and it is the part of the JRE. It provides the cross platform functionality to Java. This is the software process that converts the compiled Java byte code to machine code. Byte code is an intermediary language between Java source and the host system. Mostly compiler produce code for a particular system but Java compiler produce code for a virtual machine. JVM provides security to Java.

The programs written in Java or the source code translated by Java compiler into bytecode and after that the JVM converts the byte code into machine code for the computer one wants to run.

(b) What are the characteristics of JDBC? What are the various types of JDBC? Write a program to demonstrate how JDBC connection is established.

Ans. Java Database Connectivity: During programming you may need to interact with database to solve your problem. Java provides JDBC to connect to databases and work with it. Using standard library routines, you can open a connection to the database. Basically JDBC allows the integration of SQL calls into a general programming environment by providing library routines, which interface with the

database. In particular, java's JDBC has a rich collection of routines which makes such an interface extremely simple and intuitive.

Establishing A Connection: The first thing to do, of course, is to install java, JDBC and the DBMS on the working machines. Since you want to interface with a database, you would need a driver for this specific database.

Load the vendor specific driver: This is very important because you have to ensure portability and code reuse. The API should be designed as independent of the version or the vendor of a database as possible. Since different DBMS's have different behaviour, you need to tell the driver manager which DBMS you wish to use, so that it can invoke the correct diver.

For example, an Oracle driver is loaded using the following code snippet:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

Make the connection: Once the driver is loaded and ready for connection to be made, you may create an instance of a connection object using:

```
Connection con=DriverManager.getConnection(url,username,password);
```

Creating JDBC Statements: A JDBC Statement object is used to send the SQL statements to the DBMS. It is entirely different from the SQL statement. A JDBC Statement object is an open connection, and not any single SQL statement. You can think of a JDBC Statement object as a channel sitting on a connection, and passing one or more of the SQL statements to the DBMS.

An active connection is needed to create a statement object. The following code is a snippet, using our connection object con

```
Statement statmnt=con.createStatement();
```

Creating JDBC PreparedStatement: PreparedStatement object is more convenient and efficient for sending SQL statements to the DBMS. The main feature, which distinguishes PreparedStatement object from objects of statement class, is that it gives an SQL statement right when it is created.

The following code shows how to create a parameterised SQL statement with three input parameters:

```
PreparedStatement prepareUpdatePrice=con.prepareStatement("UPDATE Employee SET emp_address=? WHERE emp_code='1001' AND emp_name=?");
```

Example:

```
import java.sql.*;
class test
{
    public static void main(String s[])
    {
        try
        {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection con=DriverManager.getConnection
                ("jdbc;odbc:abc", "sa", "");
            Statement st=con.createStatement();
        }
    }
}
```

```

        ResultSet rs=st.executeQuery("select * from tb");
        while(rs.next())
        {
            System.out.println(rs.getString("name"));
            System.out.println(rs.getString("password"));
        }
    }
    catch(Exception e)
    {
        System.out.println("Exception Found"+e);
    }
}
}

```

(c) WAP to reverse a linked list and then copy all elements of a linked list into another list.

Ans. class link

```

{
    int data;
    public link nextlink;
    link(int d1)
    {
        data = d1;
    }
}
class List
{
    link head;
    link revhead;
    List()
    {
        head = null;
    }
    boolean isEmpty(link head)
    {
        return head==null;
    }
    void insert(int d1)
    {
        link templink = new link(d1);
        templink.nextlink = head;
        head = templink;
    }
    void printlist()

```

```

    {
        link head1 = head;
        while(!isEmpty(head1))
        {
            System.out.print(head1.data + " ");
            head1 = head1.nextlink;
        }
        System.out.println();
    }
}

void reverse()
{
    link previous=null,temp=null;
    while(isEmpty(head))
    {
        temp = head.nextlink;
        head.nextlink = previous;
        previous = head;
        head = temp;
    }
}

}

public class LinkedList
{
    public static void main(String[] args)
    {
        List list1 = new List();
        list1.insert(10);
        list1.insert(20);
        list1.insert(30);
        list1.insert(40);
        list1.insert(50);
        list1.printlist();
        list1.reverse();
        list1.printlist();
    }
}
}

```

Q5. (a) What is String Buffer? How it is different from String? Write a Java program which take your name as input and print it in upper case.

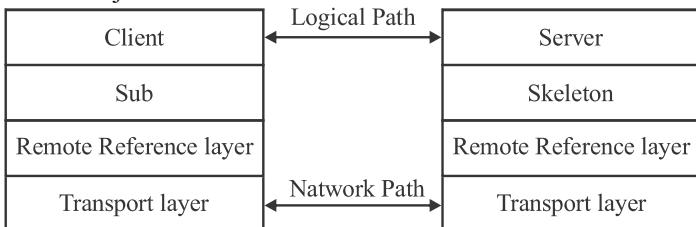
Refer to Chapter-3, Q.No.-5, Page No.-30

(b) What is RMI? Explain RMI Architecture with suitable diagram.

Ans. RMI is one of the core java APIs since version 1.1. It provides a framework for distributed computing. With RMI, separate parts of a single program can exist

in multiple java environments on multiple machines. To the client and server, the program, operates much like it is local to their environment and within a single memory space. RMI is one of the fundamental APIs that Enterprise java beans are built on. There are many cases where a distributed environment is beneficial. As applications grow large and complex they are more manageable, highly performer, and more robust when they are distributed among multiple machines. Resources can be physically located where they make the most sense. Dedicated machines can be configured optimally to perform specialised operations. Heavy processing can be allocated to the well suited hardware, data processing can be handled by centralised data servers, and web servers can just serve web pages. A large application can distribute the workload and achieve better efficiency and data isolation. RMI is a proven framework to design and implement a distributed application in a java environment. Due to the multiplatform nature of java, RMI is probably the best choice of API for this type of application.

The client is defined as the application which uses a remote object, and the server is defined as the machine which hosts the remote object. Communication is two way. Below is the java RMI architecture.



To both the client and server, the application appears to be local even though it is actually distributed among multiple java environments. This is achieved by implementing interfaces which define the methods which can be invoked. The interfaces define the contract between the objects. What is actually passes back and forth are primitive data types, remote references, or copies of local objects. As one would expect, changes to the remote reference results in changes to the object; changes to the copy do not. Remote objects are listed in the rmiregistry of the server. A client will query the rmiregistry for an object by its handle and will receive a reference to the remote object. The client then invokes methods on the remote object. What actually is happening is that the client is invoking methods locally on the stub which carries out the interaction. Likewise on the server side, the server interacts with the skeleton. The operations of the remote Reference layer are hidden from both sides. This allows different implementation of the java Virtual Machine, which may be on different platforms, to integrate seamlessly.

(c) Why we use Java Beans in Java? Explain features of Java Beans.

Refer to Page No.-142, & Chapter-11, Q.No.-7, Page No.-143

Note: Q. No. 1 is compulsory. Attempt any three questions from the rest. Attempt all parts of questions together.

Q1. (a) Bring out the difference between the string and StringBuffer classes in Java.

(b) Explain the concept of inheritance. Give a suitable example by declaring a class and showing two levels of inherited classes with the methods of each.

(c) There are some classes that reside in a public package called my Package. Give three ways in which these can be accessed in the default unnamed package. If one of the classes in my Package has the same name as a class in the default package, how can it be referred to in the default package?

(d) What are the steps involved in creating a distributed application using Remote Method Invocation?

(e) (i) How would you transfer control from the innermost loop of a thrice nested loop to the next statement outside the loops?

(ii) What are the conditions necessary for a class to be serializable?

(iii) Why is it not necessary to handle Runtime Exception?

(f) When is the “this” keyword used and why? Write a program fragment to illustrate.

(g) Write a Java program to read a text file and output the number of case insensitive occurrences of the sequence of letters “the” in it.

Q2. (a) What are Java controls and containers? Write an applet that displays three labels and four non-exclusive checkboxes on the screen.

(b) Construct a programming problem where the switch statement cannot be used for taking decisions. Now write a Java program to solve the problem.

(c) Write a program to track sessions through hidden form fields. What is the disadvantage of this approach?

Q3. (a) What is polymorphism? Write a program to illustrate late binding in Java.

(b) “Experiment 1” is a text file. Write a Java program to read “Experiment 1” and copy the contents of “Experiment 1” to a new file “Experiment 2”. Also count the number of words and lines copied.

Q4. (a) There is a database with a table emp_t that has employee_id, name, old_basic_salary and new_basic. Write a Java program to find all employees whose old basic salary is less than ₹20,000 and give them a 10% raise, while all others get a 6% raise. Update emp_t with the new basic salary of each employee.

(b) Suppose x in octal notation is $(567)_8$ and similarly $y = (-325)_8$. Write down the values of:

- (i) $x \ll y$ (ii) x/y (iii) $x \& y$ (iv) $x \gg 3$
(v) $x \ggg 3$ (vi) $x \wedge y$ (vii) $\sim x$ (viii) $\sim y$

Q5. (a) What is meant by thread priority? Write a Java program that creates two threads—one with minimum and the other with maximum priority. The first prints the letters A to Z while the second prints the numbers 1 to 26. Which thread would you expect to finish first and why?

(b) Write a Java program that finds:

- (i) the area of a circle with that radius if 1 parameter is passed to it.
(ii) the area of a rectangle with those sides if 2 parameters are passed to it.
(iii) the surface area of a box if 3 parameters are passed to it.
(c) Explain how to compile and run a Java Servlet.

Best Help Books for IGNOU STUDENTS



'Call' or 'SMS at:
9312235086

Note: Q. No. 1 is compulsory. Attempt any three questions from the rest. Attempt all parts of questions together.

Q1. (a) Bring out the difference between the statement and prepared statement classes in Java.

(b) What is the JVM? How does it help to make Java machine independent?

(c) What is Java bean? List its main features.

(d) Draw a chart to show in a class, for each access specifier, which member (class, package, subclass or all) may NOT reference data or methods of that class?

(e) Explain when the transient and volatile modifiers are used?

(f) Describe the concept of method overriding, with an example.

(g) Write a Java program to count the number of words and lines in a test file using the string Tokeniser class.

Q2. (a) Write an applet that draws a coloured line from the last point on which the mouse was clicked to the location of the current mouse click. The colour of the line should be a parameter passed to the applet when invoking it.

(b) Write a Java program to show that when more than one thread is waiting for an object's monitor, and the monitor wakes up all threads waiting on it, the highest priority thread that has been waiting is run first.

Q3. (a) There are two files called file 1 and file 2 each containing some lines of text (such as names of people) in sorted order. Write a program to print total number of words in both the files. Print which file has larger number of words.

(b) Arrange the following operators in descending order of precedence?, &&, ~, instanceof, + +, , | |, %

(c) (i) How can you prevent a class from being inherited?

(ii) Can an abstract class have a method that is not abstract?

(iii) Can a class reference the members of its superclass?

(iv) Is multiple inheritance allowed in Java? Is multi-level inheritance allowed?

(v) Can you have a try block without a catch clause?

Q4. (a) In a two dimensional Java array, can each row have a different number of columns? Write a Java program to initialise the contents of a 3 dimensional array of size $9 \times 6 \times 11$ such that each element is the sum of the cubes of its subscripts. For example, a [5] [4] [8] = $125 + 64 + 512 = 701$

(b) Write a Java program to receive a UDP datagram on port 9701 and place the data received into a file.

Q5. (a) Write a Java class that takes as input an integer array of numbers and prints out the sum of all the elements if there are 16 or less. If there are more elements it throws an exception which gives a message showing how many extra elements (more than 16) there are in the array.

(b) What is an interface? How is it declared and used? How does it differ from an abstract class? What is meant by implementing an interface?

Note: Q. No. 1 is compulsory. Attempt any three questions from the rest. Attempt all parts of questions together.

Q1. (a) What is object oriented programming? How is it different from structured programming? Explain advantages of using object oriented programming.

Ans. Object-oriented programming (OOP) is a programming paradigm that represents the concept of “objects” that have data fields (attributes that describe the object) and associated procedures known as methods. Objects, which are usually instances of classes, are used to interact with one another to design applications and computer programs.

Object Oriented Programming is different from Structured Programming

(1) The program is written as a collection of objects which communicate with each other, whereas in structured programming the program is represented as a logical structure.

(2) The basic entity is object. Each computation is performed using objects only, whereas in structured programming the flow of execution of the programming is dependent on the structure of the program.

(3) Data is given more importance whereas in structured programming code is given more importance.

(4) It can handle very complex programs, whereas structured programming can handle up to moderately complex programs.

(5) More data security as compared to structured programming.

(6) More reusability, abstraction and flexibility than structured programming.

(7) It is bottom up approach and structured programming is top down approach.

Advantages of using OOP's are:

- **Code Reuse and Recycling:** Objects created for Object Oriented Programs can easily be reused in other programs.

- **Encapsulation:** Objects have the ability to hide certain parts of themselves from programmers. This prevents programmers from tampering with values they shouldn't. Additionally, the object controls how one interacts with it, preventing other kinds of errors.

- **Design Benefits:** Large programs are very difficult to write. Object Oriented Programs force designers to go through an extensive planning phase, which makes for better designs with less flaws. In addition, once a program reaches a certain size, Object Oriented Programs are actually easier to program than non-Object Oriented ones.

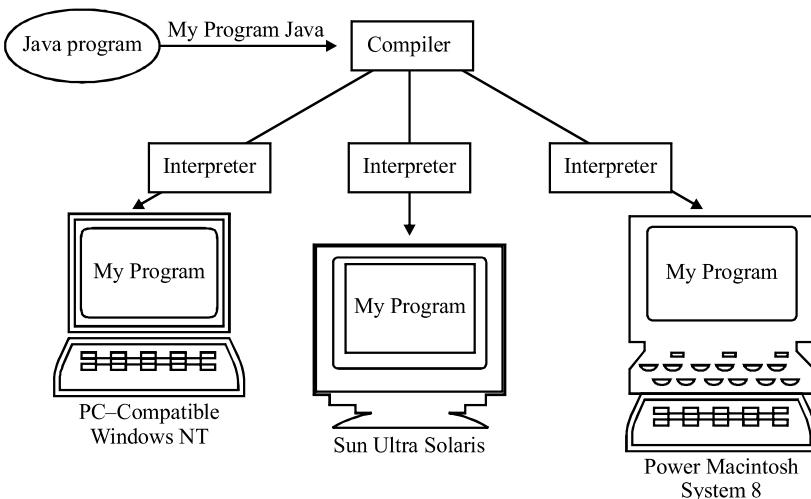
- **Software Maintenance:** Programs are not disposable. Legacy code must be dealt with on a daily basis, either to be improved upon (for a new version of an

exist piece of software) or made to work with newer computers and software. An Object Oriented Program is much easier to modify and maintain than a non-Object Oriented Program. So although a lot of work is spent before the program is written, less work is needed to maintain it over time.

(b) Explain why is Java called platform independent language?

Ans. Java is considered as platform independent because of many different reasons which are listed below:

- (1) Output of a java compiler is Non executable code, i.e. byte code.
- (2) Byte code is a highly optimised set of instructions.
- (3) Byte code is executed by java run-time system, which is called the java virtual machine (JVM).
- (4) JVM is an interpreter.
- (5) JVM accepts byte code as input and execute it.
- (6) Translating a java program into byte code makes it much easier to run a program in a wide variety of environments because only the JVM needs to be implemented for each platform.
- (7) For a given system we have run-time package, once JVM is installed for particular system then any java program can run on it.
- (8) However internal details of JVM will differ from platform to platform but still all understand the same java byte code.



(c) What is abstraction? Explain why classes in OOP, known as abstract data types.

Ans. Abstraction is process of hiding the implementation details and showing only the functionality. A class that is declared as **abstract** is known as abstract class.

Syntax:

abstract class <class-name>{}

An abstract class is something which is incomplete and you cannot create instance of abstract class. If you want to use it you need to make it complete or concrete by extending it. A class is called concrete if it does not contain any abstract method and implements all abstract method inherited from abstract class or interface it has implemented or extended.

(d) What is Java applet? How is it different from Java application program?

Ans. A Java applet is a small application which is written in Java and delivered to users in the form of byte code. The user launches the Java applet from a web page, and the applet is then executed within a Java Virtual Machine (JVM) in a process separate from the web browser itself. A Java applet can appear in a frame of the web page, a new application window, Sun's AppletViewer, or a stand-alone tool for testing applets. Java applets can be written in any programming language that compiles Java byte code.

Applet is different from Java Application Program:

Applet	Application
(1) Small program	(1) Large program
(2) Used to run a program on client browser.	(2) Can be executed on stand alone computer system.
(3) Applet is portable and can be executed by any java supported browser.	(3) Need JDK, JRE, JM installed on client machine.
(4) Applet applications are executed in a restricted environment.	(4) Application can access all the resources of the computer.
(5) Applets are created by extending the applet. Applet	(5) Applications are created by writing public static void main (String [] s) method.
(6) Applet application has 5 methods which will be automatically invoked on occurrence of specific event.	(6) Application has a single start point which is main method.

(e) Explain any five interfaces of the Java IO package.

Ans. Five interfaces of Java IO package are:

- (1) Java.io.BufferedReader
- (2) Java.io.BufferedWriter
- (3) Java.io.FileDescriptor
- (4) Java.io.InputStream
- (5) Java.io.OutputStream

The **Java.io.BufferedInputStream** class adds functionality to another input stream, the ability to buffer the input and to support the mark and reset methods. Following are the important points about BufferedInputStream:

- When the BufferedInputStream is created, an internal buffer array is created.

- As bytes from the stream are read or skipped, the internal buffer is refilled as necessary from the contained input stream, many bytes at a time.

Syntax:

```
public class BufferedInputStream  
extends FilterInputStream
```

The Java.io.BufferedOutputStream class implements a buffered output stream. By setting up such an output stream, an application can write bytes to the underlying output stream without necessarily causing a call to the underlying system for each byte written.

Syntax:

```
public class BufferedOutputStream  
extends FilterOutputStream
```

The Java.io.BufferedReader class reads text from a character-input stream, buffering characters so as to provide for the efficient reading of characters, arrays, and lines. Following are the important points about BufferedReader:

The buffer size may be specified, or the default size may be used.

Each read request made of a Reader causes a corresponding read request to be made of the underlying character or byte stream.

Syntax:

```
public class BufferedReader  
extends Reader
```

The Java.io.BufferedWriter class writes text to a character-output stream, buffering characters so as to provide for the efficient writing of single characters, arrays, and strings. Following are the important points about BufferedWriter:

- The buffer size may be specified, or the default size may be used.
- A Writer sends its output immediately to the underlying character or byte stream.

Syntax:

```
public class BufferedWriter  
extends Writer
```

The Java.io.FileDescriptor class instances serve as an opaque handle to the underlying machine-specific structure representing an open file, an open socket, or another source or sink of bytes. Following are the important points about FileDescriptor:

- The main practical use for a file descriptor is to create a FileInputStream or FileOutputStream to contain it.
- Applications should not create their own file descriptors.

Syntax:

```
public final class FileDescriptor  
extends object
```

(f) What is Exception? Explain how exceptions are handled in Java. Write a Queue class that throws an exception when the Queue is full. Display exception message as “Queue is full”.

Ans. An *exception* is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions. An exception is a

problem that arises during the execution of a program. An exception can occur for many different reasons, including the following:

A user has entered invalid data.

- A file that needs to be opened cannot be found.

- A network connection has been lost in the middle of communications or the JVM has run out of memory.

In general, an exception is handled (resolved) by saving the current state of execution in a predefined place and switching the execution to a specific subroutine known as an exception handler. If exceptions are continually, the handler may later resume the execution at the original location using the saved information. For example, a floating point divide by zero exception will typically, by default, allow the program to be resumed, while an out of memory condition might not be resolvable transparently.

```
import java.util.Scanner;
class Queue
{
    public static void main (String ar [])
    {
        try
        {
            int a [ ]=new int [5];
            Scanner in = new Scanner (System.in);
            int i, j;
            System.out.println ("enter the number");
            for (i=0; i<a.length; i++)
            {
                a [i] = in.nextInt ();
            }
            for (i=0; i<a.length; i++)
            {
                System.out.println (a [i]);
            }
        }
        catch (Exception e)
        {
            System.out.println ("Queue is full");
        }
    }
}
```

(g) Explain the Java RMI architecture with the help of a diagram.

Refer to Dec-2012, Q.No.-5(b), Page No.-365

Q2. (a) What is multithreading? Explain the steps for creating thread in Java. Also, explain different predefined thread priorities in Java.

Ans. Multithreading is a technique that allows a program or a process to execute many tasks concurrently (at the same time and parallel). It allows a process to run

its tasks in parallel mode on a single processor system. In the multithreading concept, several multiple lightweight processes are run in a single process/task or program by a single processor. For Example, when you use a word processor you perform a many different tasks such as printing, spell checking and so on. Multithreaded software treats each process as a separate program.

There are two ways to create thread in java:

- **Provide a runnable object:** The runnable interface defines a single method run, meant to contain the code executed in the thread. The runnable object is passed to the Thread constructor, as in the HelloRunnable example.

```
public class HelloRunnable implements Runnable {
    System.out.println ("Hello from a Thread");
}

public static void main (String args []) {
    (new Thread (new HelloRunnable ()).start ();
}
```

- **Subclass Thread:** The thread class itself implements runnable, through its run method does nothing. An application can subclass thread, providing its own implementation of run, as in the HelloThread example.

```
public class HelloThread extends Thread {
    System.out.println ("Hello from a thread");
}

Public static void main (String args []) {
    (new HelloThread ()).start ();
}
```

Refer to Page No.-307 [Thread Priorities]

(b) Write a Java Program which takes two 3×3 matrices as input and find sum of the two matrices? Define constructor for initialising matrix objects.

```
Ans. import java.util.*;
class MatrixAddition{
    public static void main(String[] args) throws Exception {
        Scanner input=new Scanner(System.in);
        int[][] A = new int[2][2];
        int[][] B = new int[2][2];
        int[][] C = new int[2][2];
        System.out.println("Enter elements for matrix A : ");
        for (int j=0 ; j < A[i].length ; j++){
            A[i][j] = input.nextInt();
        }
        System.out.println("Enter elements for matrix B : ");
        for (int i=0 ; i < B.length ; i++)
            for (int j=0 ; j < B[i].length ; j++)
```

```

B[i][j] = input.nextInt();
}
System.out.println("Matrix A:");
for (int i=0 ; i < A.length ; i++)
{
    System.out.println();
    for (int j=0 ; j < A[i].length ; j++){
        System.out.print(A[i][j]+” “);
    }
}
System.out.println();
System.out.println("Matrix B:");
for (int i=0 ; i < B.length ; i++)
{
    System.out.println();
    for (int j=0 ; j < B[i].length ; j++){
        System.out.print(B[i][j]+” “);
    }
}
System.out.println();
System.out.println("Sum of 2 matrices,Matrix C:");
for(int i=0;i<2;i++){
for(int j=0;j<2;j++){
C[i][j]=A[i][j]+B[i][j];
System.out.print(C[i][j]+” “);
}
System.out.println();
}
}
}
}

```

Q3. (a) Write a Java program that reads the contents of a text file and counts the number of white spaces, number of characters, number of words, number of semicolons in the contents. Print the output of program on standard output.

Ans.

```

import java.util.*;
import java.io.*;
public class TextFileInfoPrinter
{
    public static void main(String[]args) throws FileNotFoundException
    {
        Scanner console = new Scanner(System.in);
        System.out.println("File to be read: ");
        String inputFile = console.next();
        File file = new File(inputFile);
        Scanner in = new Scanner(file);
    }
}

```

```

int words = 0;
int whitespace = 0;
int chars = 0;
int semicolon = 0;
while(in.hasNext())
{
    in.next();
    words++;
}
while(in.hasNextByte())
{
    in.nextByte();
    whitespace++;
}
while(in.hasNextByte())
{
    in.nextByte();
    chars++;
}
while(in.hasNextByte())
{
    in.nextByte();
    semicolon++;
}
System.out.println("Number of lines: " + whitespace);
System.out.println("Number of words: " + words);
System.out.println("Number of characters: " + chars);
System.out.println("Number of words: " + semicolon);
}
}

```

Output:

The brown for; jump over the lazy dog;

Whitespace= 8

Words= 8

Char= 29

Semicolon=2

(b) Write a Java program to find factorial of a number.

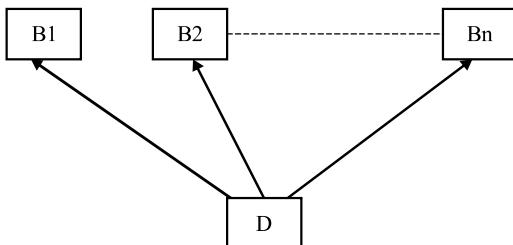
Refer to June-2006, Q.No.-4(b), Page No.-226

(c) What is inheritance? Explain with an example how is multiple inheritance achieved in Java?

Ans. Inheritance is an object oriented feature supported by Java wherein the features of one Java class can be inherited/made available in another class. This creates a

parent child relationship between these 2 classes. Class Inheritance in java mechanism is used to build new classes from existing classes. The inheritance relationship is transitive: if class x extends class y, then a class z, which extends class x, will also inherit from class y. Object-oriented programming allows classes to inherit commonly used state and behavior from other classes.

When a derived class is inherited from several base classes, called multiple inheritance.



B -> Base Class

D -> Derived Class

Example:

Class <derived-class> : <visibility-mode> <base-class1>, <visibility-mode> <base-class2>, _____ <visibility-mode> <base-class N>

{

 derived-class members;

};

Q4. (a) Write a Java program to setup JDBC and execute and SQL statement on a database table ‘BOOK’, with the fields BOOK_ISBN, BOOK_AUTH, BOOK_PRICE, BOOK_PUBL. The SQL statement should find the BOOK_AUTH and BOOK_PRICE of the Book having ISBN = 235001.

Ans. import java.sql.*;

```

class MyJDBC
{
    public static void main (String [] args)
    {
        try
        {
            String url = “jdbc:mysql://localhost/MyDb”;
            Connection conn = DriverManager.getConnection (url,””,””);
            Statement stmt= conn.createStatement ();
            ResultSet stmt = conn.createStatement ();
  
```

```

Rs = stmt.executeQuery ("SELECT BOOK_AUTH, BOOK_PRICE FROM
BOOK WHERE BOOK_ISBN = 235001");
While (rs.next ())
{
String BOOK_AUTH = rs.getString ("BOOK_AUTH");
String BOOK_PRICE = rs.getString ("BOOK_PRICE");
System.out.println ("BOOK_AUTH");
System.out.println ("BOOK_PRICE");
}
Conn.close ();
}
catch ( Exception e)
{
System.err.println ("Got an exception");
System.err.println (e.getMessage ());
}
}
}
}

```

(b) What is Polymorphism? Explain advantage of polymorphism with an example of managing different types of Bank Accounts.

Ans. Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object. Any Java object that can pass more than one IS-A test is considered to be polymorphic. In Java, all Java objects are polymorphic since any object will pass the IS-A test for their own type and for the class Object. It is important to know that the only possible way to access an object is through a reference variable. A reference variable can be of only one type. Once declared, the type of a reference variable cannot be changed.

The reference variable can be reassigned to other objects provided that it is not declared final. The type of the reference variable would determine the methods that it can invoke on the object.

A reference variable can refer to any object of its declared type or any subtype of its declared type. A reference variable can be declared as a class or interface type.

Example:

```

class Account
{
protected double bal;
protected String account_id;
protected String name;
public Account (double amount)
{

```

```
    bal = amount;
}
public Account (String id, String nm)
{
    bal = 0.0;
    account_id = id;
    name = nm;
}
public void deposit (double amount)
{
    bal = bal + amount;
}
public double withdraw ( double amount)
{
    if (bal >=amount)
    {
        bal = bal - amount;
        return amount;
    }
    else
        return 0.0;
}
public double getbal ()
{
    return bal;
}
}
class AccountDemo
{
    public static void main (String args [])
    {
        Account my_account = new Account ("34561234567", "xyz");
        System.out.println ("Account id=" + my_account.account_id);
        System.out.println ("Name=" + my_account.name);
        System.out.println ("initial balance=" + my_account.balance);
        My_account.deposit (25000.00);
        System.out.println ("After Deposit current balance" +
my_account.getbalance ());
        My_account.withdraw (5000.00);
        System.out.println ("After withdraw | Remaining balance" +
my_account.getbalance ());
    }
}
```

Q5. (a) What is Java Bean? Explain its advantages.

Ans. JavaBeans are reusable software components for Java. They are classes that encapsulate many objects into a single object (the bean). They are serialisable, have a 0-argument constructor, and allow access to properties using getter and setter methods.

Advantages are:

- The properties, events, and methods of a bean that are exposed to another application can be controlled.
- A bean may register to receive events from other objects and can generate events that are sent to those other objects.
- Auxiliary software can be provided to help configure a java bean.
- The configuration setting of bean can be saved in a persistent storage and restored.

(b) What is String Class in Java? Write a program to insert a substring in the middle of a string. (Do not use methods of string class)

Ans. String is one of the widely used java classes. The Java String class is used to represent the character strings. All String literals in Java programs are implemented as instance of the String Class. Strings are constants and their values cannot be changed after they created. Java String objects are immutable and they can be shared. The String Class includes several methods to edit the content of string.

using System;

class Program

{ static void Main()

{

//

// A. You can insert one string between two others.

//

string names = "Romeo Juliet";

string shakespeare = names.Insert(6, "and ");

console.WriteLine(shakespeare);

//

// B. You can insert a string right after another string.

//

string names2 = "The Taming of Shrew";

int index2 = names2.IndexOf("of ");

string shakespeare2 = names2.Insert(index2 + "of ".Length, "the ");

console.WriteLine(shakespeare2);

}

}

Output:

Romeo and Juliet

The Taming of the Shrew

(c) What is layout manager? Explain any two layouts in Java briefly.

Ans. Layout managers are software components used in widget toolkits which have the ability to lay out graphical control elements by their relative positions without using distance units. It is often more natural to define component layouts in this manner than to define their position in pixels or common distance units, so a number of popular widget toolkits include this ability by default. Widget toolkits that provide this function can generally be classified into two groups:

- Those where the layout behavior is coded in special graphic containers. This is the case in XUL and the .NET Framework widget toolkit (both in Windows Forms and in XAML).
- Those where the layout behavior is coded in layout managers, that can be applied to any graphic container. This is the case in the Swing widget toolkit that is part of the Java API.

Java technology provides the following Layout Managers, each of which implements the Layout Manager interface.

- Flow Layout
- Grid Layout
- Border Layout
- GridBag Layout
- Card Layout

Flow Layout Manager: Flow Layout places component in rows from left to right. Components towards the end of row are written on next row, if there is not enough space in the current row. The Flow Layout honors the specified size of a component. The size of a component never changes regardless of size of container.

Grid Layout Manager: A Grid Layout Manager places the components in a rectangular grid. Each component's position is identified by a column and row. All the cells in the grid have same size and width. Each component is stretched to the cell size

(d) Differentiate between overloading and overriding with the help of suitable example.

Refer to June-2007, Q.No.-1(b), Page No.-263

Intensify Your Child's Learning with our special book series

Visit : EnhanceYourChild.com

Note: Q. No. 1 is compulsory. Attempt any three questions from the rest. Attempt all parts of questions together.

Q1. (a) Define the concept of inheritance. Explain the advantage of inheritance with the help of an example.

(b) What is classpath? Explain the use of classpath in Java programming.

(c) What is multithreading? How is thread synchronization implemented in Java? Explain with an example.

(d) What is an event in GUI programming? Explain different components of an event.

(e) What is inner class? Explain whether inner class is secure and useful. Justify your answer.

(f) What is final keyword? Explain different uses of final in Java.

(g) What is exception? Explain difference between checked and unchecked exceptions with an example of each.

Q2. (a) Differentiate the following, with the help of example:

(i) Application and Applet

(ii) Structure Approach and Object Oriented Approach

(iii) String and String Buffer

(iv) Abstract Class and Interface

(b) Write a class complex to represent complex numbers, with suitable constructor and function to find the sum of two complex numbers.

Q3. (a) What is StreamTokenizer? Explain the different instance variables defined in StreamTokenizer. Also, explain use of StreamTokenizer with the help of an example.

(b) "Java is architecture neutral, secure and distributed programming language." Justify the statement.

(c) What is private access specifier? Explain with an example, how it is different from public and protected access specifiers.

Q4. (a) What is finalize() method in Java. Give an example to demonstrate its use.

(b) Define package in Java. Explain the need of 'user created package' in Java. Also, explain the procedure of creating a package in Java.

(c) What is the difference between throw and throws keywords used in Java? Explain with an example.

(d) Explain use of super keyword in Java, with an example.

Q5. (a) What is stateless protocol? Is HTTP a stateless protocol? What are the various ways through which you can maintain the 'State' while using HTTP?

(b) Differentiate between TCP client and TCP server socket with the help of an example.

(c) What is RMI? Explain advantages of using RMI.

(d) Briefly explain the life cycle of a servlet.

Note: Q. No. 1 is compulsory. Attempt any three questions from the rest.

Q1. (a) What is Class? How does it accomplish Data Hiding? Explain with an example.

(b) State any three differences between AWT and Swing components.

(c) Explain the concept of Inheritance and Polymorphism with an example of each.

(d) What do you mean by serialization? What is the common usage of serialization?

(e) What are cookies and session variables and where do we use them?

(f) Write a Java program to read text from keyboard and write it to a file called ‘MyText.txt’.

(g) Explain any five basic features of Java in brief.

Q2. (a) Differentiate between the following terms with examples:

(i) Structured Programming and Object Oriented Programming

(ii) Data Abstraction and Encapsulation

(iii) length and length()

(iv) Class and Object

(b) What are the two methods to create threads? Explain them with a suitable example.

Q3. (a) What is RMI? How can you create stub and skeleton? Write the steps involved.

(b) Write a program to read the content of the file ‘MyDoc.doc’ and display it on console.

(c) What is exception? Explain the basic causes of exception. Write a Java program to explain how exceptions are handled.

Q4. (a) Write a Java program to create a Shape class. Derive Circle and Square classes from it. Define constructor for these three classes.

(b) Explain the different steps in the life cycle of an applet.

(c) What is a package? Develop a Java package with simple queue and stack class.

Q5. (a) What is Servlet? Explain how session handling is performed in Servlet programming.

(b) What is an Interface? Write a Java program to show how a class implements two interfaces.

(c) What is String class? How is it different from String Buffer class? Write a Java program to find the length of a given string?

Note: Q. No. 1 is compulsory. Attempt any three questions from the rest.

- Q1.** (a) Explain the basic features of Object Oriented Programming Language.
(b) Why is Java called Machine Independent Language? Explain the functionality of JVM.
(c) What is the difference between constructor and method? Explain with an example.
(d) Explain the uses of keywords final, finally and finalize in Java.
(e) What is an abstract class? Explain the use of abstract class with an example.
(f) What are the different boolean and bitwise operators in Java? Explain in brief.
(g) Write a Java Applet to display “Java is an Object Oriented Programming Language”; in red color.

Q2. (a) Distinguish between the following terms with suitable example(s):

- (i) Method Overloading and Overriding
(ii) Application and Applet
(iii) Get Method and Post Method in Servlet.
(iv) Readers/Writers and I/O Streams.
(b) What is listener? Write a program to implement mouse motion listener.

Q3. (a) What is encapsulation? Explain its advantage with suitable example.

(b) What will be the output of the following program:

```
Class t extends Thread
{
    public void run ()
    {
        System.out.println ("start");
        yield ();
        resume ();
        System.out.println ("restart");
        stop ();
        resume ();
        System.out.println ("Nothing");
    }
    public static void main (String a [ ])
    {
        t1 = new t ();
        t1.start ();
    }
}
```

(c) Explain the output stream class hierarchy in Java.

(d) What is layout manager? Explain the Flow Layout and Border Layout. Also explain how a layout is set.

Q4. (a) What is a JavaBean? Discuss its features in brief.

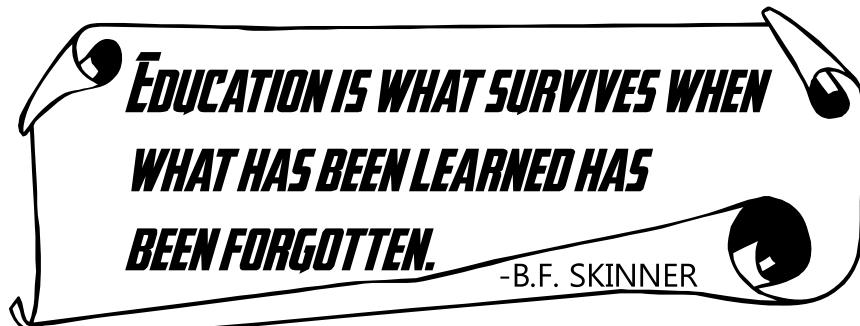
(b) What is checked exception? How is it different from unchecked exception? Explain with an example.

(c) What is inheritance? Create a class Train and inherit class Superfast Train from it, define proper constructor for both of the classes.

Q5. (a) Write a program in Java to implement socket programming using DataGram Class.

(b) How do we design a package in java? What are the steps to add classes and interfaces in a package?

(c) What is synchronization? Explain how methods are synchronized in Java, with the help of an example.



***EDUCATION IS WHAT SURVIVES WHEN
WHAT HAS BEEN LEARNED HAS
BEEN FORGOTTEN.***

-B.F. SKINNER

Note: Q. No. 1 is compulsory. Attempt any three questions from the rest.

Q1. (a) Explain the relationship between Data Abstraction and Encapsulation.

Refer to Chapter-1, Q.No.-7, Page No.-9 & Q.No.-1, Page No.-1

(b) Explain the need of main() function in writing a Java Program using an example program.

Ans. Every Java program requires a main () function in one of the program's class. The main () function is the starting point of the program. You can supply arguments to main () function. If you have runtime arguments in your program, supply them in the Arguments to Main text field. Only programs that run from the command line use runtime arguments so you usually don't have to worry about the Arguments to Main text field.

```
import java.util.Date;  
class DateApp {  
    public static void main(String args[]) {  
        Date today = new Date();  
        System.out.println(today);  
    }  
}
```

A Java application (like DateApp in the code listed above) must contain a main() method whose signature looks like this

```
public static void main(String args[])
```

The method signature for the main() method contains three modifiers:

- **public** indicates that the main() method can be called by any object.
- **static** indicates that the main() method is a static method (also known as class method).
- **void** indicates that the main() method has no return value.

The main() method in the Java language is similar to the main() function in C and C++. When you execute a C or C++ program, the runtime system starts your program by calling its main() function first. The main() function then calls all the other functions required to run your program. Similarly, in the Java language, when you execute a class with the Java interpreter, the runtime system starts by calling the class's main() method. The main() method then calls all the other methods required to run your application.

(c) Explain function overloading using an example.

Refer to Chapter-5, Q.No.-3, Page No.-67

(d) Explain Java Thread Model using Thread Life Cycle.

Ans. The benefit of Java's multithreading is that a thread can pause without stopping other parts of your program. A paused thread can restart. A thread will be referred to as dead when the processing of the thread is completed. After a thread reaches the dead state, then it is not possible to restart it.

The thread exists as an object; threads have several well-defined states in addition to the dead states. These state are:

Ready State: When a thread is created, it doesn't begin executing immediately. You must first invoke start () method to start the execution of the thread. In this process the thread scheduler must allocate CPU time to the Thread. A thread may also enter the ready state if it was stopped for a while and is ready to resume its execution.

Running State: Threads are born to run, and a thread is said to be in the running state when it is actually executing. It may leave this state for a number of reasons, which we will discuss in the next section of this unit.

Waiting State: A running thread may perform a number of actions that will cause it to wait. A common example is when the thread performs some type of input or output operations.

(e) What is the usage of literals in a programming language? Explain the usage of any six Java literals used in Java programming, with examples.

Ans. Literals are nothing but pieces of Java code that indicate explicit values. For example "Hello IGNOU!" is a String literal. The double quote marks indicate to the compiler that this is a string literal. The quotes indicate the start and the end of the string, but remember that the quote marks themselves are not a part of the string. Similarly, Character Literals are enclosed in single quotes and it must have exactly one character. TRUE and FALSE are boolean literals that mean true and false. Number, double, long and float literals also exists there. All types of literals used in Java Programming are given in table:

Examples of literals

Types of literal	Example
Number Literals	-45, 4L, 0777, 0xFF, 2.56F, 10e45, 36E-2
Boolean Literals	TRUE, FALSE
Character Literals	'a', '#', '3', '\n', '\\', '\"'
String Literals	"A string with a \t tab in it"
Double Literals	1.5, 45.6, 76.4E8
Long Literals	34L
Float Literals	45.6f, 76.4E8F, 1.5F

(f) Explain the usage of Flow Layout and Card Layout in Java Applet with diagram.

Refer to June-2009, Q.No.-4(a), Page No.-298

(g) Explain the usage of container in Java Applet and Servlet.

Refer to Chapter-9, Page No.-115[Containers]

Q2. (a) Explain the usage of reserved ports in net programming for FTP, TCP, etc.

Ans. There are some port numbers, which are reserved for specific purposes on any computer working as a server and connected to the Internet. Most standard applications and protocols use reserved port numbers, such as email, FTP, and HTTP.

When you want two programs to talk to each other across the Internet, you have to find a way to initiate the connection. So at least one of the ‘partners’ in the conversation has to know where to find the other one or in other words the address of other one. This can be done by address (IP number + port number) of the one side to the other.

However, a problem could arise if this address must not be taken over by any other program. In order to avoid this, there are some port numbers, which are reserved for specific purposes on any computer connected to the Internet. Such ports are reserved for programs such as ‘Telnet’, ‘Ftp’ and others. For example, Telnet uses port 23, and FTP uses port 21. Note that for each kind of service, not only a port number is given, but also a protocol name (usually TCP or UDP).

Two services may use the same port number, provided that they use different protocols. This is possible due to the fact that different protocols have different address spaces: port 23 of a one machine in the TCP protocol address space is not equivalent to port 23 on the same machine, in the UDP protocol address space.

(b) Explain all steps of using arrays in a Java program.

Ans. The steps of using array in a Java Program are discussed as follows:

(1) Declaring a Variable to refer to an Array

The following line of code from the sample program declares an array variable:

Int [] an Array; // declare an array of integers

An array declaration has two components:

- Array’s type
- Array’s name.

An array’s type is written type [], where type is the data type of the elements contained within the array, and [] indicates that this is an array. The example program uses int [] and name of the array is an Array. Here, an Array will be used to hold integer data. Here are some other declarations for arrays that hold other types of data:

float[] marks;

```
boolean [] gender;
Object[] listOfObjects;
String[] NameOfStudents;
```

The declaration for an array variable does not allocate any memory to contain the array elements. The example program must assign a value to an Array before the name refers to an array.

(2) Creating an Array: The programmer creates an array explicitly using Java's new operator. The next statement in the example program allocates an array with enough memory for ten integer elements and assigns the array to the variable an Array declared earlier.

```
An Array = new int[10]; // create an array of integers
```

In general, when creating an array, the programmer uses the new operator, followed by data type of the array elements, and then followed by the number of elements desired enclosed within square brackets ('[' and ']') like:

new element Type [array Size]

If the new statement were omitted from the example program, the compiler would print an error and compilation would fail.

(3) Accessing an Array Element: Let us consider the following code, which assigns values to the array elements:

```
for (int i = 0; i < anArray.length; i++)
{
    anArray[i] = i;
    System.out.print (anArray[i] + " ");
}
```

It shows that to reference an array element, append square brackets to the array name. The value between the square brackets indicates (either with a variable or some other expression) the index of the element to access. As mentioned earlier in Java, array indices begin at 0 and end at the array length minus 1.

(4) Getting the Size of an Array: You can get the size of an array, by writing "arrayname.length". Here, length is not a method, it is a property provided by the Java platform for all arrays. The "for" loop in our example program iterates over each element of an Array, assigning values to its elements. The "for" loop uses the anArray.length to determine, when to terminate the "for" loop.

(5) Array Initializers: The Java programming language provides one another way also for creating and initializing an array. Here is an example of this syntax:

```
boolean [] answers = {true, false, true, true, false};
```

The number of values provided between {and} determines the length of the array.

(6) Multidimensional array in Java: In Java also you can take multidimensional arrays as arrays of arrays. You can declare a multidimensional array variable by specifying each additional index with a set of square brackets. For example

```
float two Dim [][]= new float [2][3];
```

This declaration will allocate a 2 by 3 array of float to two Dim. Java also provides the facility to specify the size of the remaining dimensions separately except the first dimension.

(c) Write a program for reading input from the console using StringBuffer ().

Ans. //program

```
import Java.io.*;
class ConsoleInput
{
    StringBuffer response = new StringBuffer();
    try
    {
        BufferedInputStream buff = new BufferedInputStream(System.in);
        int in = 0;
        char inChar;
        do
        {
            in = buff.read();
            inChar = (char) in;
            if (in != -1)
            {
                response.append(inChar);
            }
        } while ((in != 1) & (inChar != '\n'));
        buff.close();
        return response.toString();
    }
    catch (IOException e)
    {
        System.out.println("Exception: " + e.getMessage());
        return null;
    }
}
public static void main(String[] arguments)
{
    System.out.print("\nWhat is your name? ");
    String input = ConsoleInput.readLine();
    System.out.println("\nHello, " + input);
}
```

Output:

```
C:\JAVA\BIN>Java ConsoleInput
What is your name? Java Tutorial
Hello, Java Tutorial
```

Q3. (a) Explain all the components used for handling an event in a Java program.

Refer to December-2005, Q.No.-1(h) , Page.No.-204

(b) What are Java beans? Explain their special features.

Refer to Chapter-11, Q.No.-5, Page No.-142 & Q.No.-7, Page No.-143

(c) Write a Java program to show inter thread communication.

Refer to June-2009, Q.No.-3(b), Page No.-297

Q4. (a) Explain method overriding using a program.

Refer to Chapter-5, Q.No.-5, Page No.-70

(b) How does an index method help in string operations?

Ans. String class provides methods for handling of String objects. String class methods can be grouped in index methods, value Of () methods and sub-string methods. Methods of these groups are discussed below:

Index Methods

Methods	Return
public int length() public int indexOf(int ch)	Return the length of string. Return location of first occurrence of ch in string, if ch don't exist in string return -1.
public int indexOf(int ch, int from Index)	Return location of occurrence of ch in string after from Index, if ch don't exist in string return -1.
public int lastIndexOf(int ch)	Return location of last occurrence of ch in string, if ch location does not exist in string return -1.
public int lastIndexOf(int ch, int from Index)	Return last of occurrence of ch in string after location from Index, if ch does not exist in string after location from Index return -1.
public int indexOf(String str)	Return location of first occurrence of substring str in string, if str does not exist in string return -1.
public int indexOf(String str, int from Index)	Return location of first occurrence of substring str in after location from Index string, if str does not exist in string return -1.
public int lastIndexOf(String str)	Return location of last occurrence of substring str in string, if str does not exist in string return -1.
public int lastIndexOf(String str, int from Index)	Return location of last occurrence of substring str in after location from Index string, if str does not exist in string return -1.

We can see in the example program given below, in which the index methods are used. This program will help you to know the use of index methods.

```
public class Index_Methods_Demo
{
    public static void main(String[] args)
    {
        String str = "This is a test string";
        System.out.println("The length of str is :" + str.length());
        System.out.println("Index of 't' in str is:" + str.indexOf('t'));
        // Print first occurrence of character t in string
        System.out.println("First occurrence of 't' after 13 characters in the str:" + str.indexOf('t',12));
        // Print first occurrence of character t in string after first 13 characters
        System.out.println("Last occurrence of 'z' in the str:" + str.lastIndexOf('z'));
        // Print Last occurrence of character z in string : See output and tell
        // what is printed because 'z' is not a character in str.
        System.out.println("First occurrence of substring :is substring of string str:" + str.indexOf("is"));
        // Print first occurrence of substring "is" in str
        System.out.println("Last occurrence of substring :ing in the str:" + str.lastIndexOf("ing"));
        // Print first occurrence of substring "ing" in string str
        System.out.println("First occurrence of substring :this after 11 characters in the str:" + str.indexOf("this",10));
        // Print first occurrence of substring "this" after first 11 characters in string str
    }
}
```

Output:

The length of str is :21

Index of 't' in str is:10

First occurrence of 't' after 13 characters in the str:13

Last occurrence of 'z' in the str:-1

First occurrence of substring :is substring of string str:2

Last occurrence of substring :ing in the str:18

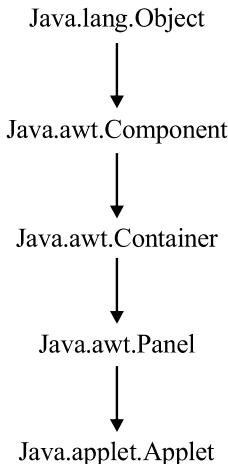
First occurrence of substring :this after 11 characters in the str:-1

In the output of the above given program you can see that if a character or substring does not exist in the given string then methods are returning: -1.

Now let us see some substring methods, comparisons methods, and string modifying methods provided in string class. These methods are used to find location of a character in string, get substrings, concatenation, comparison of strings, string reasons matching, case conversion etc.

(c) Write the steps of incorporating Applet in a web page.

Ans. Java is an Object Oriented Programming language, supported by various classes. The Applet class is packed in the Java. Applet package which has several interfaces. These interfaces enable the creation of Applets, interaction of Applets with the browser, and playing audio clips in Applets. In Java 2, class Javax.swing.JApplet is used to define an Applet that uses the Swing GUI components. In Java class hierarchy Object is the base class of Java.lang package. The Applet is placed into; the hierarchy as follows:



Now, the dos and donts of using Java Applet are discussed as follows:

Now Refer to Chapter-9, Q.No.-2, Page No.-124

Q5. (a) Explain the usage of Stream Tokenizer class.

Refer to June-2008, Q.No.-3(b), Page No.-283

(b) Write a Java program to convert all the strings of input to upper case.

The input string should be submitted using a textbox in a form of web page.

Ans. //Program to test whether the content of a Character object is Upper or Lower

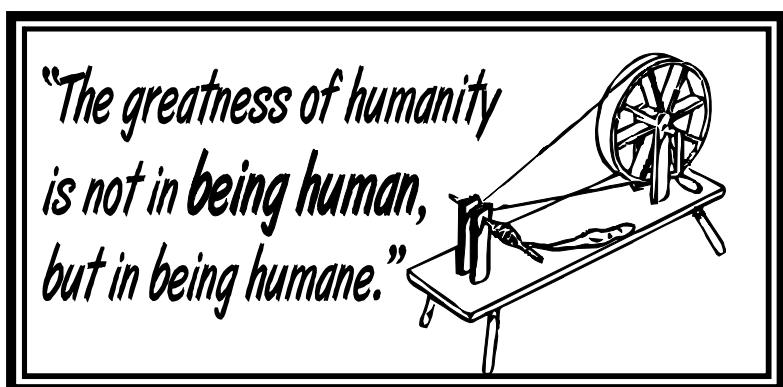
```

public class CharTest
{
    public static void main(String args[])
    {
        Character MyChar1 = new Character('i');
        Character MyChar2 = new Character('J');
        //Test for MyChar1
        if (MyChar1.isUpperCase(MyChar1.charValue()))
            System.out.println("MyChar1 is in Upper Case: "+MyChar1);
        else
            System.out.println("MyChar1 is in Lower Case: "+MyChar1);
    }
}
  
```

```
// Test for MyChar2
if (MyChar2.isUpperCase(MyChar2.charValue()))
System.out.println("MyChar2 is in Upper Case: "+MyChar2);
else
System.out.println("MyChar2 is in Lower Case: "+MyChar2);
}
}
Output:
MyChar1 is in Lower Case: i
MyChar2 is in Upper Case: J
```

(c) Write the steps of JDBC in establishing a connection for creating dynamic website.

Refer to June-2006, Q.No.-4(c), Page No.-226



Note: Q. No. 1 is compulsory. Attempt any three questions from the rest.

Q1. (a) Explain the relationship between inheritance and polymorphism.

Ans. Relationship between inheritance and polymorphism

(1) Inheritance defines father-son relationship between two classes, while Polymorphism takes advantage of that relationship to add dynamic behaviour in code.

(2) Inheritance is meant for code reuse, initial idea is to reuse what is written inside Parent class and only write code for new function or behaviour in Child class. On the other hand Polymorphism allows Child to redefine already defined behaviour inside parent class. Without Polymorphism it's not possible for a Child to execute its own behaviour while represented by a Parent reference variable, but with Polymorphism he can do that.

(3) Polymorphism helps tremendously during maintenance. In fact many object oriented design principles are based on Polymorphism e.g. programming for interface then implementation, which advocates to use interface everywhere in our code, to represent variable, in method parameters, in return type of method etc; so that code can take advantage of polymorphism and do more than what was expected it to do during writing.

(4) Java doesn't allow multiple inheritance of classes, but allows multiple inheritance of Interface, which is actually required to implement Polymorphism. For example a Class can be Runnable, Comparator and Serializable at same time, because all three are interfaces. This makes them to pass around in code e.g. we can pass instance of this class to a method which accepts Serializable, or to Collections.sort() which accepts a Comparator.

(b) Explain why main() is not included in writing a Java applet program using an applet coding.

Ans. An applet is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal. There are some important differences between an applet and a standalone Java application, including the following:

(1) An applet is a Java class that extends the java.applet.Applet class.

(2) A main() method is not invoked on an applet, and an applet class will not define main().

(3) Applets are designed to be embedded within an HTML page.

(4) When a user views an HTML page that contains an applet, the code for the applet is downloaded to the user's machine.

- (5) A JVM is required to view an applet. The JVM can be either a plug-in of the Web browser or a separate runtime environment.
- (6) The JVM on the user's machine creates an instance of the applet class and invokes various methods during the applet's lifetime.
- (7) Applets have strict security rules that are enforced by the Web browser. The security of an applet is often referred to as sandbox security, comparing the applet to a child playing in a sandbox with various rules that must be followed.
- (8) Other classes that the applet needs can be downloaded in a single Java Archive (JAR) file.

(c) Explain method overloading using an example.

Refer to Chapter-5, Q.NO.-3

(d) Explain the process of inter-thread communication.

Refer to June-2009, Q.No.-3(b) Page No.-297

(e) Explain the usage of three assignment operators and three bitwise operators in Java programming.

Refer to Chapter-3, Q.No.-6

(f) Explain the usage of GridLayout and GridBagLayout in a Java Applet with diagram.

Ans. A GridLayout divides an applet into a specified number of rows and columns, which form a grid of cells, each equally sized and spaced. It is important to note that each is equally sized and spaced as there is another similar named Layout known as GridBagLayout. As Components are added to the layout they are placed in the cells, starting at the upper left hand corner and moving to the right and down the page. Each component is sized to fit into its cell. This tends to squeeze and stretch components unnecessarily.

We will find the GridLayout is great for arranging Panels. A GridLayout specifies the number of rows and columns into which components will be placed. The applet is broken up into a table of equal sized cells. GridLayout is useful when we want to place a number of similarly sized objects. It is great for putting together lists of checkboxes and radio buttons as we did in the Ingredients applet.

Grid Bag Layout

GridBagLayout is the most precise of the five AWT Layout Managers. It is similar to the GridLayout, but components do not need to be of the same size. Each component can occupy one or more cells of the layout. Furthermore, components are not necessarily placed in the cells beginning at the upper left-hand corner and moving to the right and down.

In simple applets with just a few components we often need only one layout manager. In more complicated applets, however, we will often split our applet into panels, lay out the panels according to a layout manager, and give each panel its own layout manager that arranges the components inside it.

The GridBagLayout constructor is trivial, GridBagLayout() with no arguments.

```
GridBagLayout gbl = new GridBagLayout();
```

Unlike the GridLayout() constructor, this does not say how many rows or columns there will be. The cells our program refers to determine this. If we put a component in row 8 and column 2, then Java will make sure there are at least nine rows and three columns. (Rows and columns start counting at zero.) If we later put a component in row 10 and column 4, Java will add the necessary extra rows and columns. We may have a picture in our mind of the finished grid, but Java does not need to know this when you create a GridBagLayout.

(g) Which types of components are used in designing swing based GUI?

Refer to Chapter-9, Q.NO.-9

Q2. (a) Explain the usage of anonymous proxy servers in designing mailing application.

Ans. Anonymous proxy servers hide our IP address and thereby prevent our data from unauthorised access to our computer through the Internet. They do not provide anyone with our IP address and effectively hide any information about us. Besides that, they don't even let anyone know that we are surfing through a proxy server. Anonymous proxy servers can be used for all kinds of Web-services, such as Web-Mail (MSN Hot Mail, Yahoo mail), web-chat rooms, FTP archives, etc. ProxySite.com will provide a huge list of public proxies. Any web resource we access can gather personal information about us through our unique IP address – our ID in the Internet. They can monitor our reading interests, spy upon us, and according to some policies of the Internet resources, deny accessing any information we might need. We might become a target for many marketers and advertising agencies that, having information about our interests and knowing our IP address as well as our e-mail. They will be able to send us regularly their spam and junk e-mails.

(b) Explain the steps of using multidimensional arrays in a Java program.

Refer to Chapter-3, Q.No.-12

To understand the concept see the program given below in which different second dimension size is allocated manually:

```
class Arra_VSize
{
    public static void main (String args[])
}
```

```
{  
int i, j, k=0;  
int twoDim [][] = new int [3][];  
twoDim[0] = new int[1];  
twoDim[1] = new int[2];  
twoDim[2] = new int[3];  
for ( i=0; i < 3 ; i++)  
{  
    for (j = 0; j < i+1; j++)  
    {  
        twoDim[i][j] = k + k*3;  
        k++;  
    }  
}  
for ( i=0; i < 3 ; i++)  
{  
    for (j = 0; j < i+1; j++)  
        System.out.print(twoDim[i][j] + " ");  
    System.out.println();  
}  
}  
}  
}  
}  
Output of this program  
0  
48  
12 16 20
```

(c) Write a Java program for “writing output on console” using PrintWriter() method.

Refer to Chapter-7, Q.NO.-11 Page No.101

Q3. (a) Describe all the steps used in writing a Java program for handling events.

Ans. In life, we encounter events that force us to suspend other activities and respond to them immediately. In Java, events represent all actions that go on between the user and the application. Java’s Abstract Windowing Toolkit (AWT) communicates these actions to the programs using events. When the user interacts with a program let us say by clicking a command button, the system creates an event representing the action and delegates it to the event-handling code within the program. This code determines how to handle the event so the user gets the appropriate response.

Originally, JDK 1.0.2 applications handled events using an inheritance model. A

container sub class inherits the action () and handle Event () methods of its parent and handled the events for all components it contained. For instance, the following code would represent the event handler for a panel containing an OK and a Cancel button:

```
public boolean handleEvent(Java.awt.Event e)
{
if(e.id==Java.awt.Event.ACTION_EVENT)
{
if(e.target==buttonOK)
{
buttonOKPressed();
}
else if(e.target==buttonCancel)
(
buttonCancelPressed();
)
}
return super.handleEvent(e);
}
```

The problem with this method is that events cannot be delivered to specific objects. Instead, they have to be routed through a universal handler, which increases complexity and therefore, weakens our design.

But Java 1.1 onward has introduced the concepts of the event delegation model. This model allows special classes, known as “adapter classes” to be built and be registered with a component in order to handle certain events. Three simple steps are required to use this model:

- (1) Implement the desired listener interface in your adapter class. Depending on what event we’re handling, a number of listener interfaces are available. These include: ActionListener, WindowListener, MouseListener, MouseMotionListener, ComponentListener, FocusListener, and ListSelectionListener.

- (2) Register the adapter listener with the desired component(s). This can be in the form of an add XXX Listener () method supported by the component for example include add ActionListener (), add MouseListener (), and add FocusListener ().

- (3) Implement the listener interface’s methods in our adapter class. It is in this code that you will actually handle the event.

(b) What is the usage of RMI in calling methods from remote class?

Refer to Chapter-11, Q.NO.-1 Page No.-137

(c) Write a Java program to show interprocess synchronization.

Ans. If we want two threads to communicate and share a complicated data structure, such as a linked list or graph, we need some way to ensure that they don’t

conflict with each other. In other words, we must prevent one thread from writing data while another thread is in the middle of reading it. For this purpose, Java implements model of interprocess synchronizations.

Following is the example which explains how synchronized methods and object locks are used to coordinate access to a common object by multiple threads.

```
//program
class Call_Test
{
    synchronized void callme (String msg)
    {
        //This prevents other threads from entering call( ) while another thread is using it.
        System.out.print ("["+msg);
        try
        {
            Thread.sleep (2000);
        }
        catch ( InterruptedException e)
        {
            System.out.println ("Thread is Interrupted");
        }
        System.out.println ("]");
    }
}
class Call implements Runnable
{
    String msg;
    Call_Test ob1;
    Thread t;
    public Call (Call_Test tar, String s)
    {
        System.out.println("Inside caller method");
        ob1= tar;
        msg = s;
        t = new Thread(this);
        t.start();
    }
    public void run()
    {
        ob1.callme(msg);
    }
}
```

```
class Synchro_Test
{
    public static void main (String args [ ])
    {
        Call_Test T= new Call_Test( );
        Call ob1= new Call (T, "Hi");
        Call ob2= new Call (T, "This ");
        Call ob3= new Call (T, "is");
        Call ob4= new Call (T, "Synchronization");
        Call ob5= new Call (T, "Testing");
        try
        {
            ob1.t.join();
            ob2.t.join();
            ob3.t.join();
            ob4.t.join();
            ob5.t.join();
        }
        catch ( InterruptedException e)
        {
            System.out.println (" Interrupted");
        }
    }
}
```

Output:

```
Inside caller method
[Hi]
[This ]
[is]
[Synchronization]
[Testing]
```

If we run this program after removing synchronized keyword, we will find some output similar to:

```
Inside caller method
Inside caller method
Inside caller method
Inside caller method
```

```
[Hi[This [isInside caller method
[Synchronization[Testing]
]
]
]
]
```

Q4. (a) Explain multithreading using a program.

Refer to Dec-2005, Q.No.-4(c) Page No.-212

(b) Explain the usage of lastIndexOf (int ch, int fromIndex) method in string operation.**Ans. Methods**

```
public int lastIndexOf(int ch, int fromIndex)
```

Return

Return last of occurrence of ch in string after location fromIndex, if ch does not exist in string after location fromIndex return -1.

Now Refer to Chapter-7, Q.No.-15 Page No.-103

(c) Write a program for passing parameters to an Applet using a web page.

Refer to June-2016, Q.No.-4(c) Page No.-394

Q5. (a) Explain the term BufferedStream using a Java program.

Refer to June-2014, Q.No.-1(e) Page No.-372

(b) Write a Java program for writing files in “C:\javafiles\” location.

Ans. The File class is not I/O. It provides just an identifier of files and directories. Files and directories are accessed and manipulated through java.io.File class. So, we always remember that creating an instance of the File class does not create a file in the operating system. The object of the File class can be created using the following types of File constructors:

```
File MyFile = new File("c:/Java/file_Name.txt");
File MyFile = new File("c:/Java", "file_Name.txt");
File MyFile = new File("Java", "file_Name.txt");
```

The first type of constructor accepts only one string parameter, which includes file path and file name, here in given format the file name is file_Name.txt and its path is c:/Java.

In the second type, the first parameter specifies directory path and the second parameter denotes the file name. Finally in the third constructor the first parameter is only the directory name and the other is file name.

Following is the program for creating a file reference:

```
//program
import java.io.File;
class FileDemo
{
    public static void main(String args[])
    {
        File f1 = new File (“/testfile.txt”);
        System.out.println(“File name : “ + f1.getName());
        System.out.println(“Path : “ + f1.getPath());
        System.out.println(“Absolute Path : “ + f1.getAbsolutePath());
        System.out.println(f1.exists() ? “Exists” : “doesnot exist”);
        System.out.println( f1.canWrite()?”Is writable” : “Is not writable”);
        System.out.println(f1.canRead() ? “Is readable” : “Is not readable”);
        System.out.println(“File Size : “ + f1.length() + “ bytes”);
    }
}
```

Output: (When testfile.txt does not exist)

File name: testfile.txt
Path: \testfile.txt
Absolute Path: C:\testfile.txt
doesnot exist

Is not writable

Is not readable

File Size: 0 bytes

Output: (When testfile.txt exists)

File name : testfile.txt
Path : \testfile.txt
Absolute Path : C:\testfile.txt
Exists
Is writable
Is readable
File Size: 17 bytes

(c) Explain the steps of JDBC in establishing a connection for creating a dynamic website for INSERT/UPDATE the attendance record for employees of an organization.

Refer to June-2006, Q.No.-4(c)

Note: Q. No. 1 is compulsory. Attempt any three questions from the rest.

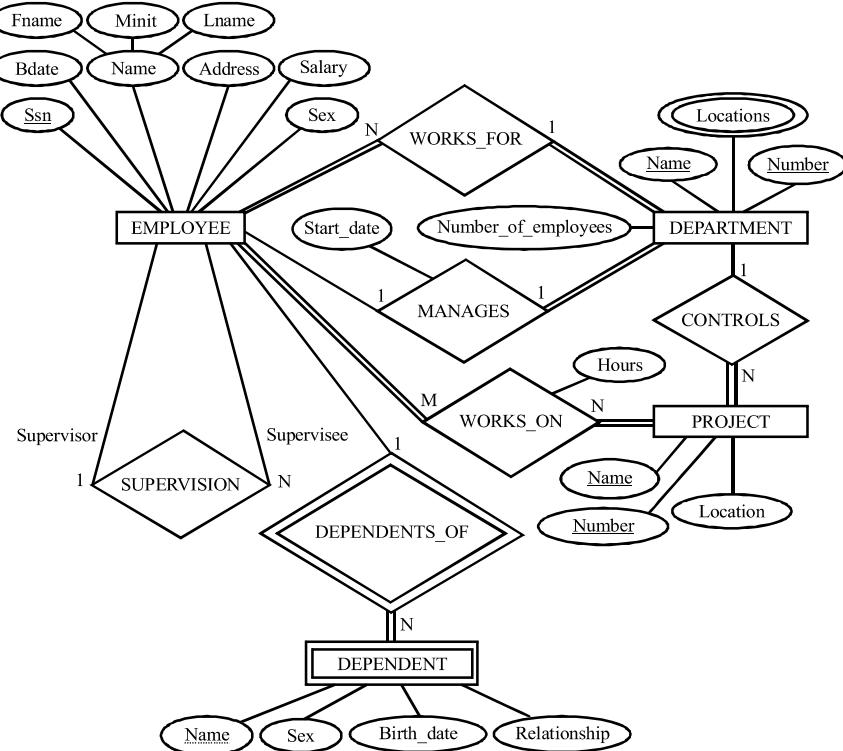
Q1. (a) An employee works in a particular department of an organisation. Every employee has an employee number, name and draws a particular salary. Every department has a name and a head of department. The head of department is an employee. Every year a new head of department takes over. Also, every year an employee is given an annual salary enhancement. Identify and design the classes for the above description with suitable instance variables and methods. The classes should be such that they implement information hiding. You must give logic in support of your design. Also create two objects of each class.

Ans. The company is organised into departments. Each department has a unique name, a unique number, and a particular employee who manages the department. We keep track of the start date when that employee began managing the department. A department may have several locations.

A department controls a number of projects, each of which has a unique name, a unique number, and a single location.

We store each employee's name, Social Security number, address, salary, sex (gender), and birth date. An employee is assigned to one department, but may work on several projects, which are not necessarily controlled by the same department. We keep track of the current number of hours per week that an employee works on each project. We also keep track of the direct supervisor of each employee (who is another employee).

We want to keep track of the dependents of each employee for insurance purposes. We keep each dependent's first name, sex, birth date, and relationship to the employee. The diagrammatic notation is introduced gradually throughout this chapter and is summarised in Figure.



An ER schema diagram for the COMPANY database.

Entities and their Attributes

Entity; Noun, thing, or object – may be physical or conceptual.

Attribute; Adjective, property, field, member – associated with a domain (type).

Composite; An assembly of sub-attributes, like a struct. May be nested.

Simple (Atomic); Not composite.

Single-valued; One entity has one value for this attribute.

Multi-valued; An entity may have more than one value.

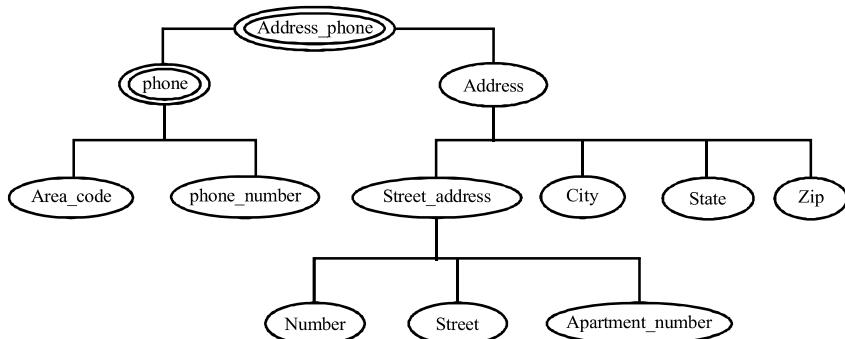
Stored; Fundamental attribute that must be stored some way.

Derived; Its value can be calculated from other attributes.

Nullable; Can be composite and/or multivalued at multiple levels. Look at it in diagram form, and in text form:

Complex

```
{Address_phone({Phone(Area_code,Phone_Number},
Address(Street_address(Number, Street, Apartment_number), City, State,
Zip))}
```



As a relational model, here expressed with SQL.

```

create table Employee (Name Text, Address Text, Age Int, Home_phone Text);
insert into Employee values ('John Smith', "2311 Kirby\nHouston, Texas 77001", 55, '713-749-2630');
  
```

As an object in an object-oriented programming language, expressed here as Ruby.

```

Employee = Struct.new(:Name, :Address, :Age, :Home_Phone)
  
```

```

employees = [Employee.new('John Smith', "2311 Kirby\nHouston, Texas 77001", 55, '713-749-2630')]
  
```

As a record in a text-based data serialisation format, expressed here as [YAML](#).

employees:

- Name: John Smith

Address: |-

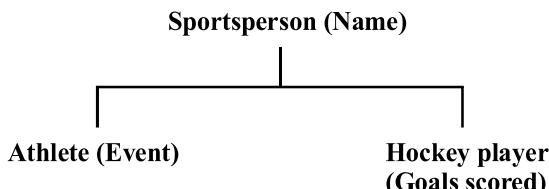
 2311 Kirby

 Houston, Texas 77001

Age: 55

Home_phone: '713-749-2630'

(b) Consider the following class hierarchy:



In this hierarchy, you can assume that a sportsperson can either be an athlete or a hockey player. Every sportsperson has a unique name. An athlete is characterised by the event in which he/she participates; whereas a hockey player is characterised by the number of goals scored by him/her. Perform the following tasks using Java:

(i) Create the class hierarchy with suitable instance variables and methods.

Ans. Refer to Gullybaba.com “download section”

(ii) Create a suitable constructor for each class.

Ans. return _instance != null && _instance.isInitialized()
}

```
// this is default C class constructor
// but subclass can have its own multiple version
// (regardless modifiers)
// and we need to find the best one
// which fits to passed arg to init method
protected C(IA2 a, IB) {
}

public static <IA extends IA2, IB extends IB1, IC extends IC1>
void initialize(Class<IC> forClass, IA a, IB b) {
    synchronize(_simpleLock) {
        try {
            // >>> lookupMatching constructor types ??? <<<
            // 1) getDeclaredConstructors
            // 2) match each permutation on class
            // and its interfaces (from sub to super?)
            // handling possible null objects?
            Class[] types =
            //
            IC constructor = forClass.getDeclaredConstructor(types);
            constructor.setAccessible(true);
            _instance = constructor.newInstance(new Object[]{a,b});
            _instance.setInitialized(true);
        } catch {Exception e} {
        }
    }
}
usage :
// call 1
C.initialize(C.class, new IA2 {impl}, new IB1 {impl});

// call 2
class IA7 implements IA2 {
}
C.initialize(C.class, new IA7(bla,bla), new IB1 {impl});

// call 3
class A7 extends A implements IA6,IA7 {
```

```

}

class IA7 extends A implements IA6 {
}

class IA7 implements IA2 {
}

class C2 implements IC1 {

    protected C2(IA7 a, IB1 b) {
    }

    protected C2(IA6 a, IB1 b) {
    }
}

```

C.initialize(C2.class, new A7(bla,bla), new IB1 {impl});
 // or
 C.initialize(C2.class, new IA6(noblbla), new IB1 {impl});

(iii) Create a method named display_all_info with suitable parameters. This method should display all the information about the object of a class.

Ans. An Object of class is created by call constructor to initialize its data member! Now if you create an object of a subclass you will call a suitable constructor of that subclass to initialize its data members. Can you tell how those data members of the parent class, which subclass is inheriting will be initialized? Therefore, to initialize superclass (parent class) data member, superclass constructor is called in subclass constructor.

To call a superclass constructor write super (argument-list) in subclass constructor and this should be the very first statement in the subclass constructor. This argument list includes the arguments needed by superclass constructor. Since the constructor can be overloaded, super () can be called using any form defined by the superclass. In case of constructor overloading in superclass, which of the constructors will be called is decided by the number of parameters or the type of parameter passed in super().

Now let us take one example program to show how subclass constructor calls superclass constructor.

```

class Student
{
    String name;
    String address;
    int age;

```

```
Student( String a, String b, int c)
{
name = a;
address = b;
age = c;
}
void display( )
{
System.out.println(" *** Student Information ***");
Sstem.out.println("Name : "+name+"\n"+ "Address:" +address+"\n"+ "Age:" +age);
}
}
class PG_Student extends Student
{
int age;
int percentage;
String course;
PG_Student(String a, String b, String c, int d , int e)
{
super(a,b,d);
course = c;
percentage = e;
age = super.age;
}
void display()
{
super.display();
System.out.println("Course:" +course);
}
}
class Test_Student
{
public static void main(String[] args)
{
Student std1 = new Student("Mr. Amit Kumar" , "B-34/2 Saket J Block",23);
PG_Student pgstd1 =new PG_Student("Mr.Ramjeet ", "241-Near Fast Lane Road
Raipur" , "MCA", 23, 80);
std1.display();
pgstd1.display();
}
}
Output:
*** Student Information ***
```

Name : Mr. Amit Kumar
 Address:B-34/2 Saket J Block

Age:23

*** Student Information ***

Name : Mr.Ramjeet

Address:241- Near Fast Lane Road Raipur

Age:23

Course:MCA

(iv) Write the main method that demonstrates polymorphism.

Ans. A superclass has method that is overridden by its subclasses, then the different versions of the overridden methods are invoked or executed with the help of a superclass reference variable.

Assume that subclasses (Hockey_Player) that derive from Player abstract class are defined with each subclass having its own Play() method.

abstract class Player // class is abstract

```
{
private String name;
public Player(String nm)
{
name=nm;
}
public String getName() // regular method
{
return (name); 40
}
public abstract void Play();
// abstract method: no implementation
}
class Hockey_Player extends Player
{
Hockey_Player( String var)
{
}
public void Play()
{
System.out.println("Play Hockey:"+getName());
}
Player ref; // set up var for an Playerl
Hockey_Player aHplayer = new Hockey_Player("Dhanaraj");
// now reference each as an Animal
ref = aHplayer;
ref.Play();
```

```
}
```

Output:

Play Hockey:Dhanaraj

(c) A text file stored on the disk is to be opened for reading operations. What are the different checks that should be performed on this file? Write the portion of the code in Java to perform these checks.

Ans. Let us see how to use a FileReader stream to read character data from text files. In the following example read () method obtains one character at a time from the file. We are reading bytes and counting them till it meets the EOF (end of file). Here FileReader is used to connect to a file that will be used for input.

// Program

```
import java.io.*;
public class TextFileInput_Appl
{
    public static void main (String args[])
    {
        File file = null;
        if (args.length > 0 ) file= new File (args[0]);
        if (file== null|| !file.exists())
        {
            file=new File("TextFileInput_Appl.Java");
        }
        int numBytesRead=0;
        try
        {
            int tmp=0;
            FileReader filereader = new FileReader (file);
            while( tmp !=-1)
            {
                tmp= filereader.read();
                if (tmp !=-1) numBytesRead++;
            }
        }
        catch (IOException e)
        {
            System.out.println("IO erro" +e);
        }
        System.out.println("Number of bytes read; " + numBytesRead);
        System.out.println("File.length()="+file.length());
    }
}
```

For checking whether we read all the bytes of the file “TextFileInput_Appl.Java” in this program we are comparing both, length of file and the bytes read by read() method. You can see and verify the result that both values are 656.

Output:

Number of bytes read: 656

File.length ()=656

(d) Explain the thread model with the help of a diagram.

Ans. The benefit of Java’s multithreading is that a thread can pause without stopping other parts of your program. A paused thread can restart. A thread will be referred to as dead when the processing of the thread is completed. After a thread reaches the dead state, then it is not possible to restart it.

The thread exists as an object; threads have several well-defined states in addition to the dead states. These state are:

(1) Ready State: When a thread is created, it doesn’t begin executing immediately. We must first invoke start () method to start the execution of the thread. In this process, the thread scheduler must allocate CPU time to the Thread. A thread may also enter the ready state if it was stopped for a while and is ready to resume its execution.

(2) Running State: Threads are born to run, and a thread is said to be in the running state when it is actually executing. It may leave this state for a number of reasons.

(3) Waiting State: A running thread may perform a number of actions that will cause it to wait. A common example is when the thread performs some type of input or output operations.

In figure given below, a thread in the waiting state can go to the ready state and from there to the running state. Every thread after performing its operations has to go to the dead state.

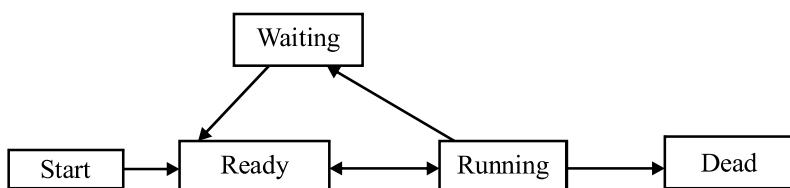


Fig: The Thread States

A thread begins as a ready thread and then enters the running state when the thread scheduler schedules it. The thread may be prompted by other threads and returned to the ready state, or it may wait on a resource, or simply stop for some time. When this happens, the thread enters the waiting state. To run again, the thread must enter the ready state. Finally, the thread will cease its execution and enter the dead state.

(e) What is a layout manager? Explain the flow layout and grid layout with the help of an example of each.

Ans. Refer to Chapter-13, Q.No.-21, Page No.-185

Grid Layout

A Grid Layout divides an applet into a specified number of rows and columns, which form a grid of cells, each equally sized and spaced. It is important to note that each is equally sized and spaced as there is another similar named Layout known as Grid Bag Layout .As Components are added to the layout they are placed in the cells, starting at the upper left hand corner and moving to the right and down the page. Each component is sized to fit into its cell. This tends to squeeze and stretch components unnecessarily.

You will find the Grid Layout is great for arranging Panels. A Grid Layout specifies the number of rows and columns into which components will be placed. The applet is broken up into a table of equal sized cells.

Grid Layout is useful when you want to place a number of similarly sized objects. It is great for putting together lists of checkboxes and radio buttons as you did in the Ingredients applet.

(f) What is an event in the context of Java? Explain the semantic event with the help of an example.

Ans. Refer to Dec-2005, Q.No.-1(h)

Q2. (a) Differentiate between the following:

(i) Procedural paradigm versus Object-oriented paradigm

Ans. Refer to Chapter-1, Q.No.-2, Page No.-2

(ii) While versus For statements of Java

Ans. The do-while statement

The Java programming language also provides another statement which is similar to the while statement; the do-while statement. The general syntax of the do-while is:

```
do
{
    statement(s)
} while (expression);
```

The do-while evaluates the expression at the bottom instead of evaluating it at the top of the loop. Thus the code block associated with a do-while is executed at least once.

Here is the previous example program rewritten using do-while:

```
public class Do While Example
{
    public static void main (String[ ] args)
{
```

```

int count = 1;
int product;
System.out.println("Table of 6");
do
{
product = 6 * count;
System.out.println(" 6 x " + count + " = " + product);
count++;
} while (count <= 10);
}
}

```

The for Statement

The for statement provides a way to iterate over a range of values. The general form of the for statement can be expressed as:

for (initialisation; termination; increment)

```

{
statement
}
```

- The initialisation is an expression that initializes the loop. It is executed only once at the beginning of the loop.
- The termination expression determines when to terminate the loop. This expression is evaluated at the top of iteration of the loop. When the expression evaluates to false, the loop terminates.
- The increment is an expression that gets invoked after each iteration through the loop. All these components are optional.

Often for loops are used to iterate over the elements in an array, or the characters in a string. The following example uses a for statement to iterate over the elements of an array and print them:

```

public class For Example
{
public static void main (String[ ] args)
{
int[ ] array O flnts = { 33, 67, 31, 5, 122, 77,204, 82, 163, 12, 345, 23 };
for (int i = 0; i < arrayOflnts.length; i++)
{
System.out.print (array O flnts [i] + " ");
}
System.out.println();
}
}
```

The output of this program is: 33 67 31 5 122 77 204 82 163 12 345 23.

(iii) Function overloading versus Method overriding

Ans. Refer to Chapter-5, Q.No.-3

(iv) Java application versus Java applet

Ans. Refer to Chapter-2, Q.No.-1

(b) Explain the importance/uses of the following with the help of an example each:

(i) Bitwise operators

Ans. Refer to Chapter-3, Q.No.-6

(ii) Operators precedence

Ans. Operator precedence is important. Rules determine which mathematical operation takes place first, i.e. takes precedence over others. Parentheses, (), may be used to force an expression to a higher precedence.

Operators higher in the chart have a higher precedence. () is higher than +.

1 has higher precedence than 2 which has higher precedence than 3.

A sub-expression in parentheses is evaluated first, regardless of the operators involved.

If there are two or more operators having the same precedence, they are usually evaluated left to right; some operators have a right to left rule and those will be identified later.

Knowing the precedence rules is good; however, it is better to add () to our expressions when it is important to improve readability.

Examples:

$2 + 3 * 4$ evaluates to 14 (multiplication first, addition second).

$(2 + 3) * 4$ evaluates to 20 (inside parentheses first, multiplication second).

$(2 * (4 + (6/2)))$ evaluates to 14 (inside parentheses first, work outward).

$4 * 5/2 * 5$ evaluates to 50 (start left, move right).

(iii) Array initializer

Ans. The Java programming language provides one another way also for creating and initialising an array. Here is an example of this syntax:

```
boolean [] answers = {true, false, true, true, false};
```

The number of values provided between {and} determines the length of the array.

(iv) Byte code

Ans. The most knowledgeable C and C++ programmers know the assembler instruction set of the processor for which they are compiling. This knowledge is crucial when debugging and doing performance and memory usage tuning. Knowing the assembler instructions that are generated by the compiler for the source code we write, helps us know how we might code differently to achieve memory or performance goals. In addition, when tracking down a problem, it is often useful to use a debugger to disassemble the source code and step through the assembler code that is executing.

Q3. (a) Define the term exception in the context of Java. What are the causes of exception? What are the actions that may need to be performed if an exception is caught? Explain the uses of try and catch in Java with the help of an example program segment.

Ans. Refer to Chapter-7, Q.No.-1, Page No.-90

Causes of Exception

Exception arises at runtime due to some abnormal condition in program for example when a method. For division encounters an abnormal condition that it can't handle itself, i.e. "divide by zero," then this method may *throw* an exception.

If a program written in Java does not follow the rule of Java language or violates the Java execution environment, constraints exception may occur. There may be a manually generated exception to pass on some error reports to some calling certain methods.

If an exception is caught, there are several things that can be done:

- Fix the problem and try again.
- Do something else instead to avoid the problem.
- Exit the application with System.exit()
- Rethrow the exception to some other method or portion of code.
- Throw a new exception to replace it.
- Return a default value (a non-void method: traditional way of handling exceptions).
- Eat the exception and return from the method (in a void method). In other words, don't give importance to the exception.
- Eat the exception and continue in the same method (Rare and dangerous. Be very careful if you do this).

Using try catch

Now we see how to write programs in Java, which take care of exceptions handling.

The program given below:

```
//program
public class Excep_Test
{
    public static void main(String[] args)
    {
        int data[] = {2,3,4,5};
        System.out.println("Value at : " + data[4]);
    }
}
```

Output:

```
java.lang.ArrayIndexOutOfBoundsException
at Excep_Test.main(Excep_Test.java:6)
Exception in thread "main"
```

At runtime, this program has got ArrayIndexOutOfBoundsException. This exception occurs because of the attempt to print beyond the size of array.

Now we see how we can catch this exception.

To catch an exception in Java, we write a try block with one or more catch clauses. Each catch clause specifies one exception type that it is prepared to handle. The try block places a fence around the code that is under the watchful eye of the associated catchers. If the bit of code delimited by the try block throws an exception, the associated catch clauses will be examined by the Java virtual machine. If the virtual machine finds a catch clause that is prepared to handle the thrown exception, the program continues execution starting with the first statement of that catch clause, and the catch block is used for executing code to handle exception and graceful termination of the program.

```
public class Excep_Test
{
    public static void main(String[] args)
    {
        try
        {
            int data[] = {2,3,4,5};
            System.out.println("Value at : " + data[4]);
        }
        catch( ArrayIndexOutOfBoundsException e)
        {
            System.out.println("Sorry you are trying to print beyond the size of data[]");
        }
    }
}
```

Output:

Sorry you are trying to print beyond the size of data[]

(b) What is an Interface? What is meant by ‘implementing interfaces’? Explain with the help of an example.

Ans. Refer to Chapter-8, Q.No.-3 , Page No.-110 and Refer to June-2012, Q.No.-1(f) Page No.-352

(c) What are the uses of “this” keyword in Java? Explain with the help of an example.

Ans. Refer to Chapter-6, Q.No.-3, Page No.-81

Q4. (a) What are the advantages of multithreading in Java? Explain the interthread communication with the help of an example.

Ans. Refer to Dec-2005, Q.No.-4(c), Page No.-212 and Refer to June-2010, Q.No.-4(a)(i), Page No.-320

(b) Write a program in Java that converts a string 127 into equivalent integer value and prints it.

Ans. There are 3 main ways to *convert String to int in Java*, using the constructor of Integer class, parseInt () method of java. lang. Integer and Integer. Value Of () method. Though all those methods return an instance of java. lang. Integer, which is a wrapper class for primitive int value, it's easy to convert Integer to int in Java. From Java 5, you don't need to do anything, autoboxing will automatically convert Integer to int. For Java 1.4 or lower version, you can use int Value () method from java. lang. Integer class, to convert Integer to int. As the name suggest, parseInt () is the core method to convert String to int in Java. parseInt () accept a String which must contain decimal digits and first character can be an ASCII minus sign (-) to denote negative integers. parseInt () throws Number Format Exception, if provided String is not convertible to int value.

By the way parseInt is an overloaded method and it's overloaded version takes radix or base e.g. 2,8,10 or 16, which can be used to convert binary, octal, hexadecimal String to int in Java. Integer. Value Of () is another useful method to convert String to Integer in Java, it offers caching of Integers from -128 to 127. Internally, value Of() also calls parseInt () method for String to int conversion. In this Java programming tutorial, we will see all three ways to convert String to int value in Java.

String to Integer Conversion Example

3 ways to convert String to Integer in Java with Example This Java program converts both positive and negative numeric String to int in Java. As I said, it uses three methods. First example uses constructor of Integer class to create Integer from String, later that Integer is converted to int using autoboxing in Java.

Second example uses Integer.parseInt() and third String conversion examples use Integer.valueOf() method to convert String to int in Java.

By the way, be careful with String.valueOf() method, as it may return same object on multiple class, if integer value is from it's cache. Also comparing int to Integer is error prone in autoboxing and can produce surprising results. See pitfalls of autoboxing in Java for more details.

```
package test;
/**
 * Java program to convert String to int in Java. From Java 5 onwards
 * autoboxing can be used to convert Integer to int automatically.
 *
 * @author http://java67.blogspot.com
 */
public class StringToIntegerHowTo {

    public static void main(String args[]) {
        //Converting positive integer value String to int in Java
        String integer = "123";
    }
}
```

```
// String to int conversion using Integer constructor and autoboxing  
int number = new Integer(integer);  
System.out.println("Integer created from String in Java : " + number);  
  
// Converting String to int using parseInt() method of java.lang.Integer  
number = Integer.parseInt(integer);  
System.out.println("String to int using parseInt() : " + number);  
  
// String to int conversion using valueOf() method of Integer class  
number = Integer.valueOf(integer);  
System.out.println("String to int Conversion example using valueOf : " +  
number);  
  
String negative = "-123";  
System.out.println("Converting negative String to int in Java");  
System.out.println("String to Integer Constructor Example : " + new  
Integer(negative));  
System.out.println("String to Integer parseInt Example : " +  
Integer.parseInt(negative));  
System.out.println("String Integer valueOf Example : " +  
Integer.valueOf(negative));  
  
}  
}
```

Output:

```
Integer created from String in Java : 123  
String to int using parseInt() : 123  
String to int Conversion example using valueOf : 123  
Converting negative String to int in Java  
String to Integer Constructor Example : -123  
String to Integer parseInt Example : -123  
String Integer valueOf Example : -123
```

That's all on How to convert String to int or Integer in Java. By the way, converting int to Integer is ridiculously easy in Java and done automatically by Java platform, from Java 5 onwards. As said in best way to convert String to number in Java, you can choose between parseInt () and value Of () for String to int conversion. Integer. Value Of () is preferred because it offers caching and eventually delegate call to parseInt for String to int conversion, if integer value is not available in cache.

(c) What are the uses of the following in the context of I/O in Java?

(i) File class

Ans. The File class is not I/O. It provides just an identifier of files and directories. Files and directories are accessed and manipulated through java.io.File class. So, we always remember that creating an instance of the File class does not create a file in the operating system. The object of the File class can be created using the following types of File constructors:

```
File MyFile = new File("c:/Java/file_Name.txt");
File MyFile = new File("c:/Java", "file_Name.txt");
File MyFile = new File("Java", "file_Name.txt");
```

The first type of constructor accepts only one string parameter, which includes file path and file name, here in given format the file name is file_Name.txt and its path is c:/Java.

In the second type, the first parameter specifies directory path and the second parameter denotes the file name. Finally, in the third constructor, the first parameter is only the directory name and the other is file name.

(ii) Buffered stream classes

Ans. Refer to June-2014, Q.No.-1(e), Page No.-372

Q5. Explain the following with the help of an example/diagram/program, if needed:

(a) Creating a choice list in Java

Ans. Choice List is created from the Choice class. List has components that enable a single item to be picked from a pull-down list. We encounter this control very often on the web when filling out forms.

The first step in creating a Choice

We can create a choice object to hold the list, as shown below:

Choice cgender = new Choice(); Items are added to the Choice List by using addItem(String) method the object. The following code adds two items to the gender choice list. cgender.addItem("Female");

cgender.addItem("Male"); After we add the Choice List it is added to the container like any other component using the add() method.

The following example shows an Applet that contains a list of shopping items in the store.

```
import java.awt.*;
Public class ChoiceTest extends java.applet.Applet {
    Choice shoplist = new Choice();
    Public void init(){
        shoplist.addItem("Bed And Bath");
        shoplist.addItem("Furniture");
        shoplist.addItem("Clothing");
        shoplist.addItem("Home Appliance");
        shoplist.addItem("Toys and Accessories");
        add(shoplist);
    }
}
```

Output:



The choice list class has several methods, which are given in Table.

Table: Choice List Class Methods

Method	Action
getItem()	Returns the string item at the given position (items inside a choice begin at 0, just like arrays)
countItems()	Returns the number of items in the menu
getSelectedIndex()	Returns the index position of the item that's selected
getSelectedItem()	Returns the currently selected item as a string
select(int)	Selects the item at the given position
select(String)	Selects the item with the given string

(b) Using checkboxes in Java

Ans. Refer to Chapter-9, Page No.-120

(c) Drawing a rectangle in Java

Ans. Drawing rectangles is simple. Start with a Graphics object g and call its drawRect() method:

```
public void drawRect(int x, int y, int width, int height)
```

The first argument int is the left hand side of the rectangle, the second is the top of the rectangle, the third is the width and the fourth is the height. This is in contrast to some APIs where the four sides of the rectangle are given.

Remember that the upper left hand corner of the applet starts at (0, 0), not at (1, 1). This means that a 100 by 200 pixel applet includes the points with x coordinates between 0 and 99, not between 0 and 100. Similarly the y coordinates are between 0 and 199 inclusive, not 0 and 200.

(d) Servlet Life Cycle

Ans. Refer to Chapter-11, Q.No.-4, Page No.-141

(e) Executing SELECT statement in Java

Ans. A query is expected to return a set of tuples as the result, and not change the state of the database. Not surprisingly, there is a corresponding method called executeQuery, which returns its results as a ResultSet object. It is a table of data representing a database result set, which is usually generated by executing a statement that queries the database.

A ResultSet object maintains a cursor pointing to its current row of data. Initially the cursor is positioned before the first row. The next method moves the cursor to the next row, and because it returns false when there are no more rows in the ResultSet object, it can be used in a while loop to iterate through the result set. A default ResultSet object is not updatable and has a cursor that moves forward only. In the program code given below:

```
String ename,eaddress;
float esal;
ResultSet rs = stmt.executeQuery("SELECT * FROM Employee");
while ( rs.next() ) {
    ename = rs.getString("emp_name");
    eaddress = rs.getString("emp_address");
    esal = rs.getFloat("emp_salary");
    System.out.println(ename + " address is" + eaddress + " draws salary " + esal + " in dollars");
}
```

The tuples resulting from the query are contained in the variable which is an instance of ResultSet. A set is of not much use to us unless we can access each row and the attributes in each row.

Now we should note that each invocation of the next method causes it to move to the next row, if one exists and returns true, or returns false if there is no remaining row.

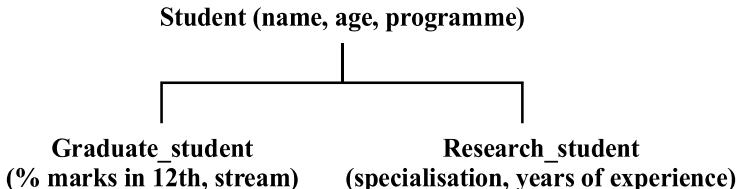
We can use the getXXX method of the appropriate type to retrieve the attributes of a row. In the above program code, getString and getFloat methods are used to access the column values. One more thing we can observe that the name of the column whose value is desired is provided as a parameter to the method.

Similarly, while working with a PreparedStatement, you can execute a query by first plugging in the values of the parameters, and then invoking the executeQuery on it.

1. ename = rs.getString(1);
- eaddress = rs.getFloat(3);
- esal = rs.getString(2);
2. ResultSet rs = prepareUpdateEmployee.executeQuery();

Note: Q. No. 1 is compulsory and carries 40 marks. Attempt any three questions from the rest of the questions.

Q1. (a) Consider the following class hierarchy:



A university has two types of students - graduate students and research students. The University maintains the record of name, age and programme of every student. For graduate students, additional information like percentage of marks and stream, like science, commerce, etc. is recorded; whereas for research students, additionally, specialisation and years of working experience, if any, is recorded. Each class has a constructor. The constructor of subclasses makes a call to constructor of the superclass. Assume that every constructor has the same number of parameters as the number of instance variables. In addition, every subclass has a method that may update the instance variable values of that subclass. All the classes have function `display_student_info()`, the subclasses must override this method of the base class. Every student is either a graduate student or a research student.

Perform the following tasks for the description given above using Java:

(i) Create the three classes with proper instance variables and methods, with suitable inheritance.

Ans. class Student

```
{  
    String name;  
    String programme;  
    int age;  
    Student(String name,int age,String programme)  
    {  
        this.name=name;  
        this.age=age;  
        this.programme=programme;  
    }  
    void display_student_info()
```

```
{  
    System.out.println("****Students Information***\n");  
    System.out.println("Name of student :" + name + "\n");  
    System.out.println("Age of student :" + age + "\n");  
    System.out.println("Name of programme :" + programme + "\n");  
}  
}  
}  
class Graduate_Student extends Student  
{  
    float percentage;  
    String stream;  
    Graduate_Student(String name,int age,String programme,float percentage,String  
stream)  
    {  
        super(name, age, programme);  
        this.percentage=percentage;  
        this.stream=stream;  
    }  
    void display_student_info()  
    {  
        super. display_student_info();  
        System.out.println("Student's percentage is: " +percentage+ " %\n");  
        System.out.println("Student's stream is: "+stream+"\n");  
    }  
}  
class Research_Student extends Student  
{  
    String specialisation;  
    int years_of_exp;  
    Research_Student(String name,int age,String programme,String specialisation,int  
years_of_exp)  
    {  
        super(name, age, programme);  
        this. specialisation = specialisation;  
        this. years_of_exp = years_of_exp;  
    }  
    void display_student_info()  
    {  
        super. display_student_info();  
        System.out.println("Student's specialisation is: "+specialisation+"\n");  
        System.out.println("Student's experience: "+ years_of_exp+" years");  
    }  
}
```

```

class TestStudent
{
public static void main(String args[])
{
Graduate_Student gs= new Graduate_Student("Sneha",16,"Graduate",79,
"Science");
Research_Student rs = new Research_Student("Rohit",22,"Research",
"Automation",2);
gs. display_student_info();
rs. display_student_info();
}
}

```

(ii) Create at least one parameterised constructor for each class.

Ans. class Student

```

{
String name;
String programme;
int age;
Student(String name,int age,String programme)
{
this.name=name;
this.age=age;
this.programme=programme;
}
public static void main(String args[])
{
Student s = new Student("Ashish",17,"Graduate");
}
}
class Graduate_Student extends Student {
float percentage;
String stream;
Graduate_Student(String name,int age,String programme,float percentage,String
stream)
{
super(name, age, programme);
this.percentage=percentage;
this.stream=stream;
}
public static void main(String args[])
{

```

```

Graduate_Student gs=new Graduate_Student("Sneha",16,"Graduate",
79,"Science");
}
}
class Research_Student extends Student
{
    String specialisation;
    int years_of_exp;
Research_Student(String name,int age,String programme,String specialisation,int
years_of_exp)
{
super(name, age, programme);
this. specialisation = specialisation;
this. years_of_exp = years_of_exp;
}
public static void main(String args[])
{
Research_Student rs=new Research_Student("Rohit",22,"Research",
"Automation",2);
}
}

```

(iii) Implement the display_student_info() method in each class.

Ans. class Student

```

{
    String name;
    String programme;
    int age;
void display_student_info()
{
    System.out.println("****Students Information***\n");
    System.out.println("Name of student :" +name+ "\n ");
    System.out.println("Age of student : " +age+ "\n");
    System.out.println("Name of programme : " +programme);
}
public static void main(String args[])
{
    Student s = new Student();
    s. display_student_info();
}
}
class Graduate_Student extends Student {
    float percentage;

```

```

String stream;
void display_student_info()
{
super. display_student_info();
System.out.println("Student's percentage is: " +percentage+ " %\n");
System.out.println("Student's stream is: "+stream);
}
public static void main(String args[])
{
Graduate_Student gs= new Graduate_Student();
gs. display_student_info();
}
}
class Research_Student extends Student
{
String specialisation;
int years_of_exp;
void display_student_info()
{
super.display_student_info();
System.out.println("Student's specialisation is: "+specialisation+"\n");
System.out.println("Student's experience: "+ years_of_exp+" years");
}
public static void main(String args[])
{
Research_Student rs = new Research_Student();
rs. display_student_info();
}
}

```

(iv) Create an appropriate main method with at least two objects of each subclass. Can you create an object of the superclass? Justify your answer.

Ans. public static void main(String args[])
{
Graduate_Student gs1= new Graduate_Student("Sneha",16,"Graduate",79,
"Science");
Graduate_Student gs2= new Graduate_Student("Karan",17,"Graduate",75,
"Commerce");
Research_Student rs = new Research_Student("Rohit",22,"Research",
"Automation",2);
Research_Student rs=new Research_Student("Meera",24,"Research",
"Algorithm",4);
}

In inheritance, subclass acquires super class properties. An important point to note is, when subclass object is created a separate object of super class object will not be created. Only a subclass object is created that has super class variable. Eg:

```
class Student
{
String name;
String programme;
int age;
Student(String name, int age, String programme)
{
this. name=name;
this. age=age;
this. programme=programme;
}
void display_student_info( )
{
System. out. println("* * * Students information * * *\n");
System. out. println ("Name of student : "+name+"\n");
System. out. println("Age of student: "+age+"\n");
System. out. println ("Name of programme:" +programme+"\n");
}
}
class Graduate_Student extends Student
{
float percentage;
String stream;
Graduate_Student (String name, int age, String programme, float percentage,
String stream)
{
super (name, age, programme);
this. percentage=percentage;
this. stream=stream;
}
void display_student_info( )
{
super. display_ student_ info( );
System. out. println ("Student's percentage is:" percentage + "%\n");
System. out. println("Student's stream is: "+stream+"\n");
}
```

```

}
class TestStudent
{
public static void main (String args[])
{
Graduate_Student gs= new Graduate_Student ("Sneha",16,"Graduate",79,
"Science");
gs. display_student_info ( );
}
}

```

As we can see that both super class (Student) object name, age and programme and subclass (Graduate_Student) object name, age and programme are same, so only one object is created. This object is of class Graduate_Student (subclass) as when we try to print name of class which object is created, it will print Graduate_Student which is subclass.

(v) Write the code in main that uses objects and overriding to show polymorphism.

Ans. public static void main(String args[])

```

{
Graduate_Studentg rs=new Graduate_Student("Sneha",16,"Graduate",79,
"Science");
Research_Student rs=new Research_Student("Rohit",22,"Research",
"Automation",2);
gs. display_student_info();
rs. display_student_info();
}
```

(b) What is multithreading? What are the advantages of multithreading? What is a main thread in the context of Java?

Refer to June-2014, Q.No.-2(a), Page No.-374 & Dec-2005, Q.No.-4(c), Page no.-212

When a Java program starts up, one thread begins running immediately. This is usually called the main thread of your program, because it is the one that is executed when our program begins. The main thread is important for two reasons:

- It is the thread from which other "child" threads will be spawned.
- It must be the last thread to finish execution. When the main thread stops, your program terminates.

Although the main thread is created automatically when our program is started, it can be controlled through a Thread object.

(c) Write a Java program that accepts the input from the keyboard and writes it to a text file.

```
Ans.import java. util.Scanner;
import java.io. FileWriter;
class Scanner Test
{
public static void main (String args [])
{
Scanner s = new Scanner (System.in);
System.out.println("Enter your name");
String name = s.nextLine ();
Filewriter writer = new FileWriter ("MyFile. text");
writer. write (name);
writer. close ();
}
}
```

(d) Explain how an event is handled in Java with the help of an example program.

Refer to Dec-2009, Q.No.-1(a), Page No.-303

(e) Assume that a database named “Student” exists with attributes student_ID, student_name and programme. Write a Java program segment which will execute a SELECT query and display the resultant records. You need not write the connection command.

```
Ans. public class query1
{
public static void main (String arg[])
{
Connection con;
Statement st = con.createStatement ();
Resultset rs = st. executeQuery ("Select * from Student");
{
while (rs.next ())
{
int student ID = rs. getInt ("student_ID");
String studentName = rs. getString("student_name");
String studentProgramme = rs.getString ("programme");
System. out.print("Id:"+studentID);
System.out.print("Name:"+studentName);
System.out.print("Programme:"+studentProgramme);
}}
```

```
}
```

(f) What is Bytecode in Java? What are the advantages of using bytecode in Java?

Refer to Chapter-2, Q.No.-4, Page No.-18

Q2. (a) List the salient features of object oriented programming approach that distinguishes it from the procedural programming.

Ans. The salient features of object oriented programming are:

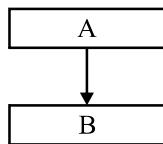
- More emphasis is on data rather than procedure.
- Programs are modularised into entities called objects.
- Data structures methods characterise the objects of the problem.
- Since the data is not global, there is no question of any operations other than those defined within the object, accessing the data. Therefore, there is no scope of accidental modification of data.
- It is easier to maintain programs. The manner in which an object implements its operations is internal to it. Therefore, any change within the object would not affect external objects. Therefore, systems built using objects are resilient to change.
- Object reusability, which can save many human hours of effort, is possible. An application developer can use objects like ‘array’, ‘list’, ‘windows’, ‘menus’, ‘event’ and many other components, which were developed by other programmers, in her program and thus reduce program development time.
- It employs bottom-up approach in program design.

Now, Refer to Chapter-1, Q.No.-2, Page No.-2

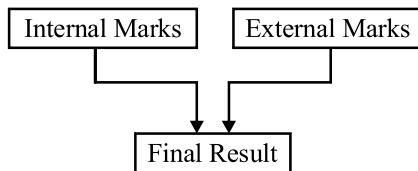
(b) Explain different forms of inheritance with the help of diagrams. What are the advantages of using inheritance?

Ans. Inheritance may have different forms, which are as follows:

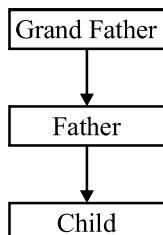
- Single Inheritance: In this form, a subclass can have only one super class.
- Multiple Inheritance: This form of inheritance can have several superclasses.
- Multilevel Inheritance: This form has subclasses derived from another subclass. The example can be grandfather, father and child.
- Hierarchical Inheritance: This form has one superclass and many subclasses. More than one class inherits the traits of one class. For example: bank accounts.



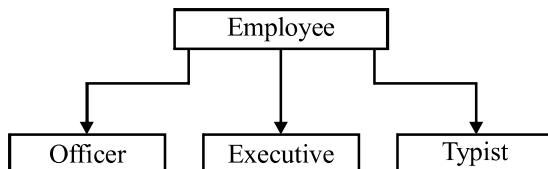
(a) Single Inheritance



(b) Multiple-Inheritance



(c) Multilevel Inheritance



(d) Hierarchical Inheritance

fig: Different forms of inheritance

Now, Refer to Chapter-1, Q.No.-9, Page No.-10

(c) Explain the following with the help of an example of each:

(i) Dynamic Initialisation

Refer to Chapter-3, Q.No.-7, Page No.-36

(ii) Operator Precedence

Ans. Table shows the operator precedence. The operators in this table are listed in order of the following precedence rule: the higher in the table an operator appears, the higher its precedence. In an expression, operators that have higher precedence are evaluated before operators that have relatively lower precedence. Operators on the same line have equal precedence.

Table: Operators precedence

Special operators	[] Array element reference Member selection (params) Function call
unary operators	+ + , - - , + , - , ~ , !
Creation of cast	new(type)expression
multiplicative	* , / , %
Additive	+ , -
Shift	<<,>>,>>>
Relational	<,>,<=,>=,instance of

When operators of same precedence appear in the same expression, some rule must govern which is evaluated first. All binary operators except for the assignment operators are evaluated in left-to-right order and assignment operators are evaluated right to left.

(iii) Switch Statement

Refer to Chapter-4, Page No.-47

(iv) Array Initialisation

Ans. Although it is not possible to assign to all elements of an array at once using an assignment expression, it is possible to initialise some or all elements of an array when the array is defined. The syntax like this:

int a[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

The list of values, enclosed in braces { }, separated by commas, provides the initial values for successive elements of the array.

(Under older, pre-ANSI C compilers, we could not always supply initialisers for “local” arrays inside functions; we could only initialise “global” arrays, those outside of any function. Those compilers are now rare, so we shouldn’t have to worry about this distinction any more.)

If there are fewer initialisers than elements in the array, the remaining elements are automatically initialised to 0. For example,

int a [10] = {0, 1, 2, 3, 4, 5, 6,};

would initialise a[7], a[8] and a[9] to 0. When an array definition includes an initialise, the array dimension may be omitted, and the compiler will infer the dimension from the number of initialises. For example,

int b[] = {10, 11, 12, 13, 14};

would declare, define, and initialise an array b of 5 elements (i.e. just as if you'd typed int b[5]). Only the dimension is omitted; the brackets [] remain to indicate that b is in fact any array.

In the case of arrays of char, the initialiser may be a string constant:

```
char s1[7] = "Hello,";
char s2[10] = "there,";
char s3[] = "world!";
```

As before, if the dimension is omitted, it is inferred from the size of the string initialiser. (All strings in C are terminated by a special character with the value 0. Therefore, the array s3 will be of size 7, and the explicitly-sized s1 does need to be of size at least 7. For s2, the last 4 characters in the array will all end up being this zero-value character.)

Q3. (a) What is Static Method in Java? Explain with the help of an example.

Ans. A static method is a characteristic of a class, not of the objects it has created. Static variables and methods are also known as class variables or class methods since each class variable and each class method occurs once per class. Instance methods and variables occur once per instance of a class or you can say every object is having its own copy of instance variables.

One very important point to note here is that a program can execute a static method without creating an object of the class. All other methods must be invoked through an object, and, therefore an object must exist before they can be used. You have seen that every Java application program has one main() method. This method is always static because Java starts execution from this main() method, and at that time no object is created.

For example:

```
import Java.util.Date;
class DateApp
{
    public static void main(String args[])
    {
        Date today = new Date();
        System.out.println(today);
    }
}
```

The last line of the main() method uses the System class from the Java.lang package to display the current date and time.

(b) Explain with the help of an example program, how objects can be passed as parameters in Java.

Ans. In Java, we can pass a reference of the object to the formal parameter. If any changes to the local object that take place inside the method will modify the object

that was passed to it as argument. Due to this reason objects passing as parameter in Java are referred to as passing by reference. In the program given below object of the class Marks is passed to method Set_Grade, in which instance variable grade of the object passed is assigned some value. This reflects that the object itself is modified.

```
class Marks
{
String name;
int percentage;
String grade;
Marks(String n, int m)
{
name = n;
percentage = m;
}
void Display()
{
System.out.println("Student Name :" + name);
System.out.println("Percentage Marks:" + percentage);
System.out.println("Grade : " + grade);
System.out.println("*****");
}
}
class Object_Pass
{
public static void main(String[] args)
{
Marks ob1 = new Marks("Naveen",75);
Marks ob2 = new Marks("Neeraj",45);
Set_Grade(ob1);
System.out.println("*****");
ob1.Display();
Set_Grade(ob2);
ob2.Display();
}
static void Set_Grade(Marks m)
{
if (m.percentage >= 60)
m.grade = "A";
else if( m.percentage >=40)
m.grade = "B";
else
m.grade = "F";
}
```

```
}
```

```
}
```

Output of this program is:

```
*****
```

Student Name :Naveen

Percentage Marks:75

Grade : A

```
*****
```

Student Name :Neeraj

Percentage Marks:45

Grade : B

```
*****
```

(c) What is a Package in Java? How are they related to CLASSPATH? Explain with the help of an example program.

Refer to Chapter-8, Q.No.-1, Page No.-105 & June-2008, Q.No.-1(d), Page No.- 279

(d) What is an Interface in Java? How are they different from ABSTRACT classes?

Refer to Chapter-8, Q.No.-3 & 4, Page No.-110

Q4. (a) Explain with the help of an example program, how interthread communication is performed in Java using wait() and notify() and other methods.

Refer to June-2010, Q.No.-4(i), Page No.-320

(b) How is Character class in Java different to String class? Explain how you will compare two objects of String class. Also write a program that converts lowercase characters to uppercase characters of a string.

Ans. Character and strings are the most important data type. Java provides Character, String, and StringBuffer classes for handling characters, and strings. String class is used for creating the objects which are not to be changed. The string objects for which contents are to be changed StringBuffer class, is used. Character class provides various methods like compareTo, equals, isUpperCase. String class provides methods for index operations, substring operations, and a very special group of valueOf methods. Comparison methods are also provided in String class. String class methods like replace, toLowerCase, trim are available for minor modifications though, in general string classes is not used for dynamic Strings. StringBuffer objects allow to insert character or substring in the middle or append it to the end. StringBuffer also allows deletion of a substring of a string.

Now, Refer to June-2010, Q.No.-2(b)(i), Page No.-318

(c) What is the purpose of the following Stream classes?

(i) PrintStream

Ans. PrintStream class: It provides the familiar print() and println() methods.

(ii) RandomAccessFile

Ans. RandomAccessFile is somewhat disconnected from the input and output streams in java.io—it doesn't inherit from the InputStream or OutputStream. This has some disadvantages in that we can't apply the same filters to RandomAccessFiles that we can to streams. However, RandomAccessFile does implement the DataInput and DataOutput interfaces, so if we design a filter that works for either DataInput or DataOutput, it will work on some sequential access files (the ones that implemented DataInput or DataOutput) as well as any RandomAccessFile.

(iii) ByteArrayInputStream

Ans. ByteArrayOutputStream provides some additional methods not declared for OutputStream. The reset() method resets the output buffer to allow writing to restart at the beginning of the buffer. The size() method returns the number of bytes that have been written to the buffer. The write to () method is new.

- SequenceInputStream: Concatenate multiple input streams into one input stream.
- StringBufferInputStream: Allow programs to read from a StringBuffer as if it were an input stream.

(iv) FilterOutputStream

Ans. The FilterOutputStream class provides three subclasses - BufferedOutputStream, DataOutputStream and Printstream.

Q5. Explain the following with the help of a/an diagram/example/program, if needed:

(a) Paint() Method of Applet

Refer to Dec-2006, Q.No.-1(e), Page No.-240

(b) HTML Applet Tag

Ans. The HTML <applet> tag specifies an applet. It is used for embedding a java applet within an HTML document. eg: –

Now, Refer to Dec-2009, Q.No.-1(b), Page No.-305

(c) Button

Refer to Chapter-9, Page No.-118

(d) Checkbox Group**Ans. The Checkbox group**

Checkbox Group is also called like a radio button or exclusive check boxes. To organise several Checkboxes into a group so that only one can be selected at a time,

we can create Checkbox Group object as follows:

Checkbox Group radio = new Checkbox Group ();

The Checkbox Group keeps track of all the check boxes in its group. We have to use this object as an extra argument to the Checkbox constructor.

Checkbox (String, Checkbox Group, Boolean) creates a checkbox labeled with the given string that belongs to the Checkbox Group indicated in the second argument. The last argument equals true if box is checked and false otherwise.

The set Current (checkbox) method can be used to make the set of currently selected check boxes in the group. There is also a get Current () method, which returns the currently selected checkbox.

(e) Grid Layout (No need of writing program)

Refer to June-2009, Q.No.-4(a), Page No.-298

(f) Container

Refer to Chapter-9, Page No.-115

(g) Uses of RMI

Refer to Chapter-11, Q.No.-1, Page No.-138

(h) POST Method

Refer to Dec-2005, Q.No.-2(a), Page No.-206

Note: Question No. 1 is compulsory . Attempt any three questions from the rest of the questions.

Q1. (a) What is method overloading? Explain with suitable example.

Refer to Chapter-5, Q.No.-3 (Pg. No.-67)

(b) What is an abstract class? How is it used to implement polymorphism in Java? Give suitable example in support of your answer.

Refer to June-2008, Q.No.-2(a) (Pg. No.-281)

(c) What is JVM? Explain advantages of JVM.

Refer to Chapter-2, Q.No.-3 (Pg. No.-18)

Advantages of Java Virtual Machine

Java Virtual Machine (JVM) has some not advantages, which are, as follows:

(1) Ensures Security: When Java was initially developed, *security* was the priority. That is the reason why Java programs run separated in an enclosed area of the Java Virtual Machine which acts as a protecting shield.

Java Virtual Machine allows developers to write Java programs that are highly secure with help of its built-in security features.

Java Virtual Machine keeps the Operating System secure by preventing any malicious software from attacking it. It ensures its *safety* by not allowing the Java applications to interact with the Operating System resources.

(2) Cross Platform: A program that is Cross platform has the capability to run successfully on different types of hardware. Java is also a cross platform language i.e. it is possible to run a single piece of code written on a particular hardware on any other hardware that has JVM installed on it.

Therefore, JVM makes Java a cross platform language. A browser is not always needed for a Java program to execute, there are many java apps available too for the same. They run on the desktop the same way as the regular programs do.

(3) Just-in-time compiler: The Java Virtual Machine comes with a Just-in-time compiler which converts the Java code into a *low level machine language* that can run as fast as the regular applications.

This compiled code goes into the cached memory of the browser, this means that one can use the code again without downloading it again and again compiling it.

(d) Differentiate between “Final”, “Finally” and “Finalise” statements.

Refer to June-2009, Q.No.-1(e) (Pg. No.-295)

(e) Compare throw statement in Java with throws statement.

Refer to Chapter-7, Q.No.-4, (Pg. No.-93)

(f) Explain container class in Java. Give its significance for Java GUI programming.

Same as Chapter-9, (Pg. No.-125)

(g) What is public access specifier? How is it different from private access specifier? Explain with example.

Refer to Chapter-13, Q.No.-2 (Pg. No.-171)

(h) What is an applet? Explain various ways to execute an applet.

Refer to Dec-2006, Q.No.-1(e) (Pg. No.-240)

Q2. (a) What is servlet life cycle? Explain GET and POST methods in servlet programming.

Refer to Chapter-11, Q.No.-4 (Pg. No.-141) & Refer to Dec.-2005, Q.No.-2(a) (Pg. No.-206)

(b) What is import? Explain the need of importing a package in Java.

Refer to Chapter-8, Q.No.-1 (Pg. No.-109)

(c) What is a classpath? Discuss the utility of claspath with suitable example.

Refer to June-2008, Q.No.-1(a) (Pg. No.-279)

Q3. (a) What is inheritance? How is inheritance related to interface? Explain, how multiple inheritance can be implemented using interface in Java. Give suitable example.

Same as Chapter-13, Q.No.-31, (Pg. No.-196) and Same as Dec-2009, Q.No.-4(b) (Pg. No.-310)

(b) Write a Java program to create a Teacher class and derive Professor/ Associate_Professor/Assistant_Professor class from Teacher class. Define appropriate constructor for all the classes. Also define a method to display information of Teacher. Make necessary assumptions as required.

Ans. Constructor of sub class is invoked when we create the object of subclass, it by default invokes the default constructor of super class. Hence, in

inheritance the objects are constructed top-down. The superclass constructor can be called explicitly using the super keyword , but it should be first statement in a constructor. The super keyword refers to the superclass, immediately above of the calling class in the hierarchy. The use of multiple super keywords to access an ancestor class other than the direct parent is not permitted.

```

class teacher{
    //teacher class constructor//
    teacher(){
        System.out.println("Teacher");
    }
}

class professor extends teacher{
    professor(){
        /* It by default invokes the constructor of teacher class
         * We can use super() to call the constructor of teacher.
         * It should be the first statement in the derived class
         * constructor, you can also call the parameterized constructor
         * of teacher class by using super like this: super(10), now
         * this will invoke the parameterized constructor of int arg
        */
        System.out.println("professor of physics");
    }
}

class Associate_professor extends professor {
    Associate_professor() {
        System.out.println(" professor of physics2");
    }
}

class Assistant_professor extends Associate_professor {
    Assistant_professor() {
        System.out.println("Assistant teacher of physics");
    }
}

public static void main(String args[]){
    //Creating the object of derive class//
    professor();
    Associate_professor();
    Assistant_professor();
}
}

Output:
teacher

```

professor of physics
professor of physics2
Assistant teacher of physics

Q4. (a) What are the features of object oriented Programming? Explain encapsulation and its advantages, with an example.

Refer to Chapter-1, Q.No.-2, 5 & 7 (Pg. No.-2,7, 9)

(b) How is String class different from StringBuffer class? Write a program in Java to find the length of a given string.

Refer to Chapter-3, Q.No.-5 (Pg. No.-30) & Refer to Chapter-7, Q.No.-7 (Pg. No.-99)

(c) Write a program in Java, to copy the text content of one file into another file.

Ans. The below code would copy the content of “Gullybaba.txt” to the “MyGullybaba.txt” file. If “MyGullybaba.txt” doesn’t exist then the program would create the file first and then it would copy the content.

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
public class CopyExample
{
    public static void main(String[] args)
    {
        FileInputStream instream = null;
        FileOutputStream outstream = null;

        try{
            File infile =new File("C:\\\\Gullybaba.txt");
            File outfile =new File("C:\\\\MyGullybaba.txt");

            instream = new FileInputStream(infile);
            outstream = new FileOutputStream(outfile);

            byte[] buffer = new byte[1024];

            int length;
            /*copying the contents from input stream to
             * output stream using read and write methods
```

```

*/
while ((length = instream.read(buffer)) > 0){
    outstream.write(buffer, 0, length);
}

//Closing the input/output file streams
instream.close();
outstream.close();
System.out.println("File copied successfully!!");

}catch(IOException ioe){
    ioe.printStackTrace();
}
}
}

```

Output:

File copied successfully!!

Q5. (a) Explain each component of the statement “public static void main (String args [])”.

Ans. Java JVM will always look for specific method signature to start running an application, and that would be
“public static void main(String args[])”.

Public: It is an Access modifier, which defines who can access this Method. Public means that this Method will be accessible by any Class(If other Classes can access this Class.).

Static: Static is a keyword which identifies the class related thing. It means the given Method or variable is not instance related but Class related. It can be accessed without creating the instance of a Class.

Void: Is used to define the Return Type of the Method. It defines what the method can return. Void means the Method will not return any value.

main: Main is the name of the Method. This Method name is searched by JVM as a starting point for an application with a particular signature only.

String args[] / String... args: It is the parameter to the main Method. Argument name could be anything.

(b) What is an event in Java? How does Java handle events? Write a program in Java, to capture any event generated by a keyboard.

Refer to Dec.-2005, Q.No.-1(h) (Pg. No.-204)

Program to capture any event generated by a keyboard.

import java.applet.Applet;

```
import java.awt.*;
import java.awt.event.*;
/* <APPLET CODE ="KeyboardEvents.class" WIDTH=300 HEIGHT=200>
</APPLET> */
public class KeyboardEvents extends Applet implements KeyListener
{
    TextArea tpress,trel;
    TextField t;
    public void init()
    {
        t=new TextField(20);
        t.addKeyListener(this);
        tpress=new TextArea(3,70);
        tpress.setEditable(false);
        trel=new TextArea(3,70);
        trel.setEditable(false);
        add(t);
        add(tpress);
        add(trel);
    }
    public void keyTyped(KeyEvent e)
    {
        disppress(e,"Key Typed:");
    }
    public void keyPressed(KeyEvent e)
    {
        disppress(e,"KeyPressed:");
    }
    public void keyReleased(KeyEvent e)
    {
        String charString,keyCodeString,modString;
        char c=e.getKeyChar();
        int keyCode=e.getKeyCode();
        int modifiers=e.getModifiers();
        charString="Key character='"+c+"'";
        keyCodeString="key
code='"+keyCode+"("+KeyEvent.getText(keyCode)+")'";
        modString="modifiers='"+modifiers+
trel.setText("Key
released"+charString+keyCodeString+modString);
```

```

        }
        protected void disppress(KeyEvent e,String s)
        {
            String charString,keyCodeString,modString,tmpString;
            char c=e.getKeyChar();
            int keyCode=e.getKeyCode();
            int modifiers=e.getModifiers();
            if(Character.isISOControl(c))
            {
                charString="key character=(an unprintable control
character)";
            }
            else
            {
                charString="key character='"+c+"'";
            }
            modString="modifiers="+modifiers;

            tmpString=KeyEvent.getKeyModifiersText(modifiers);

            if(tmpString.length()>0)
            {
                modString+="("+tmpString+")";
            }
            else
            {
                modString+="(no modifiers)";
            }

            keyCodeString="key
code='"+keyCode+"("+"KeyEvent.getKeyText(keyCode)+"")';
tpress.setText(s+charString+keyCodeString+modString);
}
}

```

(c) What is exception in Java? What are the basic caused of occurrence of exception? Write a program in Java, to exhibit the concept of exception handling. Support your program with suitable comments.

Refer to Dec.-2005, Q.No.-4(b) (Pg. No.-211) & Refer to Chapter-7, Q.No.-1 (Pg. No.-90)

Note: Question No. 1 is compulsory . Attempt any three questions from the rest of the questions.

Q1. (a) What do you understand by ‘this’ keyword? Explain its usage with an example.

(b) What is an event in Java? Describe different components of an event.

(c) What is an interface? Discuss the utility of interface with suitable example.

(d) What are cookies? Discuss the role of cookies in session tracking.

(e) Compare AWT and Swing components.

(f) Write a Java program, to find factorial of a given number.

(g) What is a layout manager? Explain Border layout with an example.

(h) Explain the Java RMI architecture with the help of a diagram.

Q2. (a) Write a Java program that reads the content of a text file and counts the number of white spaces, number of characters, number of words and number of full stops in the content. Support your code with comments, to describe your logic.

(b) What is multithreading? Explain the steps for creating thread in Java. Also explain the predefined thread priorities in Java.

Q3. (a) What are class and objects? Explain how an object is created in Java? Also explain the role of constructor in Java using Java code.

(b) What is Polymorphism? Explain the difference between method overloading and method overriding with the help of example.

Q4. (a) Explain the life cycle of Applet. List the methods involved in it.

(b) Write a Java program which takes two 3×3 matrices as input and finds the sum of the two matrices. Define constructor for initialization of matrix objects.

Q5. Write short notes on the following (Give example for each):

(a) Message Passing

(b) Java Beans

(c) Strings in Java

(d) Package

Note: Question No. 1 is compulsory . Attempt any three questions from the rest of the questions.

Q1. (a) Write a java program to read the contents of binary file and write it on the standard output.

Ans. Refer to Dec-2010, Q.No.-1(a) (Pg. No.-326)

(b) What is purpose of Serilisation of an object? How does the volatile modifier to a data type affect serialisation?

Ans. Refer to Dec-2010, Q.No.-1(c) (Pg. No.-326)

(c) Justify “Java is secure and architectural neutral language”.

Ans. Refer to Dec-2010, Q.No.-1(d) (Pg. No.327)

(d) What are cookies and session objects? Where do we use them?

Ans. Session object: A HttpSession object (derived from Session object) allows the servlet to solve part of the HTTP stateless protocol problems. After its creation a Session is available until an explicit invalidating command is called on it (or when a default timeout occurs). The same Session object can be shared by two or more cooperating servlets. This means each servlet can track client's service request history.

A servlet accesses a Session using the getSession() method implemented in the HttpServletRequest interface.

getSession() method returns the Session related to the current HttpServletRequest.

Cookies: Using JSDK (Java Servlet Development Kit) it is possible to save client's state sending cookies. Cookies are sent from the server and saved on client's system. On client's system cookies are collected and managed by the web browser. When a cookie is sent, the server can retrieve it in a successive client's connection. Using this strategy it is possible to track client's connections history.

Cookie class of Java is derives directly from the Object class. Each Cookie object instance has some attributes like max age, version, server identification, comment.

(e) Explain the concept of inheritance and polymorphism with an example of each.

Ans. Refer to Chapter-13, Q.No.-31 (Pg. No.-196) and Refer to Chapter-5, Q.No.-5 (Pg. No.-70)

(f) Differentiate between AWT and Swing Components

Ans. Refer to Chapter-9, Q.No.-9 (Pg. No.-126)

(g) What is the difference between notify () and notify All () method.

Ans. Refer to Dec-2010, Q.No.-1(g) (Pg. No.-327)

Q2. (a) What is exception? What are causes of exception? What can be done once an exception is caught? Explain use of finally clause in exceptions handing, with the help of an example.

Ans. Refer to June-2017, Q.No.-3(a) (Pg. No.-419) and Refer to Chapter-7, Q.No.-5 (Pg. No.-95)

(b) Differentiate between following:**(i) Public member and private member**

Ans. Private

Private is the most restrictive access level. A private member is accessible only to the class in which it is defined. You should use this access to declare members that you are going to use within the class only. This includes variables that contain information if it is accessed by an outsider could put the object in an inconsistent state, or methods, if invoked by an outsider, could jeopardize the state of the object or the program in which it is running. You can see private members like secrets you never tell anybody.

To declare a private member, use the private keyword in its declaration. The following class contains one private member variable and one private method:

```
class First {
    private int MyPrivate; // private data member
    private void privateMethod() // private member function
    {
        System.out.println("Inside privateMethod");
    }
}
```

Objects of class First can access or modify the MyPrivate variable and can invoke privateMethod. Objects of other than class First cannot access or modify MyPrivate variable and cannot invoke privateMethod. For example, the Second class defined here:

```
class Second {
    void accessMethod() {
        First a = new First();
        a. MyPrivate = 51; // illegal
        a.privateMethod(); // illegal
    }
}
```

```
}
```

Public

This is the easiest access specifier. Any class, in any package, can access the public members of a class's. Declare public members only if you want to provide access to a member by every class. In other words you can say if access to a member by outsider cannot produce undesirable results the member may be declared public.

To declare a public member, use the keyword public. For example,
public class Account

```
{  
    public String name;  
    protected String Address;  
    protected int Acc_No;  
    public void publicMethod()  
    {  
        System.out.println("publicMethod");  
    }  
}  
  
class Saving_Account  
{  
    void accessMethod()  
    {  
        Account a = new Account();  
        String MyName;  
        a.name = MyName; // legal  
        a.publicMethod(); // legal  
    }  
}
```

(ii) Overloading and Overriding

Ans. Refer to Chapter-13, Q.No.-3 (Pg. No.-172)

(iii) Java application and Applet

Ans. Refer to Chapter-2, Q.No.-1 (Pg. No.-17)

(iv) FOR statement and WHILE statement

Ans. Refer to Chapter-4, (Pg. No.-47– 48)

Q3. (a) What is Multithreading? What is the advantage of multithreading in Java?

Ans. Refer to Dec-2005, Q.No.-4(c) (Pg. No.-212)

(b) What are interfaces in Java? What do you understand by ‘implementing interfaces’? Explain with the help of an example.

Ans. Refer to Chapter-8, Q.No.-3 (Pg. No.-110) and Refer to Dec-2012, Q.No.-1(f) (Pg. No.-352)

(c) What is JDBC? What are the characteristics of JDBC? Write a program to demonstrate how JDBC connection is established.

Ans. Refer to Dec-2006, Q.No.-1(g) (Pg. No.-242) and Refer to Dec-2012, Q.No.-4(b) (Pg. No.-362)

Q4. (a) What is String in Java? Explain any three constructors of String Class. Write a program in Java to find length of a given string and display the string in upper case.

Ans. Refer to Chapter-3, Q.No.-5 (Pg. No.-30) and Refer to June-2009, Q.No.-2(a) (Pg. No.-296)

Java Programming Code to Find Length of String

```
class StringLen
{
    public static void main(String[] args)
    {
        String MyStr = new String(" Practice in programming is always good");
        System.out.println("The length of MyStr is :" + MyStr.length());
        int i = MyStr.lastIndexOf("r") - MyStr.indexOf("r");
        System.out.println("Difference between first and last occurrence of 'r' in MyStr
is :" + i);
    }
}
```

Output:

The length of MyStr is: 39

Difference between first and last occurrence of 'r' in MyStr is: 15

Java String to Uppercase Example:

```
import java.io.*;
public class Test {
    public static void main(String args[])
    {
        String Str = new String("Welcome to TutorialsPoint.com");
        System.out.print("Return Value :");
        System.out.println(Str.toUpperCase());
    }
}
```

Output:

Return Value :WELCOME TO TUTORIALSPOINT.COM

(b) What is servlet? How servlet differs from Applet? Explain the lifecycle of Applet.

Ans. Refer to Dec-2005, Q.No.-2(a) (Pg. No.-206) and Refer to Dec-2006, Q.No.-1(e) (Pg. No.-240)

Q5. (a) Write a Java program to create student class with data member student_name, roll-number and Marks. Define appropriate constructor in the program. Create array often student objects Also display the name and roll-number of the student with highest marks (in the array created).

Ans. Same as June-2018, Q.No.-3(b) (Pg. No.-445)

(b) What is RMI? Explain RMI architecture with suitable diagram.

Ans. Refer to Chapter-11, Q.No.-1 (Pg. No.-137)

(c) What are Java Beans? Discuss the utility of Java Beans. Explain the features of Java Beans.

Ans. Refer to Chapter-11, Q.No.-5 and Q.No.-7 (Pg. No.-142,143)

Utility of Java Beans:

- A Bean obtains all the benefits of Java's "write-once, run-anywhere" paradigm.
- The properties, events, and methods of a Bean that are exposed to an application builder tool can be controlled.
- A Bean may be designed to operate correctly in different locales, which makes it useful in global markets.
- Auxiliary software can be provided to help a person configure a Bean. This software is only needed when the design-time parameters for that component are being set. It does not need to be included in the run-time environment.
- The configuration settings of a Bean can be saved in persistent storage and restored at a later time.