UNIT 1

OBJECT ORIENTED THINKING AND INHERITANCE

PART-A
SHORT QUESTIONS WITH SOLUTIONS

Q1. Define the basic characteristics of object oriented programming.

Answer :

Nov./Dec.-18(R16), Q1(b)

Following are the object-oriented concepts,

1. Everything is an object.
2. To perform a computation, objects communicate with each other and request other objects to carry out the required actions. Communication between objects is achieved by sending and receiving messages. A message contains the request for a particular action to be performed along with the associated arguments that are required to complete the task.
3. Each and every object manage their own memories that store other objects.

Q2. Define class.

Answer :

Model Paper-II, Q1(a)

A class can be defined as a template that groups data and its associated functions. The class contains two parts namely,

(a) Declaration of data variables

(b) Declaration of member functions.

The data members of a class explains about the state of the class and the member function explains about the behaviour of the class. There are three types of variables available for a class. They are,

(i) Local variables

(ii) Instance variables

(iii) Class variables.

Q3. How does a class accomplish data hiding?

Answer :

Nov./Dec.-16(R13), Q1(b)

A class can be defined as skeleton for an object or blueprint for an object or template for an object. A class defines all the variables and methods, an object should have. An object is an instance of a class.

Datahiding is accomplished with the help of encapsulation mechanism. This mechanism is responsible for binding the manipulated code and data. This binding will keep the code and data safe from arbitrary accesses to it by the external code.

Classes accomplish datahiding with the help of public and private access specifiers. If a method is declared as private then, only its class members can access this method. In addition to that only the public methods of the class can access the private members of that class from anywhere within the program. When a class has an interface which is declared as public it will contain data that can be known by external users as well.

Q4. What is the difference between an object and a class?

Answer : Nov./Dec.-16(R13), Q1(a)

Class

Classes are user defined data types and behaves like the built in types of a programming language. When we define a class, we declare its exact form and nature, we do this by specifying the data it contains and the code that operates on data. While very simple classes may contain only code or only data, most real world classes contain both. A class code defines the interface to its data.

Object

Once you define a class it becomes a template for an object. By using this template we can create any number of objects, so object is an instance of a class. Objects are the basic runtime entities in an object-oriented system, they may represent a person, a place, a bank account, a table of data that the program may handle.

Q5. What is an inner class?

Answer : May-19(R16), Q1(c)

A class which is defined inside another class is known as the inner class. Inner class are used inorder to hide itself from other classes of same package. An inner class object can access the implementation of the object from which it is created. Inner classes are capable of inheriting the class members of the outer class. It manipulates the objects of outer class as if it is created in the inner class itself. In addition to this it can also implement interfaces. It does not care if the object inheriting is used or implemented by the outer class. It can also implement from multiple sources.

Q6. Define method overriding.

OR

What is the necessity of overridden methods?

Answer : Nov./Dec.-17(R13), Q1(d)

Method overriding is a phenomenon in which a method in subclass is similar to the method in superclass. The return type and signature of a subclass method matches with the return type and signature of superclass method. The advantage of this method is that, it provides an implementation which is already provided by its superclass. This method can be used for runtime polymorphism. When the method of subclass is called the superclass's method is not referred and it is hidden.

Q7. List the features of Java.

Answer :

The following are the various features of Java,

1. Object-oriented
2. Compiled and interpreted
3. Platform-independent and portable
4. Distributed
5. Robust and secure
6. Familiar, simple and small
7. Multithreaded and interactive
8. Dynamic and extensible
9. High performance.

Q8. Define variable.

Answer :

A variable is the name given to a unit/memory location that stores data value of the variable. The name given to the variable is known as Identifier.

The variable value may change several times during the execution of program. Each variable is associated with its scope that depicts the life time and visibility of the variables.

Q9. Define type casting.

Answer :

'Type casting' is an explicit conversion of a value of one type into another type. And simply, the data type is stated using parenthesis before the value. Type casting in Java must follow the given rules,

1. Type casting cannot be performed on Boolean variables. (i.e., boolean variables can be cast into other data type).
2. Type casting of integer data type into any other data type is possible. But, if the casting into smaller type is performed, it results in loss of data.
3. Type casting of floating point types into other float types or integer type is possible, but with loss of data.
4. Type casting of char type into integer types is possible. But, this also results in loss of data, since char holds 16-bits the casting of it into byte results in loss of data or mixup characters.

Q10. Define one-dimensional array.

Answer : Nov./Dec.-17(R13), Q1(a)

One-dimensional Array

A one-dimensional array is an ordered list of homogeneous data items with one subscript.

Example

Subject[5];

Creation of One-dimensional Arrays

Creation of an array needs three steps to be processed. They are,

(i) Declaration of the array

(ii) Allocation of memory locations for the array

(iii) Initializing the memory location.

Q11. Define an expression.

Answer :

Expression

An expression can be defined as a combination of operators, variables and constants. These are referred as constituents of an expression.

Example

```
c = a + b;
a = 5;
b = a * c;
```

Q12. Differentiate between print( ) and println( ) methods is Java.

Answer : Nov./Dec.-17(R16), Q1(a)

The print( ) statement places the cursor on the same line after printing the output text. Where as the println( ) statement places the cursor on the next line after printing the output text.

Example for print( )

```
System.out.print("SIA")
System.out.print("Group");
```

Output

SIA Group

Example for println( )

```
System.out.println("SIA")
System.out.println("Group");
```

Output

SIA
Group

Q13. What are symbolic constants? Explain with examples.

Answer : Nov./Dec.-17(R16), Q1(b)

Symbolic Constants

Symbolic constants are unique constants which can be appear very frequently in a number of places within the same program.

Symbolic constants are very useful in those programs in which the same value mean different things in different places. For example, the number 101 may mean the total number of passengers in the Bus and the "Bus Number" at another place of the same program. This results in confusion as it is not clear what actually 101 mean. Then issue is resolved with the help of symbolic constants by giving a particular constant which will never change its meaning throughout the program. For example, use the name PASSENGERS to represent the number of passengers in the Bus and BUS NUMBER to represent the Bus number by assigning the constant values at the beginning of the program.

Syntax

Syntax for declaring a Symbolic constant is,

```
final type Symbolic_name = value;
```

Example

```
final int PASSENGERS = 101;
final int BUS_NUMBER = 101;
```

Q14. Explain the use of 'for' statement in Java with an example.

Answer : (Model Paper-I, Q(a) | April/May-18(R16), Q1(b))

for Statement

The for loop defines initializer, condition and iterator sections. It performs iteration through a loop and tests a specific condition before executing the body of a loop. Moreover, a for loop executes only if the condition is true, else it terminates.

Syntax

```
for (initialization ; condition ; iterator)
Code statement.
```

Where,

- initialization expression specifies the initial value.
- condition specifies the condition to be tested at each pass.
- iterator specifies the step taken for each execution of loop body.

Example

The following example illustrates the use of for loop, that prints integers from 0 to 49.

```
for(int i=0; i<50; i=i+1)
{
        System.out.println("The value of i is :" + i);
}
```

In the above code, initially an integer 'i' is declared and initialized to '0'. Then, the for loop checks whether integer 'i' is less than 50. If this condition holds true, then the code within the loop gets executed displaying the value 0. Now, the counter is incremented by one and the process of Iteration continues until 'i' reaches 50.

Q15. Describe finalize( ) method with example.

Answer : Nov./Dec.-17(R13), Q1(b)

finalize( )

An object makes use of many non-Java resources such as file handle or window character font, etc. These resources should be freed before an object is destroyed. Finalization is a mechanism that frees all the resources used by an object. With the help of finalization, one can define specific action that should be taken when an object is about to be destroyed by the garbage collector.

There is a finalize( ) method that performs the finalization. This method is called by the Java run time whenever it is about to destroy an object of that class. Inside the finalize( ) method one can include all those functionalists that must be performed before an object is destroyed. The garbage collector checks dynamically for each object that is not used by any class directly or indirectly and just before freeing an asset the Java run time calls the finalize( ) method on the object.

General Form

```
protected void finalize( )
{
    // body of finalization
}
```

The protected keyword prevents the access to finalize( ) by any code defined outside its class. This method is only called just before an attempt of garbage collection.

Q16. What is the purpose of inheritance? Give example. Nov./Dec.-16(R13), Q1(c)

OR

What is inheritance? Give example.

Answer : (Model Paper-II, Q1(b) | Nov./Dec.-18(R16), Q1(a))

Inheritance

Inheritance can be defined as a mechanism where in one class inherits the features of another class. The class which acquires the features is called the child class or derived class and the class whose features are acquired is called parent class or base class. The parent class contains only its own features, where as the child class contains the features of both parent and child classes. Moreover, an object created for parent class can access only parent class members. However, child class object can access members of both child class and as well as parent class. The child class can acquire the properties of parent class using the keyword 'extends'.

Example

```
class Single
{
    String str;
    Single( )
    {
        str = "Hello";
    }
}
class subclass extends Single
{
    subclass( )
    {
        str = str.concat("World");
    }
}
```

Q17. Define polymorphism.

Answer : May-19(R16), Q1(a)

Polymorphism

Polymorphism is an important concept in OOP. It is derived from the Greek word poly (multiple) and morphism (forms) which together mean multiple forms. It is a method through which an operation/function can take several forms based on the type of objects. An individual operator/function can be used in multiple ways.

Types of Polymorphism

There are two types of polymorphism, they are,

(i) Adhoc polymorphism

(ii) Pure polymorphism.

Q18. Write a short note on Java class libraries.

Answer : Model Paper-I, Q1(e)

The mostly used built-in methods are println( ) and print( ) methods. They are contained in System.out.system which is a class of java by default.

Java language depends upon various built in class libraries that will in turn contain built-in methods which support I/O, string handling, networking and graphics. These classes also support GUI. Therefore, Java is considered as a combination of itself and standard classes.

These classes provide the functionality of Java.

Q19. What is abstract class? Give example.

Answer : April/May-18(R16), Q1(a)

Abstract Class

Java defines abstract classes using the keyword "abstract". The class that doesn't have body is called abstract class. An abstract class can contain subclasses in addition to abstract methods. If the method containing in the subclasses does not override the methods of the base class then they will be considered as abstract classes. So, all the subclass must provide an implementation for all the base class's abstract methods. Abstract classes cannot be instantiated.

Example

```
abstract class Baseclass
{
    abstract display( );
}
class Derivedclass extends Baseclass
{
    void display( )
    {
        System.out.println("Derivedclass");
    }
}
class Demo
{
    public static void main(String args[ ])
    {
        Derived dc = new Derivedclass( );
        dc.display( );
    }
}
```

PART-B
ESSAY QUESTIONS WITH SOLUTIONS

1.1 OBJECT ORIENTED THINKING

1.1.1 A Way of Viewing World – Agents and Communities

Q20. Explain the need for object oriented programming paradigm and also write advantages of OOP.

OR

What are the unique advantages an object oriented paradigm?

(Refer Only Topic: Advantages of OOP)

Answer :

Nov./Dec.-16(R13), Q2(a)

Need of OPP

Traditionally, the structured programming techniques were used. There were many problems because of the use of structured programming technique. The structured programming made use of a top-down approach. To overcome the problems faced because of the use of structured programming, the object oriented programming concept was created. The object oriented programming makes use of bottom-up approach. It also manages the increasing complexity. The description of an object-oriented program can be given as, a data that controls access to code. The object-oriented programming technique builds a program using the objects along with a set of well-defined interfaces to that object. The object-oriented programming technique is a paradigm, as it describes the way in which elements within a computer program must be organized. It also describes how those elements should interact with each other.

Advantages of OOP

Advantages of object-oriented programming over process-oriented programming or procedural programming are as follows,

1. In object-oriented programming, emphasis is on data rather than procedure.
2. Object-oriented programming allows code reusability.
3. In object-oriented programming, data is secure.
4. Object-oriented programming allows extensibility of code.
5. In object-oriented programming, maintenance cost is less.
6. Object-oriented programming focuses on the relationship between programmer and user.
7. Object-oriented programming treats data and functions as a single entity.
8. In object-oriented programming, hierarchical representation of classes is possible.
9. In object-oriented programming, programs are divided into classes and objects instead of functions or procedures.
10. Object-oriented programming can be implemented on real-world problems.
11. Object-oriented programming is based on bottom-up approach.
12. In object-oriented programming, exceptions or errors can be caught at run-time.
13. In object-oriented programming, modularity is achieved.
14. In object-oriented programming, complex problems can be reduced to smaller or manageable problems.

Q21. Write about agents and communities.

Answer :

The structure of an object-oriented program is similar to that of a community, that consists of agents interacting with each other. These agents are also called as objects. An agent or an object plays a role of providing a service or performing an action, and the other members of this community can access these services or actions.

Consider an example, where Rubin and Ruhi are good friends who live in two different cities far from each other. If Rubin wants to send flowers to Ruhi, she can request her local florist, 'Meher' to send flowers to 'Ruhi', by giving her all the information about what variety and quantity of flowers to send and Ruhi's address as well.

Meher works as an agent (or object) who performs the task of satisfying Rubin's request. Meher then performs a set of operations or methods to satisfy the request which is actually hidden from Rubin.

Meher forwards a message to Ruhi's local florist, to send the requested flowers to Ruhi. That florist has a subordinate who arranges the flowers. After the flower arrangement is completed, Ruhi's florist asks a delivery person to deliver those flowers to Ruhi. Ruhi's florist actually obtained those flowers from a wholesaler who in turn got it from a grower who manages a team of gardeners.

The problem of sending flowers to a friend in a different city was solved with the help of many other individuals, forming a community and, each of the individuals worked as agents or objects to solve the problem.

The community of objects or agents helping Rubin in solving the problem of sending flower to her friend Ruhi can be shown in the figure,

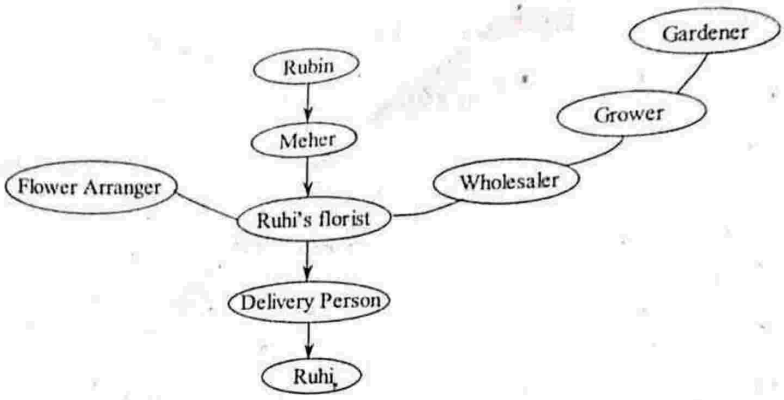


Figure: A Community of Agents Helping Rubin

1.1.2 Messages and Methods, Responsibilities, Classes and Instances

Q22. What are messages and methods? Write about information hiding with respect to message passing.

Answer :

When a message is passed to an agent (or object) that is capable of performing an action, then that action will be initiated in object-oriented programming. The action that is requested is encoded by the message and it is also associated with additional information required to process the request. An object which receives the message sent is called a 'receiver'. When a receiver accepts a message, it means that the receiver has accepted the responsibility of processing the requested action. It then performs a method as a response to the message in order to fulfil the request.

The information hiding with respect to message passing corresponds to the transparency between the client who sends the request and the means through which the request is fulfilled. While programming, using conventional languages, information hiding is regarded as an important feature.

Q23. What is the difference between a message passing and a procedure call?

Answer :

There are two important differences between a message passing and a procedure call. Though, they have a set of well-defined steps that are initiated when a request is placed,

1. **Designated Receiver**

A message contains a designated receiver, as the receiver is an object that receives the message sent. A procedure call does not contain a designated receiver.

2. **Interpretation**

The method selected for execution in response to a message is different for different receivers. At run time, the receiver for the given message will be known, only then, the method to be invoked is found out. This will lead to late binding between the fragment of the code i.e., method and the message.

In conventional procedure calls, there is an early binding of name to the fragment of code, as the binding is carried out during compile time.

Q24. Write about the concepts of Responsibility, classes and instances in object-oriented programming.

Answer :

Responsibility

In object-oriented programming actions are described in terms of responsibilities. A request to perform an action denotes the desired result. An object can use any technique that helps in obtaining the desired result and this process will not have any interference from other object. The abstraction level will be increased when a problem is evaluated in terms of responsibilities. The objects will thus become independent from each other which will help in solving complex problems. An object has a collection of responsibilities related with it which is termed as 'protocol'.

The operation of a traditional program depends on how it acts on data structures i.e., in changing fields within an array or a record. Whereas an object-oriented program operates by requesting data structure to perform a service.

Classes and Instances

A receiver's class determines which method is to be invoked by the object in response to a message. When similar messages are requested then all the objects of a class will invoke the same method.

All the objects are said to be the instances of a class.

For example, if 'Flower' is a class then 'Rose' is its instance.

1.1.3 Class Hierarchies – Inheritance, Method Binding

Q25. Write about class hierarchies/inheritance and method binding.

Answer : **Model Paper-I, Q2(a)**

Class Hierarchies/Inheritance

It is possible to organize classes in the form of a structure that corresponds to hierarchical inheritance. All the child classes can inherit the properties of their parent classes. A parent class that does not have any direct instances is called an abstract class. It is used in the creation of subclasses.

Example

Let, 'Meher' be a florist, but a florist is a more specific form of shopkeeper. Additionally, a shopkeeper is a human and a human is definitely a mammal. But, a mammal is an animal and animal is a material object.

All these categories along with their relationships can be represented using a graphical technique as shown in figure. Each category is regarded as a class. The classes at the top of the tree are said to be more abstract classes and, the classes at the bottom of the tree are said to be more specific classes.

Inheritance is nothing but a principle, according to which knowledge of a category or class which is more general can also be applied to a category or class which is more specific.

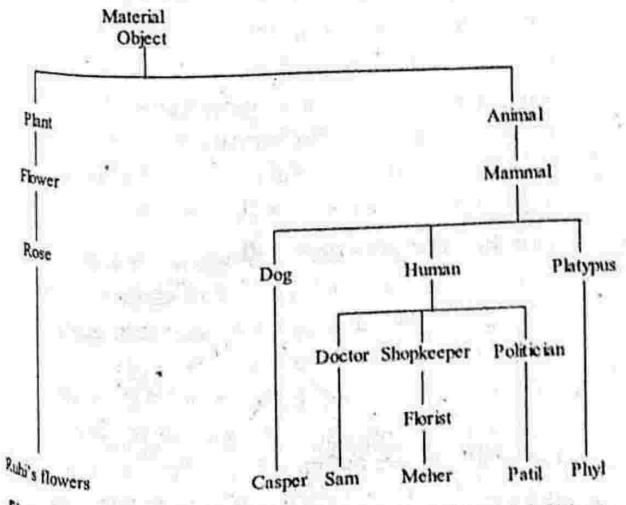


Figure: Class Hierarchy for Different Kinds of Material Objects

Method Binding

When more than two subclasses of a class has a method with the same name then that method is overridden.

The program will find out a class to which the reference is actually pointing and that class method will be binded.

Example

```
class Parent
{
    void print( )
    {
        System.out.println("Hi !");
    }
}
class Child
{
    void print( )
    {
        System.out.println("Hello !");
    }
}
class Bind
{
    public static void main(String[ ] args)
    {
        Child ob = new Child( );
        ob.print( );
    }
}
```

The child's object "ob" will point to Child class print( ) method and thus, that method will be binded.

1.1.4 Overriding and Exceptions, Summary of Object-oriented Concepts

Q26. Write briefly about,

(i) Overriding

(ii) Exceptions.

Answer :

(i) Overriding

For answer refer Unit-I, Q83.

(ii) Exceptions

If the normal flow of the program is disrupted, during the program execution, an event is occurred which is called as "Exception".

Exceptions are generally used to manage errors.

It has an object called exception object that holds the error information. This information includes, the type and state of the program when the error occurred.

To throw an exception, an exception object is created and is then handed over to the runtime system.

Examples

Memory error, Stack overflow etc.

Q27. Explain the fundamental characteristics of OOPs.

OR

What are object-oriented concepts?

Answer :

Following are the object-oriented concepts,

1. Everything is an object.
2. To perform a computation, objects communicate with each other and request other objects to carry out the required actions. Communication between objects is achieved by sending and receiving messages. A message contains the request for a particular action to be performed along with the associated arguments that are required to complete the task.
3. Each and every object manage their own memories that store other objects.
4. All the objects are instances of a class. Whereas, a class is a collection of similar objects (i.e., integers or lists).
5. A class stores within it, the behavior of an object. Thus, identical actions will be performed by those objects that are instances of a particular class.
6. A tree structure with a single root in which the classes are arranged is called an inheritance hierarchy. The offsprings of a class in the tree structure can automatically inherit the memory and behavior of all the instances of a class.

1.1.5 Java Buzzwords

Q28. Write down the properties of Java language.

OR

Write about any six distinct features in Java programming.

Answer :

The people who invented Java wanted to design a language which can provide solution to the problems faced in modern programming. The following are the various features of Java,

1. Object-oriented
2. Compiled and interpreted
3. Platform-independent and portable
4. Distributed
5. Robust and secure
6. Familiar, simple and small
7. Multithreaded and interactive
8. Dynamic and extensible
9. High performance.

A discussion on each of them in detail is given below.

1. **Object-oriented**

Almost everything in Java is in terms of an object. Complete program code and data reside within objects and classes. Java is said to be a true object-oriented language. It comes with an extensive set of classes, arranged in packages, which we can use in the program by inheritance. In Java, the object model is not only very simple but also very easy to extend.

2. **Compiled and Interpreted**

Generally a computer language will be either compiled or interpreted but Java combines both these approaches. In the first stage, Java compiler translates source code into byte code instructions. But byte code instructions are not machine instructions and therefore Java interpreter generates machine code in the second stage which can be directly executed by the machine which is running the Java program. In this way Java is not only a compiled but also an interpreted language.

3. **Platform-independent and Portable**

Java supports portability i.e., Java programs can be moved from one system to another, anywhere and anytime easily. Changes and upgrades in processors, operating systems etc., will not force any changes in Java programs. Java programs can be run on any platform.

4. **Distributed**

Java is a distributed language. It not only has the ability to share data but also programs. Java applications can open and access remote objects on internet very easily. This enables many programmers residing at different locations to work on a single project.

5. **Robust and Secure**

Java provides many safeguards in order to ensure reliable code. Java has strict compile time and run time checking for data types and it also incorporates the concept of exception handling, that captures errors and eliminates the risk of system crash. Because of this, Java is said to be robust language.

For a language which is used for programming on internet, security is a very important one to consider, Java not only verifies the memory access but also ensures that no viruses communicate with an applet.

6. **Familiar, Simple and Small**

Java is a familiar language. It is modelled on C and C++ languages. Java uses several constructs of C and C++ and therefore Java code looks like a C++ code. In fact, Java is a simplified version of C++. Java is not only small but also a simple language. Many features of C and C++ are not included in Java because they are redundant. Java does not use pointer, goto statement, operator overloading, multiple inheritance etc.

Multithreaded and Interactive

Java supports multithreading i.e., it is not necessary for an application to finish one task before starting another. In this way we can handle several tasks simultaneously. The Java runtime comes with tools which support multiprocessor synchronization and construct interactive systems running smoothly.

Dynamic and Extensible

Java is a dynamic language, which is capable of dynamically linking in new class libraries, methods and objects. It can also find the type of class through a query, thus making it possible to either dynamically link or abort the program. Java also supports extensibility. It support functions written in other languages like C and C++. This makes the programmers to use the efficient functions available in these languages. These functions are called native methods which can be linked dynamically at runtime.

High Performance

Because of using intermediate byte code, Java performance is excellent for an interpreted language. In order to reduce overhead during runtime, Java architecture is designed carefully. Multithreading improved the overall execution speed of Java programs.

1.1.6 An Overview of Java

Q29. Explain the basic concepts of object oriented programming.

Answer :

Object oriented programming is a technique that builds the program using the objects along with a set of well defined interfaces to that object. It describes the way in which elements within a computer program must be organized and how these elements can interact with each other. Object oriented programming uses bottom-up approach and also manages the increasing complexity.

Data Abstraction

Abstraction is one of the important element of OOPS for managing complexities. It is a mechanism that hides the implementation details and shows only the functionality. For example, while performing transaction on ATM, the end user just type the password and accepts the services provided to him. But the user is unaware of the internal processing. Thus, an abstraction is one that mainly focuses on what the object does but not how it does.

Encapsulation

Encapsulation is a mechanism of binding data members and corresponding methods into a single module called class inorder to protect them from being accessed by the outside code. An instance of a class can be called as an object and it is used to access the members of a class. In encapsulation, objects are treated as 'block boxes' since each object performs specific task.

The data and functions available in a class are called as members of a class. The data defined in the class are called as member variables or data members and the functions defined are called as member functions.

The main idea behind the concept of encapsulation is to obtain high maintenance and to handle the application's code.

Java support encapsulation mechanism by providing access specifier that defines the scope and the access permission of a class member or method. The access specifiers includes: public, private and protected. A class data member or method if declared as private then they can be accessed within its own class only. But if they are declared as protected then they can be accessed within its own class, sub-classes and in all the other classes residing in the same package. However a private data member of a class can be made to accessible from any other class by declaring it as 'public'. The public access specifier makes the data member to be available out-side the class.

Inheritance

Inheritance can be defined as a mechanism of acquiring features from one class called parent class to another class called child class. Parent class is also known as base class or super class and child class is known as derived class or sub class. The parent class contains only its own features whereas the child class contains the features of both parent as well as the child classes. Moreover, an object created for parent class can access only parent class members. However, child class object can access members of both child class as well as parent class. The child class can acquire the properties of parent class using the keyword 'extends'.

Polymorphism

Polymorphism is one of the OOPs concept. It can be used to design and implement systems that can be extensible more easily. 'poly' means many and 'morph' mean forms i.e., polymorphism is the ability to take more than one form. There are two types of polymorphism supported by Java. They are static polymorphism and dynamic polymorphism.

Q30. What is meant by byte code? Briefly explain how Java is platform independent.

Answer : Nov./Dec.-17(R16), Q2(a)

Byte Code

A Bytecode is an intermediate representation of java source code. It is also called as compiled format of java source code. It consist of optimized set of instructions which are usually executed by Java run time system on a special machine called Java Virtual Machine (JVM). A JVM is a bytecode interpreter. It performs line by line execution of bytecode.

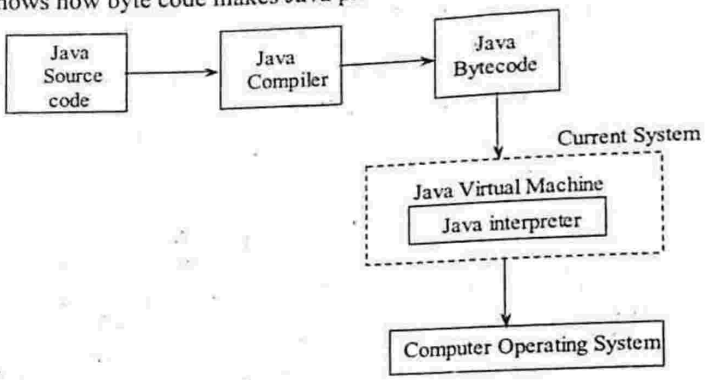
A Bytecode can be transferred across any network and then can be executed by Java Virtual Machine (JVM). They generally have .class extension.

A bytecode usually comprises of one or two bytes that represent the instructions.

Java is Platform Independent

Java is platform independent, meaning that a Java program can run on any operating system and on any machine architecture. The basic asset for Java to be platform independent is byte code. Basically a Java program does not run on the computer we are presently using, it runs on a machine called Java virtual Machine (JVM). This JVM is emulated i.e., embedded inside the computer we are using by means of a program.

The following figure shows how byte code makes Java platform independent.



Figure

Generally any language compiler translates source code of a program directly to machine code, this is the reason why other language programs can run only on specified operating systems. Consider the figure, the Java source code is translated into an intermediate code called byte code. A virtual machine called Java virtual machine is already embedded in the computer system we are running by means of a program. The JVM consists of an interpreter called Java Interpreter. The byte code, which is the output of Java compiler is deciphered by the Java interpreter and finally program is executed. As the Java program we run is in byte code instead of machine code it is not effected by differences in the hardware architecture of different computer systems. An interpreted Java code runs slowly when compared to an equivalent native machine instructions.

Java does not support pointers because, pointers are unsafe. They corrupt the firewall between the host computer and Java execution environment. Moreover, when the Java programmers are confined to the execution environment, they will never require pointers and will never get benefited in using them. Java supports reference types instead of pointers. Because, with reference types, the firewall cannot be corrupted, since the references cannot be manipulated as that of pointers.

Q31. Write the significance of Java Virtual Machine.

Answer : Nov./Dec.-18(R16), Q2(b)

Java Virtual Machine

Java Virtual Machine (JVM) is the most important part of Java technology. The JVM and Java API together, forms a platform for all Java programs to run. JVM and Java API is also known as Java runtime system.

Basically, JVM acts like an interpreter for Java byte code, which is an intermediate code. The JVM loads the Java classes, verifies the byte code, interprets and executes it. Additionally, it provide functions like security management and garbage collection. The figure shows the internal architecture of JVM.

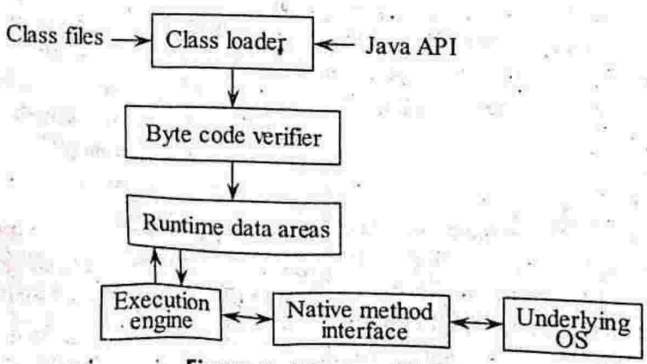


Figure: Architecture of JVM

Class Loader

1. It is a mechanism that perform loading of classes and interfaces into JVM. When a program attempts to invoke a method of certain class, then JVM checks whether that class is already loaded, if not, the JVM uses class loader to load the binary representation of that class.

Byte Code Verifier

2. After loading the class, the byte code verifier, checks whether the loaded representation is well-formed, whether it follows the semantic requirements of Java programming language and JVM. Also checks whether the code contains proper symbol table. If a problem occurs during verification then an error is thrown.

Runtime Data Areas

3. These are special memory areas which JVM maintains to temporarily store byte codes and other information like loaded class files, objects, parameters to methods, return values, local variables and results etc.

Execution Engine

4. It is a mechanism which is responsible for executing instructions contained in the methods of loaded classes.

Native Method Interface

5. If any Java program calls a non-Java API method or platform-specific native method then, the code of these native methods is obtained from underlying OS through the use of native method interface.

Example

The given fraction of program contains several errors, the corrected program,

```
Line 1: int[ ] a = 10, 20, 30, 40, 50;
Line 2 : int i = 4;
Line 3 : System.out.println(a[i] = i = 2);
```

In line 1, the array 'a' is initialized to five integer values 10, 20, 30, 40 and 50.

i.e.,

In line 2, variable i is initialized to 4 i.e.,

a[0]	a[1]	a[2]	a[3]	a[4]
10	20	30	40	50

In line 3, at first, the chain of assignments is executed. That is, value of i = 2 is 2, which is then assigned to a[i] i.e., a[i] = 2. But i = 2. Thus, the value of a[2] = 2 is 2.

i.e., a[i] = i = 2

a[i] = 2

a[2] = 2

Thus, the output of the given fraction of program is 2 and the updated array 'a' is,

a[0]	a[1]	a[2]	a[3]	a[4]
10	20	30	40	50

Q32. Describe the structure of a typical Java program with an example. Nov./Dec.-18(R16), Q2(a)

OR

Explain different parts of a Java program with an appropriate example.

Answer : Nov./Dec.-17(R16), Q3(a)

Java Program

A Java program consists of one or more classes. Only one of these classes defines the main( ) method. A class consists of data and methods that operate on the data of a class. The general syntax of a Java program is shown below.

Documentation

↓

Package Statement

↓

Import Statements

↓

Interface Statements

↓

Class Definitions

↓

Main Method class definition

{

main( ) method definition

}

Figure: General Syntax of a Java Program

Parts of Java Program

The different parts of java program are given as follows,

1. Documentation

This section consists of a set of comments about the program including the name of the program. This section is suggested as it helps in understanding the program.

2. Package Statement

This is the first statement in every Java program, if needed. This statements tells the compiler that the classes defined here belongs to this package. This statement is optional.

Example: package myPack;

3. Import Statement

The import statement tells the interpreter to include the classes from the package defined. This is the next statement after the package declaration but should be written before defining a class. There may be a number of import statements. This statement is optional.

Example: import java.io.*;

4. **Interface Statements**

The interface statement defines method declaration without body for the subclasses to provide implementation for them. This is optional, it is used only if we want to implement multiple inheritance feature in our programs.

5. **Class Definition**

This section consists of a number of class definitions where each class consists of data members and methods. The number of classes required are based on the complexity of the problem.

6. **Main Method Class**

This is an essential section of a Java program. Every Java program must have a class definition that defines the main( ) method. This is essential because main( ) is the starting point for running Java stand-alone programs. The main( ) method instantiates the objects of various classes and establishes the communication between them. The program terminates on reaching the end of the main( ) method.

Example

```
// A first Java program
// File name : Simple.java
class Simple
{
public static void main(String args[ ])
{
System.out.println("A simple Java program");
}
}
```

The above is a simple Java program.

Now, let's see the process of compiling and running this program. In order to compile this program, we should execute the compiler javac, specifying the name of the source file on the command line as shown below.

```
c:\> javac Simple.java
```

The javac compiler creates a file known as Simple.class which contains the bytecode version of the program. The output of javac is not the code which can be executed directly. Therefore, to actually run the program, we should use the Java interpreter, called Java. We should pass the class name Simple as command-line argument as shown below.

```
c:\> Java Simple
```

After running the above program we get the following output,

Output

A simple Java program.

Q33. Discuss about using blocks of code.

Answer :

In Java, two or more statements can be grouped into blocks of code. It is also called as code blocks. Statements can be grouped by using opening and closing curly braces. After the code blocks are created, they can be used anywhere logically as single statement. Consider the below code,

```
if(a>b)
{
    a = b;
    b = 0;
}
```

In the above code, all the statements in if block become the logical unit. All those statements are interrelated.

Example

```
class Block
{
    public static void main(String args[ ])
    {
        int i, a;
        a=10;
        for(i=0; i<5; i++)
        {
            System.out.println("i value:"+i);
            System.out.println("a value:"+a);
            a=a - 2;
        }
    }
}
```

In the above program, for loop contains the block of code. These statements get executed for every iteration of loop. The purpose of it is to create inseparable units of code logically. It even has additional properties and uses.

Q34. List and explain the lexical issues of java.

Answer :

The lexical issues of java are defined as follows,

1. **Whitespace**

Java does not require any special indentation rules because it is a free form language. Whitespace is a space, tab are new line in Java. For example, a program can be written in one line completely until there is a whitespace character among the tokens.

2. **Identifiers**

Identifiers are the names assigned to classes, variables and methods. The names can contain uppercase letters, lower case letters, numbers and underscore or dollar sign. But they must not begin with number.

Examples: Count, test, x2

Literals

A literal means the representation of constant value in Java.

Example: 50, 32.8, 'A', "SIAGROUP Publications".

Comments

Comments in Java are of three types. They are single-line, multiline and documentation comment. The third type of comment is used to generate HTML file which contain documents of your program. It starts with /* and ends with */.

Separators

The separators are the characters that are used for separating. Mostly used separator is ;(semicolon) in Java. Other separators include ( )(parenthesis), { }(braces), [ ](brackets), ,(comma), .(period) and : :(colons).

Java Keywords

Keywords are reserved words that have special meaning. They defines specific functionality of the language. They are written in lower case letters and cannot be used as variable name, class name and method name. Java has 50 keywords. Some of them are listed below,

1.	Abstract	2.	Assert
3.	Boolean	4.	Byte
5.	Break	6.	Case
7.	Catch	8.	Char
9.	Class	10.	Const
11.	Continue.	12.	Default
13.	Do	14.	Double
15.	Enum	16.	Extends
17.	Final	18.	Finally
19.	Float	20.	For
21.	Goto	22.	If
23.	Implements	24.	Instance of
25.	Interface.		

The keywords of java are combined with separators, operators and form the foundation of Java. They must be used as identifiers. In addition to this, they should not be used as names for variables, methods or classes.

1.1.7 Data Types, Variables and Arrays

Q5. List the primitive data types available in Java and explain.

Answer : April/May-18(R16), Q3(a)

Data types are mainly used in Java since, it is a strongly typed language. In java, compiler evaluates all the type of variables, values and expressions in order to maintain type compatibility. Meanwhile, errors are reduced and reliability is increased. Primitive data type is one of the simplest data type that consist of a single value.

Java supports eight types of primitive data types which are grouped into four types as shown below,

(i) Integer data type

(ii) Floating-point data type

(iii) Character data type and

(iv) Boolean data type.

(i) Integer Data Type

The integer type can represent the signed integer values that may be positive or negative. Java does not allow unsigned integer values since, they are not required. This data type itself consists of four different types. They are as follows,

(a) byte

(b) short

(c) int

(d) long.

(a) byte

The byte data type is used to maintain integer values that have a size of 1 byte i.e., 8 bits. The range of byte data type is from –128 to 127.

Syntax: byte variable_name;

Example

```
byte x = 3;
```

(b) short

The short data type represents the signed integer values and the size of this data type is 2 bytes i.e., 16 bits. The range of this data type is from –32, 768 to 32, 767.

Syntax: short variable_name;

Example

```
short a, b;
a = 2376;
```

(c) int

The int data type is most preferable among all the integer data types. Since, it can control index of arrays and loops.

The size of this data type is 4 bytes i.e., 32 bits and the range is from –2,147,483,648 to 2,147,483,647.

Syntax: int variable_name;

Example

```
int a, b;
a = 2376789;
```

(d) long

The long data type is a signed 8 byte (64 bit) which has a range from – 9,223,372,036,854,775,808 to 9,223,372,036,854,775,807. Generally, this data type is used to manage the integer values that have a range greater than int.

Syntax: long variable_name;

Example

```
long x, y;
x = 1234567890L;
```

Program

```
class IntegerExample
{
    public static void main(String[ ] args)
    {
        byte x;
        short y;
        int z;
        long p;
        x = 2;
        y = 23456;
        z = 117438;
        p = 732227363054L;
        System.out.println("The value of byte data type is:" + x);
        System.out.println("The value of short data type is:" + y);
        System.out.println("The value of int data type is:" + z);
        System.out.println("The value of long data type is:" + p);
    }
}
```

(ii) Floating-point Data Type

Floating-point type can be defined as a data type which indicates the fractional values. The two types of floating-point data types are as follows,

(a) float and

(b) double.

(a) float

The float data type is used to represent single-precision numbers and the size of this data type is 4 bytes i.e., 32 bits. The maximum value of float literal is approximately 3.4×10^{38}.

Syntax

```
float variable_name;
```

Example

```
float x = 3.14F;
```

(b) double

Double data type is used to represent double-precision numbers and the size of this data type is 8 bytes i.e., 64 bits. The largest value of double literal is approximately 1.8×3^{308}. The double data type is frequently used in java since, the math functions in Java class library uses double values.

Syntax

```
double variable_name;
```

Example

```
double x;
```

Program

```
class FloatExample
{
    public static and main(String [ ] args)
    {
        float x;
        double y;
        x = 3.14F;
```

```
        y = Math.sqrt(x);
        System.out.println("The float value of x is:" + x);
        System.out.println("The double value of y is:" + y);
    }
}
```

(iii) Character DataType

The character (i.e., char) data type represents characters and the size of this data type is 16-bits (2 bytes). Java language uses 2 bytes since it supports unicode characters. Unicode characters can be defined as a set of characters that indicates all the characters of human languages. The character data type stores unsigned 16-bit characters with a range of 0 to 65,536. The character value must be enclosed in single quotes.

Syntax

```
char variable_name;
```

Example

```
char x;
x = 'n';
```

Program

```
class characterExample
{
    public static void main(String[ ] args)
    {
        char y;
        y = 'N';
        System.out.println("The value of y is:" + y);
        y--;        //The char value is decremented
        System.out.println("The new value of y is:" + y);
        y = 82;     //The char data type can be assigned with integer
        System.out.println("The another value of y is:" + y);
    }
}
```

(iv) Boolean Data Type

The boolean data type indicates whether the value of an expression is either true or false. Therefore, a variable or expression which is declared as boolean type can use these two keywords.

Syntax

```
boolean variable_name;
```

Example

```
boolean a;
a = true;
    or
a = false;
```

Program

```
class BooleanExample
{
    public static void main(String[ ] args)
    {
        boolean a;
        int x, y, z;
        x = 10;
        y = 20;
```

```
            z = x > y;
            a = false;
            if(a)
            {
               System.out.println("a is :" + a);
               System.out.println("x is not greater than y");
            }
            z = y > x;
            a = true;
            if(a)
            {
               System.out.println("a is :"+a);
               System.out.println("y is greater than x");
            }
         }
      }
```

The above program can print the boolean values true/false on to the standard output device using println( ). Next, the control statement 'if' can be managed by using the boolean variable 'a' directly. However, there is no need to use the statement, which is shown below,

```
if(a == true)
{
   //Statements
}
```

Q36. Discuss about literals in Java.

Answer :

In Java, the different types of literals are,

1. Integer literal
2. Floating-point literal
3. Boolean literal
4. Character literal
5. String literal.

1. Integer Literal

An integer literal is a whole number value. That can be any decimal, octal and hexadecimal value. Decimal numbers have base 10 and they do not have a leading zero. For example, 1, 54, 196 are said to be the decimal numbers.

Octal values have base 8 and they range from 0 to 7. They are denoted by a leading zero. Example 00,06 and 07 are octal values. It is important to note that 09 is not an octal value. Hexadecimal values have the base 16 and they range from 0 to 15. The hexadecimal numbers 10 to 15 are represented by alphabets A to F respectively.

They are denoted by a leading zero-x, (0x or 0X).

An integer literal creates an *int* value that can be assigned to a byte, short, long or char.

A long literal is denoted by appending a *l* or L to the literal. For example, 0x7fffffffffffffffL or 9223720368-54775807L is said to be the largest long literal.

2. Floating-point Literal

The number that contains a decimal value followed by a fraction component is called floating-point literal. It can be expressed in two notations,

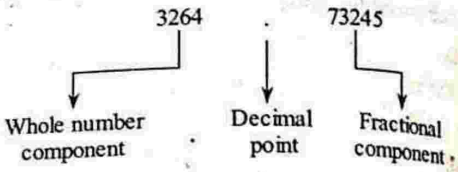
(i) Standard notation

(ii) Scientific notation.

(i) Standard Notation

Standard notation represents the floating-point literal in the form of whole number component and a fractional component separated by a decimal point.

Example



(ii) Scientific Notation

Scientific notation consists of a standard floating-point number followed by a suffix. Suffix specifies a power of '10' by which a number should be multiplied.

The exponent is denoted either by '*E*' or '*e*' followed by a negative or positive decimal number.

Example

7.324E32

3e+1000

In Java the default float literal is double. A double literal can also be explicitly specified by appending a D or d to the constant.

3. Boolean Literal

Boolean literals can only take two logical values 'true' and 'false'. Both of them cannot be converted to numerical representation.

In Java, boolean values (i.e., true and false) can only be assigned to boolean variables and they can be used in expressions involving the boolean operators.

4. Character Literals

In Java, character literals are indices into the Unicode character set. These literals are represented within single quotes.

They are 16-bit values which can easily be converted into integers and can be manipulated using integer operators such as addition, subtraction etc.

They use the '\' (Slash) for entering the ASCII characters which cannot be entered directly such as new line character '\n' and tab character '\t'.

They allow us to enter the value of a character in both octal and hexadecimal notation.

Example

For 'a', an octal notation is '\141' and hexadecimal notation is '\u0061'.

String Literals

String literal consists of a sequence of characters enclosed within double quotes.

For example, "Good Day" is a string literal. String literal must be written on a single line because no line-continuation escape sequence is provided by Java. The working of octal notation, hexadecimal notation and escape sequences in string literal is same as that of character literals.

Q37. What is a variable and explain the declaration and initialization of variable?

Answer :

Variable

A variable is the name given to a unit/memory location that stores data value of the variable. The name given to the variable is known as Identifier.

The variable value may change several times during the execution of program. Each variable is associated with its scope that depicts the life time and visibility of the variables.

Declaration of a Variable

A variable can be declared inorder to avoid the confusion to the compiler. The declaration of a variable is preceded by its data type so that it can accept data values of its associated type.

Syntax

```
datatype variable_name;
```

Here datatype is the type of variable such as int, double, char and variable_name indicates the name of the variable.

Example

```
int x;
double y;
```

When a variable is declared, an instance is created for its data type which identifies the capability of variable. This is because of means that when a variable is declared as int then it cannot store the values of boolean data type. This, strong type checking is supported by Java.

Initialization of a Variable

Initialization of a variable can be defined as a process of assigning a value to the variable. This can be done directly during the declaration of a variable or after the declaration of variable. The syntax for initializing a variable is as follows,

Syntax

```
datatype variable_name = value;
datatype variable_name;
variable_name = value;
```

Example

```
int x = 10;
float y = 1.5F;
char ch;
ch = 'N';
```

Multiple variables that are declared with similar type can be initialized simultaneously. This can be done by using comma operator.

Example

```
int x, y, z = 8, p = 9;
```

Dynamic Initialization of a Variable

Dynamic initialization of a variable can be defined as a process of initializing the variable at run-time i.e., during execution of the program. This can be performed by assigning a valid expression to required variable.

Program

```
class VariableExample
{
    public static void main(String[ ] args)
    {
        int x = 10, y = 20;
        int z;
        z = x + y;
        System.out.println("Addition of x, y = " + z);
        z = x – y;
        System.out.println("Difference between
                                    x, y =" + z);
    }
}
```

In the above program, two variables *x*, *y* are declared and initialized with 10 and 20 respectively. Another variable *z* is declared and can be initialized dynamically. After the computation of addition operation, *z* is initialized with 30 and after subtraction, *z* is initialized with –10. This depicts that the value of a variable may vary during execution of the program.

Q38. Explain the scope and lifetime of a variable with example.

Answer : Model Paper-I, Q2(b)

Scope of a Variable

The scope of a variable can be defined as a process of localizing the variable and preventing it from undesired modifications. The scope of a variable is limited to the block of code. A block of code can be defined as a set of statements enclosed within opening and closing curly braces ({...}). The variables that are declared inside a block are not visible from outside of the block. This means, the scope of these variables is only within the block. The scope of a variable begins with opening braces of a method and if the method itself contain parameters, their scope is used only within the method. Scope of a variable can be nested. If a variable is declared inside a block which itself is defined in another block. Then the scope of outer variable is visible to the inner block. However, inner block variable can be invisible to outer block.

A variable can be declared at any point and they are valid only after the declaration. Therefore, a variable which can be declared in the beginning of method can be accessible throughout the block. Whereas, a variable which can be declared at the end of block is not used since the block of code cannot access it. A variable is destroyed when scope of the variable is completed.

Program

```
class ScopeExample
{
    public static void main(String[ ] args)
    {
        int a;
        a = 20;
        if(a = = 20)
        {
            int c = 30;
            System.out.println("The values of a and c are:" + a + " " + c);
            a = c + a;
        }
        c = 80;//Here, an error(c not declared) is raised, since the scope of c is completed.
        System.out.println("a = 8" + a);
    }
}
```

The variable '*a*' can be used in main() method and as well as in if block, since it is declared in outside the block. However, the value of '*c*' can be accessed only within if block since, it is declared in inner scope.

Lifetime of a Variable

The lifetime of a variable can be defined as the time period in which the variable is present in computer memory. The variables can be accessed during the execution of program and restricted to its scope. When a variable is declared in a block, its lifetime will be started and completed at the end of the block.

If a variable is declared and initialized with a value, then the value is re initialized every time when the declaration statement is executed.

Program

```
class LifetimeExample
{
    public static void main(String[ ] args)
    {
        int a;
        a = 0;
        while (a < 5)
        {
            int b = 2;
            System.out.println("The value of b is" + b);
            b = 10;
            System.out.println("The value of b is" + b);
            a++;
        }
    }
}
```

In the above program, the variable *b* is initialized with a value 2 and re-initialized with 10. If the loop is repeated, then *b* again is re-initialized with 2 and prints the value 2. Again, the variable *b* initialized with 10. This process is repeated until the specified condition is not satisfied (i.e., a<5) and print the values 2 and 10 alternatively.

Q39. Describe about type conversion. Also explain how casting is used to perform type conversion between incompatible types.

Answer :

Type Conversion

"Type conversion" refers to conversion of data from one form to other. It is one of the most important aspects of 'C' programming. Since programmers are prone to commit errors, care should be taken (while writing programs) that the code written, obeys the principles of type conversion. For example, consider that there are two integer variables namely $a = 2$, $b = 3$. In general terms one would expect that the solution obtained by performing 'b/a' would yield 1.5. But, this is not the case, when the same logic is applied in the programs. This is because, as both the variables a and b are declared integers, hence 'C' compilers, considers the result of 'b/a' as integer, eventually discarding the decimal value. Hence, proper type conversion is necessary to ensure accurate results. Consider an example for type conversion which can be shown as follows,

```
int x;
float y;
x = 5;
y = x;  //Here, an integer value 5 is assigning to float
```

If compatible types are assigned to variables, the right hand side expression is assigned directly to left hand side expression. However, it is not possible to convert all types since java support strict type checking. Similarly, all implicit type conversions are not supported. An automatic type conversion can be done only if the following conditions are satisfied,

- The data type of the variable and data must be compatible.
- The data type of destination (left hand side) must be large in size compared to the source (right hand side) type.

The int type can hold byte, short values since it is higher than these types. Whereas, long value cannot be assigned to int. The integer types and floating point types are compatible with each other. However, these numeric types cannot be compatible with boolean or char. Similarly, char and boolean are also not compatible. As a matter of fact, it is possible to assign integer constant to char type.

Example

Consider an example program of converting long to double which can be performed automatically.

```
class TypeCastExample
{
    public static void main(String[ ] args)
    {
        float l;
        double d;
        l = 1234567L;
        d = l;
        System.out.println("The value of long type is" + l + "and the value of double type is" + d);
    }
}
```

Here, the long type can be converted automatically to double. However, double cannot be converted into long type since this conversion is not compatible.

Type Casting

'Type casting' is an explicit conversion of a value of one type into another type. And simply, the data type is stated using parenthesis before the value. Type casting in Java must follow the given rules,

1. Type casting cannot be performed on Boolean variables. (i.e., boolean variables can be cast into other data type).
2. Type casting of integer data type into any other data type is possible. But, if the casting into smaller type is performed, it results in loss of data.
3. Type casting of floating point types into other float types or integer type is possible, but with loss of data.
4. Type casting of char type into integer types is possible. But, this also results in loss of data, since char holds 16-bits the casting of it into byte results in loss of data or mixup characters.

The general form of type casting is as follows.

Syntax

(target-type) expression;

Here, target-type is the specification to convert the expression into required type.

Example

```
float a, b;
int c = (int) (a + b);
```

The result of float type variables can be converted in to integer type explicitly. The parentheses for the expression $a + b$ is necessary. Otherwise, the variable can be converted into int but not the result of $a + b$. The casting of double to int is necessary since, they are not compatible.

Furthermore, Java performs assignment of value of one type to a variable of another type without performing type casting. Here the conversion is performed automatically which is referred to as automatic type conversion. It can take place when destination type hold sufficient precedence to store source value.

Example

```
byte  b = 50;
int   a = b;
```
} valid statement

The assignment of smaller data type to larger data type is referred to as widening or promotion and the assignment larger type value to smaller is referred to as narrowing (it results in data loss).

While performing the conversion between two incompatible type, the use of word cast is necessary. It is an explicit type conversion.

Syntax

(target_type) value target type.

It is a desired type to convert the given value.

While casting the types, it is necessary to prevent the narrowing conversion which causes data lost. If the long value is converted to short then there may be a chance to loss of data. Similarly, when a floating-point value 3.25 is casted to an int value, then the value will be converted to 3 and remaining data 0.25 will be lost. It is better to avoid narrowing conversions to prevent loss of data.

Program

```
class CastingExample
{
    public static void main(String[ ] args)
    {
        char c;
        int x;
        double a, b;
        byte p;
        a = 8.0;
        b = 9.0;
        x = (int) (a + b);
        System.out.println("The integer value of
                                (a + b) is" + x);
        x = 80;
        p = (byte) x;          //Byte can store 80.
                               //So no data loss.
        System.out.println("The byte value is" + p);
        p = 97;                //It is the ASCII code of a
        c = (char) b;
        System.out.println("The char value is:" + c);
        //The ASCII code of 97 is 'a'. It can be
        //assigned to c.
    }
}
```

Q40. Explain how type conversions takes place in expressions.

Answer :

The type conversion takes place in expressions, if the value of an intermediate result exceeds the range of one or the other operand. This type promotion is implicitly. Consider the following example.

```
int result = x * y / z;
```

If x, y and z are of **byte** then the intermediate result of x * y may exceed the range of either of its byte operands. In this case Java promotes the byte or short value into Int type automatically.

However, the automatic type promotions are very confusing and may cause compile time errors. The errors occurs if the Java promotes the operands to one type and the result is of another type.

For example,

```
byte  x = 100, y;
      y = x * 5;
```

This statement is even though logically correct but causes compile time error since by multiplying byte value of x with 5 exceed the byte range and will be promoted to int type by Java while evaluating the expression. Since the resultant type is int and we are storing in byte variable results in compile time error.

In such case where the resultant type may exceed we must use explicit casting as shown below.

```
byte  x = 100, y;
      y = (byte) (x * 5);
```

Rules for Type Promotions

1. In evaluating expressions byte or short types are promoted to 'int'.
2. If one of the operands of an expression is of,
   - ❖ Float type then entire expression is promoted to **float.**
   - ❖ Double type then entire expression is promoted to **double.**
   - ❖ Long type then entire expression is promoted to **long.**

Example

The following example illustrates the type promotion in expressions.

```
class PromotionExample
{
public static void main(String args[ ])
{
byte a = 30;
char b = 'Z';
int c = 450;
float d = 32.67f;
double e = 1.260;
double expr = (c*d)/(b*c) + (e*a);
System.out.println("Expression =" + expr);
}
}
```

Q41. What is an array? What are different types of array? List out the advantages of using arrays.

OR

What is an array? How do you declare the array in Java? Give Examples.

(Refer Only Topics: Array, Declaration of the Array)

Answer : Nov./Dec.-18(R16), Q3(b)

Array

An array is defined as a set of homogeneous data items, clubbed under a single name. Every member in the array is assigned an index number or subscript.

Types of Arrays

The three different types of arrays are,

1. One-dimensional array
2. Two-dimensional array
3. Multi-dimensional array.

1. One-dimensional Array

A one-dimensional array is an ordered list of homogeneous data items with one subscript.

Example

Subject[5];

Creation of One-dimensional Arrays

Creation of an array needs three steps to be processed. They are,

(i) Declaration of the array

(ii) Allocation of memory locations for the array

(iii) Initializing the memory location.

(i) Declaration of the Array

An array can be declared in two ways.

1st way

datatype arrayname[ ];

2nd way

datatype[ ] arrayname;

Examples

int subject[ ];

int[ ] subject;

Here an important point to note is size of the array should not be given during declaration.

(ii) Allocation of Memory Locations for the Array

Memory locations are dynamically allocated with the aid of 'new' operator.

Syntax

arrayname = new datatype[size];

Example

subject = new int[5];

Both declaration and memory allocation can be done in a single step as follows,

Syntax

datatype arrayname = new datatype[size];

Example

int subject = new int[5];

The first two steps are pictorially depicted below.

Step 1: datatype arrayname[ ];

Step 2: arrayname = new datatype[size];

Example

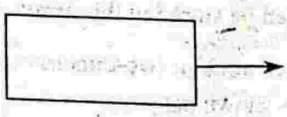
Step 1: int subject[ ];

Step 2: subject = new int[5];

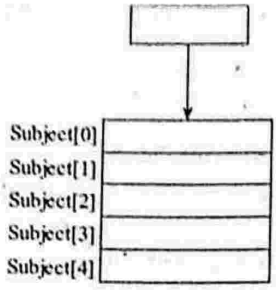
Statement

int subject[ ];

Result



A reference variable is created.



subject = new int[5];

(iii) Initialization of the Memory Location

Initialization means inserting values into the array. Initialization of arrays can be done through loops or by direct assignment.

Example

```
for(i = 0 ; i < 5 ; i + +)
{
    subject[i] = 1;
}
```

subject[0]	1
subject[1]	1
subject[2]	1
subject[3]	1
subject[4]	1

or subject[2] = 101;

subject[0]	1
subject[1]	1
subject[2]	101
subject[3]	1
subject[4]	1

2. Two-dimensional Array

When the data is required to be stored in the form of a matrix, two-dimensional arrays are used.

Syntax

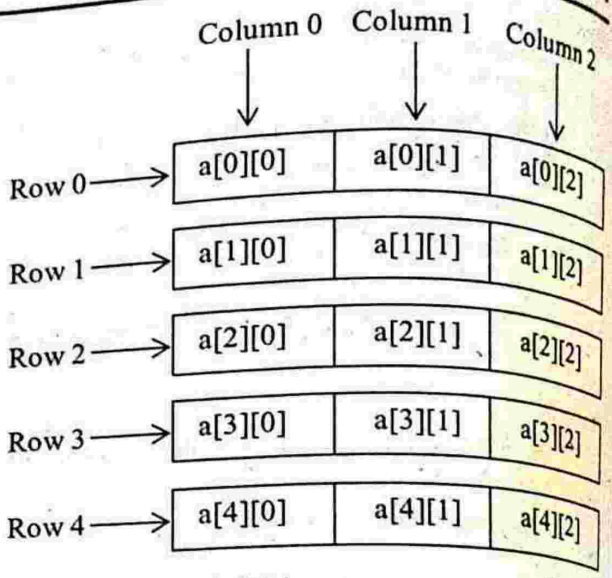
data_type array-name [row size] [column size];

Example

int a[5][3];

Above declaration represents a two-dimensional array consisting of 5 rows and 3 columns. So, the total number of elements which can be stored in this array are 5×3 i.e., 15.

The representation of two-dimensional array of size 5 × 3 in the memory is shown below,



3. Multi-dimensional Array

For answer refer Unit-I, Q42.

Advantages of Using Arrays

1. Array is the simplest kind of data structure.
2. It is relatively easy to create, understand and implement arrays.
3. In arrays direct access to any element is possible. However, modifications done to one element does not effect the other.
4. Arrays have the capability of linking data together, especially with multiple dimension.
5. Every array element can be accessed at constant time.
6. Array variables are capable of storing more number of values.

Q42. Explain about multidimensional arrays with examples.

Answer : (Model Paper-I, Q3(a) | Nov./Dec.-17(R13), Q3(a))

Multidimensional Arrays

These arrays consists of homogeneous data items stored in more than one dimension.

Example

Subject[2][5];

Let us consider subject refers to subjects of engineering.

Subject[2][5] as a whole means 2 semesters 5 subjects.

Subject[0][0] I semester 1st subject
Subject[0][1] I semester 2nd subject
Subject[0][2] I semester 3rd subject
Subject[0][3] I semester 4th subject
Subject[0][4] I semester 5th subject
Subject[1][0] II semester 1st subject
Subject[1][1] II semester 2nd subject
Subject[1][2] II semester 3rd subject
Subject[1][3] II semester 4th subject
Subject[1][4] II semester 5th subject.

Here, one more dimension called semester is added to the earlier single dimensional array. The number of subscripts indicate number of dimensions.

If one more dimension called 'year' is to be added the array would be as shown below.

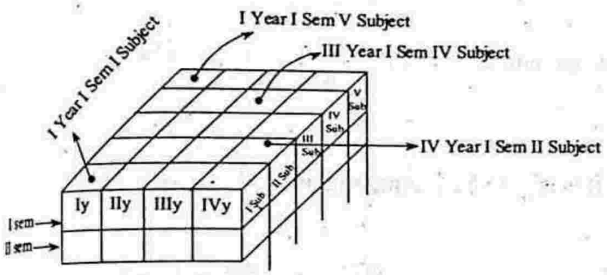
Subject[year][semester][subject];

Example

Subject[4][2][5]

It indicates that there are 4 years and in each year there are 2 semesters and each semester has 5 subjects.

This can be pictorially depicted as follows,



Creation of Multidimentional Arrays

(i) **Declaration of Multidimensional Arrays**

This is similar to single dimensional arrays.

Syntax

datatype arrayname[1][2] . . . [n];

'n' is required number of dimensions.

Example

int Subject[ ][ ][ ];

(ii) **Allocation of Memory**

Syntax

arrayname = new datatype[1][2] . . . . [n];

Example

Subject = new int[4][2][5];

(iii) **Initialization**

```
for(i = 0 ; i < a ; i + +)
{
for(j = 0 ; j < b ; j + +)
{
for(k = 0; k < c ; k + +)
{
arrayname[i][j][k] = value;
}
}
}
```

Example

```
for(i = 0 ; i < 4 ; i + +)
{
for( j = 0 ; j < 2 ; j + +)
{
for(k = 0 ; k < 5 ; k + +)
{
Subject[i][j][k] = 2;
}
}
}
```

1.1.8 Operators

Q43. What are the different types of operators present in Java? Explain.

OR

What are the different kinds of bitwise and boolean logical operators? Nov./Dec.-16(R13), Q3(a)

(Refer Only Topics: Bitwise Operators, Logical Operators)

OR

What are arithmetic operators? Explain.

(Refer Only Topic: Arithmetic Operators)

Answer : Nov./Dec.17(R13), Q2(b)

Operator

An operator can be defined as a symbol which perform specific operations such as mathematical, logical and other manipulations using one or more operands. Java supports large number of operators.

Types of Operators

The operators that are present in java are as follows,

1. Arithmetic operators
2. Relational operators
3. Logical operators
4. Assignment operators
5. Bitwise operators
6. Shift operators
7. Ternary operator (?).

1. Arithmetic Operators

The operators that can perform arithmetic operations are called as arithmetic operators. The various arithmetic operators present in java are as follows;

(a) + (Addition)

The '+' operator is used to perform addition operation between two operands.

(b) – (Subtraction)

The '–' operator performs subtraction operation between two operands.

(c) * (Multiplication)

The '*' operator calculates the product of two operands.

(d) / (Division)

The '/' operator computes division between two operands and yields the quotient as result.

(e) % (Modulus)

The '%' operator performs modulo division between two values and gives the remainder as result.

(f) ++ (Increment)

The '++' is an increment operator that adds the value 1 to its operand (same as a = a + 1). An increment operator can be applied in two ways to the operand as follows,

(i) Pre-increment

If the operand is preceded by increment operator, then it is said to be pre-increment.

Example

```
++a;    //pre-increment
```

If a = 5, then the value of a will be '6'.

(ii) Post-increment

If the operand is succeeded by increment operator, then it is said to be post-increment.

Example

```
a++;    //post-increment
```

If a = 6 then the value of a will be '7'.

Mostly, the prefix and postfix increments are similar. However, they differ when used in larger expressions. If an operand is pre-incremented, then the Java compiler evaluates this operator first, since it has higher precedence compared to post-increment.

(g) –– (Decrement)

The '––' is a decrement operator that subtracts the value 1 from its operand (same as a = a – 1). The decrement operator is often applied in two ways to its operand as follows.

(i) Pre-decrement

If the operand is preceded by decrement operator then it is said to be pre-decrement.

Example

```
––a;    //pre-decrement
```

If a = 7 then the value of 'a' will be '6'.

(ii) Post-decrement

If the operand is succeeded by decrement operator then it is said to be post-decrement.

Example

```
a––;    //post-decrement
```

If a = 9 then the value of 'a' will be '8'.

As similar to increment operator, pre-decrement operator has highest precedence compared to post-decrement. arithmetic operators can be applied to integer types, floating point types and as well as mixed values of both integer & floating point type.

Program

```
public class ArithmeticExample
{
    public static void main(String[ ] args)
    {
        int a, b,c;
        a = 5;
        b = 6;
        c = a + b;
    System.out.println("Addition of a, b :" + c);
    c++;
    System.out.println("The value of c after postincrement is:" + c);
    s = a – b;
    System.out.println("Subtraction of a, b:" + c);
    c – –;
    System.out.println("The value of c after postdecrement:" + c);
    c = a/b;
    System.out.println("Division of a, b:" + c);
    ++ c;
    System.out.println("The value of c after preincrement:"+ c);
    c = a * b;
    System.out.println("product of a, b:" + c);
    – – c;
    System.out.println("The value of c after predecrement:"+ c);
    c = a%b;
    System.out.println("Modulo division of a, b:" + c);
    }
}
```

2. **Relational**

Relational operators can be defined as the operators that can perform comparisons between two operands. The relational operators can be shown as follows,

Operator	Description
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to
!=	Not equal to

Table: Relational Operators

The relational operators yield boolean values result. It can be applied to all numeric types and also to char type.

Program

```
class Relational
{
    public static void main(String[ ] args)
    {
        int x, y;
        boolean a, b;
        x = 5;
        y = 6;
        if(x < y)
            System.out.println("x < y");
        if(x < = y)
            System.out.println("x less than or equal to y");
        if(x! = y)
            System.out.println("x is not equal to y");
        if(x > y)
            System.out.println("x is greater than y");
        if(x > = y)
            System.out.println("x is greater than or equal to y");
        if(x = = y)
            System.out.println("x is equal to y");
    }
}
```

3. Logical Operators

Logical operators can be defined as the operators that can perform logical operations between two operands. The logical operators can be shown as follows,

Operator	Description
&	AND
|	OR
^	XOR (Exclusive OR)
||	Logical OR (OR) Short-circuit OR
&&	Logical AND (or) short-circuit AND
!	Logical NOT

Table: Logical Operators

The logical operators yield boolean values as result. It compare only boolean type values. As a matter of fact, these operators perform operations based on the following truth table.

A	B	A & B	A | B	A ^ B	!A
T	T	T	T	F	F
T	F	F	T	T	F
F	T	F	T	T	T
F	F	F	F	F	T

The other logical operators which can produce efficient code can be called as short-circuit logical operators. Because, short-circuit logical operators i.e., Logical AND(&&) and Logical OR(||) evaluates the first operand and as well as second operand if necessary. If Logical AND (&&) operator is evaluating, it checks the first operand and it is true, then it checks second operand otherwise it terminates. The logical OR (||) evaluates the first operand and if it is true it does not evaluate second operand and if it is false then evaluates second operand. However, logical operators such as &, |, ^, ! evaluates both operands every time. Therefore, short-circuit logical operators are efficient compared, to logical operators.

Program

```
class ShortCircuitExample
{
    public static void main(String[ ] args)
    {
        int a, b, c, d;
        a = 10;
        b = 20;
        c = 30;
        if(c > a && c > b)
        System.out.println("c is greater ");
        if(a < b || a < c)
        System.out.println("a is smaller than b or c");
    }
}
```

4. Assignment Operators

Assignment operator can be defined as an operator which assigns a value to the variable or operand. The general form of assignment operator is as follows,

Syntax

```
variable_name = expression;
```

Here, variable_name is name of the variable and the expression is required expression that has to be assigned to the variable. This expression is valid only if the both type of variable and expression are same. The other feature of assignment operator in java is "chain of assignment" which assigns a single right hand value to all the variables in the chain.

Program

```
public class AssignmentExample
{
    public static void main(String[ ] args)
    {
        int a, b, c, d;
        a = b = c = 10;    //chain of assignment
        d = 10;
        System.out.println("The values of a, b, c and d are:" +a + " " + b + " " + c + " " + d);
    }
}
```

In assignment operator, Java supports another feature called as short hand assignment operator. This feature makes the coding of assignment operator easy. The general form of short hand assignment operator is as follows,

Syntax

variable_name operator = expression;

Example

x + = 2; //which is similar to x = x + 2

x – = 5; //which is similar to x = x – 5

The arithmetic and logical shorthand assignments are as follows,

Arithmetic	Logical
+ =	& =
– =	& =
* =	| =
/ =	| =
% =	^ =

5. Bitwise Operators

Bitwise operators can manipulate the set of bits associated with the operands. These operators can be applied to integer type and character type values and cannot be applied to floating-point and boolean type values since bitwise operators can test, set or shift the individual bits to build another value. These operators are essential for different programming tasks.

Java supports various bitwise operators. These operators are as follows,

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
~	One's complement
<<	Shift left
>>	Shift right
>>>	Shift right with zero fill (or) unsigned shift right

Table: Bitwise Operators

The result of bitwise operators can be shown in the following table,

A	B	A & B	A | B	A ^ B	~A
1	1	1	1	0	0
1	0	0	1	1	0
0	1	0	1	1	1
0	0	0	0	0	1

6. Shift Operators

Shift operators can be defined as the operator that can shift the bit position value to the left or to the right depending on the required position. The shift operators present in Java are as follows,

Operator	Description
<<	Left shift
>>	Right shift
>>>	Unsigned right shift

The general form of shift operators is follows,

Value << number of bits

Value >> number of bits

Value >>> number of bits

The value is the one which have to be shifted depending on specified bit positions. The left shift operator is used to shift the value on left depending on specified bit positions and brings the sign bit value 0 on to right position. The right shift is used to shift the value on right and assigns 0 on left.

The >>> (Unsigned right shift) operator always assigns 0 on left. Thus, it is called as zero-fill right shift.

7. Ternary Operator (?)

The general form of ternary operator is as follows,

condition? expr1 : expr 2;

Here, condition is the required condition and expr1, expr2 are required expressions of any type excluding void data type.

Example

a > b ? "a is big" : "b is big"

If the condition a > b is true, then the first expression "a is big" will be the result of entire expression. Whereas if condition is false, second expression "b is big" is the result. The ternary operator is also called as conditional operator.

Program

```
class TernaryExample
{
    public static void main(String[ ] args)
    {
        int x, a = – 3;
        while(a < 5)
        {
            x = a! = 0? 100/a: 0;  //it prevents the
                                   //expression 100/
            if(a! = 0)
            System.out.println("The output is 100/
        }
    }
}
```

Q44. Discuss the order of precedence of various operators.

Answer : Model Paper-II, Q2(a)

Operator Precedence

In Java, every operator is assigned a precedence which determine when it must be evaluated once it is used in an expression. The operator with higher precedence will be evaluated first and the operator with the lower precedence will be evaluated next.

The below table depicts the precedence of the operators in the order from higher to lower.

Operator	Description
+ +	Post increment
– –	Post decrement
+ +	Pre increment
– –	Pre decrement
~	Tilde
!	Logical Not
+	Unary increment
–	Unary decrement
*	Multiplication
/	Division
%	Modulo division
+	Addition
–	Subtraction
>>	Right shift
>>>	Unsigned right shift
<<	Left shift
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
==	equal to
!=	Not equal to
&	Bitwise AND
^	Bitwise exclusive OR
|	Bitwise OR
&&	Logical AND
||	Logical OR
?:	Conditional
=	Equal

The symbols( ), [ ] and. have highest precedence of all the operators and they are evaluated first.

Example

$8 + 4 * (8 - 2)$

$= 8 + 4 * 6$

$= 8 + 24$

$= 32$

Q45. Write a short note on making use of parenthesis.

Answer :

The question of precedence of the operations is raised by the parenthesis holding those operators. This is very much important is obtaining the desired result. Consider the below expression.

$$x >> y + 5$$

The above expression will first add 5 and then shifts 'x' by the obtained result. So, it can be written as follows,

$$x >> (y + 5)$$

If the user needs to first shift x right by y and then add 5 to the result then the above expression can be parenthesized as follows,

$$(x >> y) + 5$$

Rather than changing the precedence of an operator, parenthesis can even be used to clarify the expression meaning. Certain complex expressions might be difficult to understand. Such confusions can be cleared using parenthesis. Therefore, parenthesis added to reduce the ambiguity will not at all affect the program in anyway.

1.1.9 Expressions

Q46. Define an expression. Discuss how an expression is evaluated with an example.

Answer :

Expression

An expression can be defined as a combination of operators, variables and constants. These are referred as constituents of an expression.

Example

```
c = a + b;
a = 5;
b = a * c;
```

Expression Evaluation

Expressions in Java are evaluated through the use of assignment statement, which is of the form as,

variable = expression;

In the above statement, the variable indicates the name of the variable. The expression depicts a complete expression containing operands and operators. The expression will be evaluated initially and then the result of it will be assigned to the variable which is on the left hand side. Even if the variable holds any other value it will be replaced with the evaluated result. Consider the below statements,

```
a = x + y – z;
b = (y * z) + x/p;
```

In the above statements, the values are first assigned to the respective variables. The expression is then evaluated and the result is assigned to the variable on the left hand side.

1.1.10 Control Statements

Q47. What are conditional statements? Discuss various types with examples.

Answer :

Conditional Statements

Conditional statements can be defined as the statements that are executed depending on the specified condition. If the condition is true, block of statements are executed. Otherwise control flow comes out of the block.

Types of Conditional Statements

The various conditional statements available in java are as follows,

(a) if statement
(b) if-else statement
(c) nested if-else statements
(d) if-else-if ladder
(e) switch statement
(f) nested switch statements.

(a) if Statement

'if' statement (or simple 'if' statement) is used for decision making. It allows the computer to first evaluate the condition and depending on its resultant value it transfers the control to a particular statement in the program.

This statement performs an action if the condition is true, otherwise it skips that action and executes other statement.

Syntax

```
if (condition)
{
statement-block;
}
statement-next;
```

Here, if the condition is satisfied, then statement-block is executed followed by statement-next. Otherwise, statement-block is skipped and only statement-next is executed.

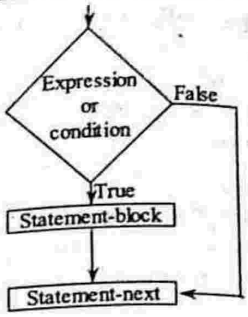


Figure: Flowchart of Simple if Statement

(b) if-else Statement

This statement is an extension of simple 'if' statement.

Syntax

```
if (condition)
{
statement-1;
}
else
{
statement-2;
}
statement-next;
```

In the above syntax, statement-1 will be executed followed by statement-next when the condition is true. Otherwise statement-2 will be executed followed by statement-next.

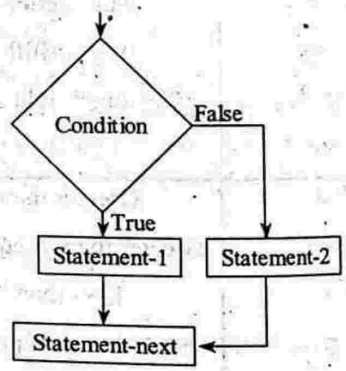


Figure: Flowchart of if-else Statement

Program

```
class IfExample
{
    public static void main(String[ ] args)
    {
        int a, b;
        a = 5;
        b = 6;
```

```
        if (a > b)
        {
            System.out.println("a is greater than b");
        }
        else
        {
            System.out.println("b is greater than a");
        }
    }
}
```

(c) **Nested if-else Statements**

A nested if-else statement is an if statement which is defined in another if statement. The nested if's are essential in programming since they yields a follow-up selection depending on the result of previous selection.

Syntax

```
if(condition-1)
{
    if(condition-2)
    {
        statement-1;
    }
    else
    {
        statement-2;
    }
}
else
{
    statement-3;
}
statement-next;
```

If condition-1 is true, then condition-2 is executed otherwise, statement-3 will be executed followed by statement-next. If condition-2 is evaluated then statement-1 is executed (if condition is true) or statement-2 is executed (if condition is false) followed by statement-next in both cases.

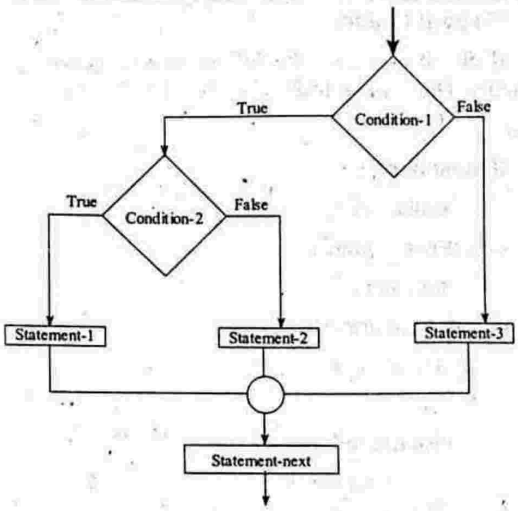


Figure: Flowchart for Nesting of if-else Statement

Program

```
public class NestedExample
{
    public static void main(String[ ] args)
    {
        int a, b, c;
        a = 7;
        b = 8;
        c = 9;
        if(a > b)
        {
            if(a > c)
            {
                System.out.println("a is big");
            }
            else
            {
                System.out.println("c is big");
            }
        }
        else
        {
            if(b > c)
            {
                System.out.println("b is big");
            }
            else
            {
                System.out.println("c is big");
            }
        }
    }
}
```

(d) if-else-if Ladder

if-else-if ladder can be defined as a sequence of if-else statements. The general form of if-else-if ladder is as follows,

Syntax

```
if(condition-1)
    statement-1;
else if(condition-2)
    statement-2;
else if(condition-3)
    statement-3;
    .
    .
    .
    else if(condition-k)
        statement-k;
        else
            statement-k+1;
statement-next;
```

The conditions are evaluated sequentially (i.e., from top to bottom). If a particular condition is false then the next following conditions are evaluated. If a condition is satisfied, then its corresponding statement is executed.

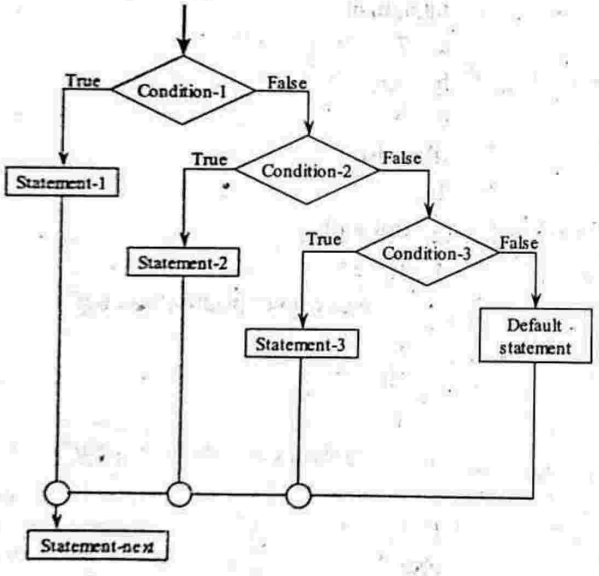


Figure: Flowchart of else-if Ladder

Program

```
class LadderExample
{
    public static void main(String[ ] args)
    {
        int x = 2;
        if (x < 0)
        System.out.println("The value of x is negative");
        else if(x = = 0)
        System.out.println("The value of x is zero");
        else if(x = = 1)
        System.out.println("The value of x is one");
        else if(x = =2)
        System.out.println("The value of x is two");
        else
        System.out.println("The value of x is
                                    greater than two");
    }
}
```

(e) switch Statement

switch statement can be called as multi-way branching statement. The statement provides multiple alternatives and the user can select the required option.

Syntax

```
switch(expression)
{
    case constant1: statement-1;
                    break;
    case constant2: statement-2;
                    break;
                    .
                    .
                    .
    case constantn: statement-n;
                    break;
    default: default-statement;
             break;
}
statement next;
```

Here, the expression is a valid 'C' expression and constant is the result of the expression. When the constant value is a character, it has to be enclosed within single quotes.

The value of the expression is evaluated and it is compared with constant case values. When match is found, the corresponding statement block associated with the case is executed.

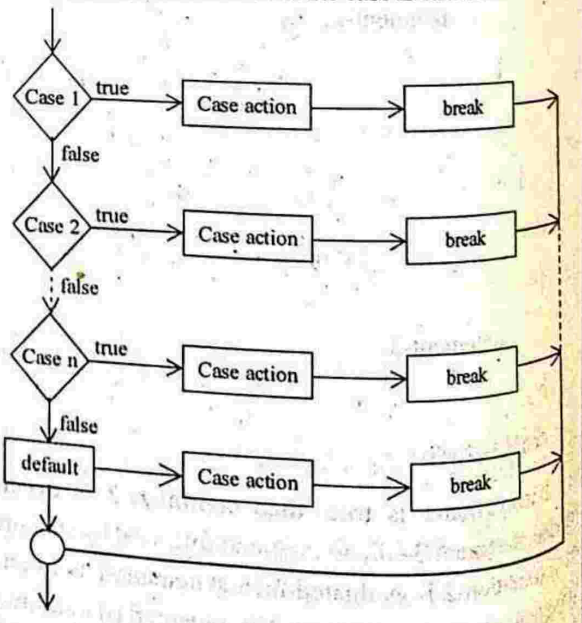


Figure: Flowchart of Switch Statement

Program

```
class SwitchExample
{
    public static void main(String[ ] args)
    {
        int x;
        for(x = 0; x < 7; x++)
        switch(x)
        {
        case 0:
            System.out.println("Sunday");
            break;
        case 1:
            System.out.println("Monday");
            break;
        case 2:
            System.out.println("Tuesday");
            break;
        case 3:
            System.out.println("Wednesday");
            break;
        case 4:
            System.out.println("Thursday");
            break;
        case 5:
            System.out.println("Friday");
            break;
        case 6:
            System.out.println("Saturday");
            break;
            default: System.out.println("No week days");
        }
    }
}
```

Generally, the break statement is not mandatory. It terminates the switch block after execution of particular case. Thereby, it prevents the execution of successive cases including default case.

(f) **Nested switch Statements**

Nested switch statements can be defined as the switch statement with in another switch block.

Syntax

```
switch(expression1)
{
    //body
    switch(expression 2)
    {
        //body
    }
}
```

Example

```
switch(opt)
{
    case 1:  System.out.println("outer Switch block");
    break;
    switch(opt 1)
    {
        case 1: System.out.println("Inner Switch block");
        break;
        //body
    }
    break;
    case 2:
    ==
}
```

Q48. Explain the different types of loops with examples.

Answer :

Loop

Loop is a process of executing action or series of actions defined in the blocks of code infinitely. It is necessary to terminate the loop as soon as the required task is completed. Hence, a condition is used to control the loop before or after the execution of every block of code. The looping mechanism must include the following steps,

Step 1: Declaration and initialization of counter.

Step 2: The executable statements in the loop.

Step 3: The condition to execute the loop.

Step 4: Incrementation/decrementation of the loop.

Types of Loops

The different loops that present in Java are as follows,

(a) while loop

(b) do-while loop

(c) for loop

(d) for-each loop.

(a) while Loop

The 'while' statement will be executed repeatedly as long as the expression remains true.

Syntax

```
while (expression)
{
body of the loop;
}
```

When the 'while' is reached, the computer evaluates expression. If it is found to be false, body of loop will not be executed and the loop terminates. Otherwise, the body of loop will be executed, till the expression becomes false.

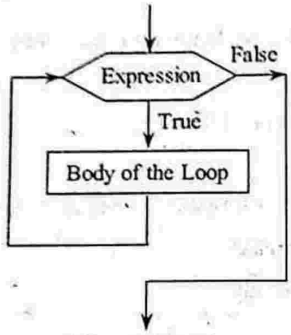


Figure: Flowchart of while Statement

Program

```
import java.util.Scanner;
public class WhileExample
{
    public static void main(String[ ] args)
    {
        Scanner sc = new Scanner(System.in);
                //Scanner class for reading input
        System.out.println("Enter the input number");
        int ip = sc.nextInt( );
        int res = reverseMethod(ip);
        System.out.println("The reversed number is"
                                        + res);
    }
    public static int reverseMethod(int val)
    {
        int res = 0;
        int remainder;
        while(val > 0)
        {
            remainder = val%10;
            val = val/10;
            res = res *10 + remainder;
        } while(x > 0);
        return res;
    }
}
```

(b) do-while Loop

It is similar to that of 'while' loop except that it is executed at least once. The test of expression for repeating is done after each time body of loop is executed.

Syntax

```
do
{
body of loop;
} while (expression);
```

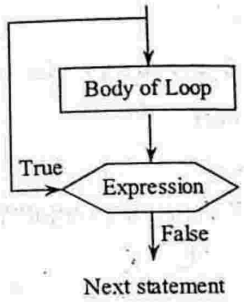


Figure: Flowchart of do-while Statement

Here, Java compiler executes the body of the do-while loop first without evaluating any condition. After this execution, the condition can be checked and if it is true the loop is executed, otherwise terminated. The do-while statement can be called as exit-controlled loop since it checks the condition at the end of loop.

Program

```
import java.util.Scanner;
class DoWhileExample
{
    public static void main(String[ ] args)
    {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number");
        int ip = sc.nextInt( );
        int res = reverseMethod(ip);
        System.out.println("The result after reversing
        the number is" + res);
    }
```

```
    public static int reverseMethod(int x)
    {
        int res = 0, remainder;
        do
        {
            remainder = x%10;
            x = x/10;
            res = res * 10 + remainder;
        } while(x > 0);
        return res;
    }
}
```

for Loop

'for' statement is another type of looping statement.

Syntax

```
for (initializing expression; testing expression; updating expression)
{
statements;
}
```

Here the initialization expression specifies the initial value. The testing expression is a condition that is tested at each pass. The program is executed till this condition remains true. Updating expression is an unary expression that is either incremented or decremented to change the initial value. The conditional expression is evaluated and tested at the beginning while unary expression is evaluated at the end of each pass.

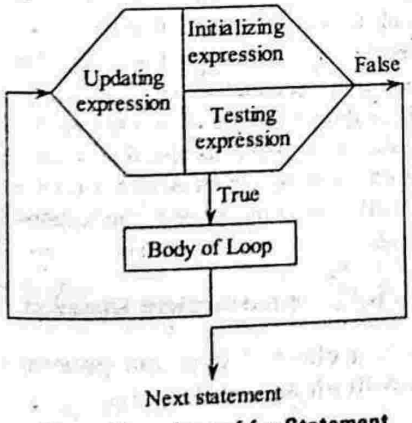


Figure: Flowchart of for Statement

Program

```
public class ForExample
{
    public static void main(String[ ] args)
    {
        int a;
        for(a = 5; a < 10; a++)
        {
            System.out.println("The a value is" +a);
        }
    }
}
```

(d) for-each Loop

The for-each loop can be called as an enhanced for loop which traverses array elements or collection of elements. The for-each loop is essentially to reduce programming errors and can be used to build the code in high readable format.

Syntax

```
for(data-type variable: array-name collection of elements)
{
    =
}
```

Here, data-type is the variable type. A variable can be created to access the elements of an array and array-name is the name of the array.

The array elements are assigned to the variable one after the other and the loop body is executed until the elements in an array are completed. It does not require any condition and increment or decrement statement to execute body of the loop.

Program

```
public class ForEachExample
{
    public static void main(String[ ] args)
    {
        int a[ ] = {1, 2, 3, 4, 5};
        for(int x : a)
        {
            System.out.println("The element of array is");
            System.out.println(x);
        }
    }
}
```

Q49. What are the different types of jump statements? Explain each of them with an example.

Answer :

Jump statements can be used to skip over the loop by terminating a set of statements. The various types of jump statements in java are as follows,

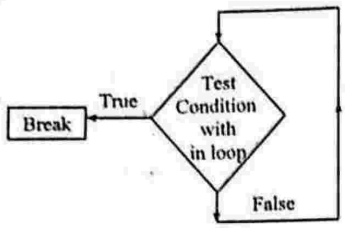
(a) Break

(b) Continue.

(a) Break

The 'break' statement performs unconditional jump that terminates or exists the iteration or switch statement. It terminates the loop as soon as the control encounters it. And then the control is transferred to the statement that occurs after the iteration or switch statement. Therefore, it closes the smallest enclosing do, for, switch, while statements. It is written as,

break;



Figure

Program

```
class BreakExample
{
    public static void main(String[ ] args)
    {
    int x, a, b;
    x = 10;
    for(a = 0; a < x; a ++)
    {
    b = a*a;
    if(b > = x)
    break;
    System.out.print("b=" + b+ " ");
    }
    System.out.println("Completion of loop");
    }
}
```

Here, the for loop is terminated when the square value of 'a' exceeds 10. After the termination, the control goes to next statement loop which prints "Completion of loop".

When the break statement is used in inner loop of nested loops, then the control comes out of the inner loop and goes to the next statement after inner loop. The break statement does not terminates the outer loop. While using break statement, there is a need to remember two points.

(i) In a loop multiple break statements can be used

(ii) If break is used in switch, then the switch block is terminated but not the loop in which switch block is defined.

(b) Continue Statement

Continue statement is a unconditional jump statement. It tells the interpreter to continue the next iteration of the loop. It is a keyword used for continuing the next iteration of the loop. In contrast to the break statement, the continue statement does not exit from the loop but transfers the control to the testing expression (while, do-while) and to the updating expression (for). While and do-while loops can logically act as a jump statement that transfer the control to the end of the loop body. This is because both these statements transfer their control to two different positions.

Program

```
public class ContinueExample
{
    public static void main(String[ ] args)
    {
        int x = 0;
        while(x < 15)
        {
            if((x%2)! = 0)
            continue;
            System.out.println(x);
            x++;
        }
    }
}
```

Here, the even number is printed whenever x%2 = 1 and gets iterated using continue statement when x%2! = 1. This loop iterates until the *x* value reaches 15. When continue statement is used in inner loop of nested loops then the control goes to the condition of outer loop but not inner loop. If continue statement is used in while and do-while loops, the control goes to the loop condition immediately. However, if continue is used in for loop, it evaluates the iteration expression of loop first, next loop condition is executed and finally the loop body is executed. The continue statement is mainly used in a rich set of loop statements that used in frequent applications. Due to this reason, java supports continue statement.

1.1.11 Introducing Classes

Q50. What is a class? Write the general form of a class with an example.

Answer : Model Paper-I, Q2(a)

Class

A class can be defined as a template that groups data and its associated functions. The class contains two parts namely

(a) Declaration of data variables

(b) Declaration of member functions.

The data members of a class explains about the state of class and the member function explains about the behaviour the class. There are three types of variables available for a s. They are,

(i) Local variables
(ii) Instance variables
(iii) Class variables.

Local Variables

Local variables are the variables that are declared inside the methods.

Instance Variables

Instance variables are the variables that are declared inside the class but outside of the methods.

Class Variables

Class variables are the variables that are declared inside the class with static modifier and they reside outside of the method.

eneral Form of a Class

A class can be declared with the help of 'class' keyword llowed by the name of the class. The syntax of a class is as llows,

```
class name_of_the_class
{
    //declaration of Instance variables
type data_variable1;
type data_variable2;
    //declaration of functions
type function_name1(arg_list)
{
    body of function
}
type function_name2(arg_list)
{
    body of function
}
}
```

It is better to maintain information of one logical entity in a class.

Definition of a Class

```
class Student
{
    int stdId;
    int stdName;
    void getStdId( );
    int setStdId(int sid);
}
```

The above definition creates a class Student. In this class, tdId and stdName are the instance variables and getStdId( ), etStdId( ) are the member functions of the class.

In Java, main( ) method often defined in the class itself.

Q51. Give a brief description about object. Also explain how an object can be declared and initialized.

Answer :

Object

An object can be defined as an instance of a class which is used to access the members of a class.

Declaration of An Object

An object is similar to that of variable declaration.

Syntax

name_of_the_class object_name;

Example

Student objStudent;

Here, Student is the name of the class and objStudent is the reference variable which can represent the object. A reference variable objStudent is declared for an object just to hold the address of the object.

Student objStudent ⇒ null

objStudent

Initialization of Object

When a reference variable is declared, it is necessary to assign physical copy of the object to that variable. This can be done by using 'new' operator. This operator allocates memory dynamically to store instance variables using a single argument called as constructor call. After the memory allocation, new operator returns the address of memory location to the class. This address can be stored in the reference variable which is created.

Syntax

class_name object_name = new class_name( );

Example

Student objStudnet = new Student( );

Here, an object named as objStudent is created and memory is allocated. Meanwhile, instance variables are assigned with various values. However, it is required to start with initial values. If initial values are not specified, default values are assigned to these instance variables depending on its data type. The initial values are assigned using the following mechanisms.

(a) Instance variable initializer
(b) Constructor.

(a) Instance Variable Initializer

The instance variable initializer assigns values directly to any instance variable which is declared outside the function/method but inside the class.

(b) Constructor

Constructor is often used to initialize the objects. When an object is created, constructor is automatically invoked. If constructor is not created manually, a default constructor with no arguments and body will be created by java compiler. This default constructor is invoked after the execution of below statement.

new Student( );

The name of both constructor and class is always same.

Accessing Class Members

An object can access the members of a class using dot(.) operator.

Syntax

objectname.membername;

Example

objStudent.stdId = 101;

objStudent.getStdId( );

Here, objStudent is the name of the object, '.'is the operator used to access member functions and stdId is an instance variable. Finally getStdId( ) is the function which is to be invoked.

Q52. Discuss in brief about assigning object reference variables.

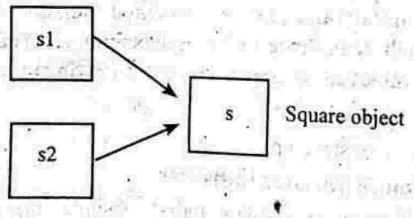
Answer :

The behavior of the object reference variables is different than the assignment of it. For example consider two objects defined below,

Square s1 = new Square( );

Square s2 = s1;

In the above code fragment, it is not that s2 is assigned a copy of the object that s1 refers to. The s1 and s2 does not refer to separate objects. Rather they both refer to same object. The assignment statement will not allocate any memory or copy of the original object. The s2 will refer to the object to which s1 refers to. The changes done to s2 will also affect the object to which even s1 is referring.



In the above figure s1 and s2 refer to same object. Therefore if any other assignment is done to s1 then it will unhook from square object without affecting s2.

Square s1 = new Square( );

Square s2 = s1;

s1 = null;

Here s1 will be set to null but s2 will remain pointing to original object.

Q53. Explain in detail about 'this' keyword and garbage collection.

OR

Demonstrate the use of 'this' keyword.

(Refer Only Topics: 'this' Keyword, Example)

May-19(R16), Q3(b)

Answer :

'this' Keyword

The keyword 'this' is used to create a reference to an object. It will be passed implicitly when method is called. This will be done automatically. Any member function can find the address of the object and access its data (to which it belongs) by using this keyword.

Example

```
class Exp
{
    double x;
    int ex;
    double y;
    Exp(double base, int e)
    {
        this.x = base;
        this.ex = e;
        this.y = 1;
        if(e == 0)
        return;
        for(e = 10; e > 0; e --)
        this.value = this.value * base;
    }
    double get( )
    {
        return this.value;
    }
}
class Demo
{
    public static void main(String[ ] args)
    {
        Exp a = new Exp(2.3, 4);
        System.out.println(a.x + "is raised to" +
                                    a.ex + "power
                                    =" + a.get( ));
    }
}
```

Garbage Collection

Dynamic allocation of objects is done using new keyword in java. Some times allocation of new objects fail due to insufficient memory. In such cases memory space assumed by unused objects is de-allocated and made available for reallocation. This is done manually in languages such as C and C++ using delete keyword. In java a new concept called garbage collection is introduced for this purpose. It is a trouble-free approach which reclaims the objects automatically. This is done without the interference of the programmer. The objects which are not used for long time and which does not have any references are identified and their space is de-allocated. This recycled space can now be used by other objects.

Garbage collection is only performed during program execution. It is a time consuming process therefore it is performed only at relevant instance.

Q54. What is the use of finalize( ) method? What is its disadvantage?

Answer :

finalize( ) Method

An object makes use of many non-Java resources such as file handle or window character font, etc. These resources should be freed before an object is destroyed. Finalization is a mechanism that frees all the resources used by an object. With the help of finalization, one can define specific action that should be taken when an object is about to be destroyed by the garbage collector.

There is a finalize( ) method that performs the finalization. This method is called by the Java run time whenever it is about to destroy an object of that class. Inside the finalize( ) method one can include all those functionalists that must be performed before an object is destroyed. The garbage collector checks dynamically for each object that is not used by any class directly or indirectly and just before freeing an asset the Java run time calls the finalize( ) method on the object.

General Form

```
protected void finalize( )
{
    // body of finalization
}
```

The protected keyword prevents the access to finalize( ) by any code defined outside its class. This method is only called just before an attempt of garbage collection.

Disadvantage

The finalize( ) method is called only prior to the garbage collection. For example, it is not called if an object goes out-of-scope. That is, it will be not be known, even if it is executed.

Q55. Discuss about the stack class.

Answer :

Stack Class

In Java, encapsulation is achieved through classes. It is considered as a new data type to define the nature of the data and routines of it. The methods tend to define a consistent as well as controlled interface for the class data. The class can be used through its methods irrespective of their implementation. In other word's class is assumed as "data engine". The internal features of it are not exposed so the internal working can be changed as required. A practical and archetypal example of this would be the stack. It contains the data in the form of first-in-last-out order. It allows two operations to be performed on it such as push and pop. The push operation is used to insert an item at the top of the stack. The pop operation is used to remove an item from the top of the stack. A stack can be easily encapsulated.

Example

```
class Stack
{
    public static void main(String args[ ])
    Stack stk1 = new Stack( );
    Stack stk2 = new Stack( );
    for(int i =0; i<5; i++)
        stk1.push(i);
    for(int i = 5; i < 10; i++)
        stk.push(i);
    System.out.println("stk1 contents");
    for (i = 0; i <5; i++)
    System.out.println(stk1.pop( ));
    System.out.println("stk2 contents");
    for(i = 0; i <10; i++)
        System.out.println(stk2.pop( ));
}
}
```

In the above program two stacks will contain the items separately.

1.1.12 Methods and Classes

Q56. What is a method? How a method is used in the class? Explain.

Answer :

Method

A method is used to define what an object can accept in java. It will be created in the class and therefore, it remains as the part of the associated class. A method contains a set of statements which define certain actions. Each method performs a particular task.

Syntax

```
return_type method_name(arg1, arg2, ..., arg n)
{
    method_body
}
```

In the above syntax, the return_type indicates the type of value that the method returns. The method_name indicates the name given to the method. It should be a valid identifier. Arguments are enclosed in the parenthesis. They can also be declared when passed to the method. The body of the method contains the operations that are to be performed on the data.

A method can be called by an object as shown below,

object_name.method_name(arg1, arg2, ..., argn)

A method terminates or the control returns when a closing brace or return statement is encountered. The result obtained after the task performed by the method would be the return value of that method.

Example

```
class Square
{
    int side;
    void getData(int x)
    {
        side = x;
    }
    int Sarea( )
    {
        int area = 4 × side;
        return(area);
    }
}
class Square
{
    public static void main(String args[ ])
    {
        int a, b;
        Square s = new Square( );
        Square t = new Square( );
        s.side = 12;
        a = 4 * s.side;
        t.getData(5);
        b = t.area( );
        System.out.println("Area of square 1 =" + a);
        System.out.println("Area of square 2 =" + b);
    }
}
```

Q57. How a method can be overloaded? Explain the process of overloading constructors.

OR

What is meant by overloading methods? Explain in detail.

(Refer Only Topic: Method Overloading)

Answer : Nov./Dec.-17(R13), Q3(b)

Method Overloading

Several methods in java are allowed to have same method name with different parameters and different definitions. This is called method overloading. This concept is used when there is necessity to perform similar tasks with different arguments. Compiler does not get confused when calling these methods. It matches the method name, number of arguments and their type. based on this it decides which method to be called. This process is called polymorphism.

Every method should be provided unique parameters. This concept allows the programmer to use several methods with same name and sometimes with same definition.

Example

```
class Square
{
    int side;
    method 1(int a)
    {
        side = a;
    }
    method 2(int b)
    {
        side = b;
    }
    int area( )
    {
        int res = 4*s;
        return res;
    }
}
```

Constructor Overloading

Constructors can also be overloaded in the same way as methods are done. They are declared and defined as methods. But constructors does not have return type. Constructors have same name of the class but each of it provides a different definitions and parameters of different types.

All the constructors and the class can only be differentiated with their parameters. This is because all of them have the same name.

```
ple
class Square
{
    int side;
    Square( )
    {
        side = 15;
    }
    Square(int a)
    {
        side = a;
    }
    int area( )
    {
        int res = 4*side;
        return side;
    }
}
```

Discuss in brief about using objects as parameters.

swer :

It is possible even to pass objects as parameters to ctions rather than simple types. This is illustrated in the ow example.

```
mple
class Object
{
    int x, y;
Object(int p, int q)
{
        x = p;
        y = q;
}
boolean equation(Object obj)
{
    if(obj.x = = x && obj.y = = y)
        return true;
    else
        return false;
}
}
```

```
class ObjectPass
{
public static void main(String args[ ])
{
    Object ob1 = new Object(100, 22),
    Object obj2 = new Object(100, 22);
    Object obj3 = new Object(– 1, – 1);
    System.out.println("ob1 = = 0b2:"+ ob1.
                                    equalTo(ob2));
    System.out.println("ob1 = = ob3:" + ob1.
                                    equalTo(ob3));
}
}
```

In the above program, the equalTo( ) method will compare two objects for equality. It compares the invoking object with the object that is passed as argument. The type of this object will be its class i.e., object.

A common usage of this concept would be constructors. The objects might be required to be created frequently. This can be done by defining the constructors that receives object of its class as parameter.

Q59. Explain how objects are passed to a function.

Answer :

An object in Java can be passed as an argument to a function in the same way as passing any variable to a function. There are two ways to pass an object to a function, they are as follows,

1. Pass by value
2. Pass by reference.

1. Pass by Value

In this method, the copy of the object is passed as an argument to the function. The changes made to the object inside the function will not affect the actual object that is used to call the function. The scope of the object will be limited only upto the function to which it is passed. It is used in the function just as any new object.

2. Pass by Reference

In this method, the reference of the object is passed as an argument to the function not the object or a copy of it. The reference of the object is passed by passing the address i.e., the location of the object. The function can directly work on the object. So the changes made to the object in the function will reflect back to the actual object. This method is mostly used and is an efficient method. This is because the effort required here is to pass only the address of the object but not the entire object.

An object is allowed to be passed even to non-member functions of a class also. These members can then access the public member functions of the class through the function objects passed to it. They cannot access the private members functions of the class.

Example

```
import java.io.*;
import java.util.*;
class Square
{
    int side;
    Square(int s)
    {
        side = s;
    }
    void area(Square sq)
    {
        int area = 4*sq.s;
        System.out.println("Area of square:" + area);
    }
}
class Demo
{
    public static void main(String[ ] args)
    {
        Square sq = new Square(5);
        sq.area(sq);
    }
}
```

Q60. Write short notes on returning objects.

Answer :

A method can even return the data of class type created by the user in addition to other types of data. Consider the below example,

Example

```
class ObjReturn
{
    int x;
    ObjReturn(int a)
    {
        x = a;
    }
    ObjReturn incr( )
    {
        ObjReturn ob = new ObjReturn(x + 5);
        return ob;
    }
}
class ObjRet
{
    public static void main(String args[ ])
    {
    ObjReturn ob1 = new ObjReturn(3);
    ObjReturn ob2;
    ob2 = ob1.incr( );
    System.out.println("ob1.x:" + ob1.x);
    System.out.println("on2.x:"+ ob2.x);
    ob2 = ob2.incr( );
    System.out.println("ob2.x after next increment:"
    + ob2.x);
    }
}
```

In the above program, a new object gets created for each time the incr( ) is invoked. This function will return an object containing the value of *x* greater than the invoking object.

Q61. What is recursion? Explain with an example.

Answer :

Recursion

Recursion is the process or technique by which a function calls itself. A recursive function contains a statement within its body, which calls the same function. Thus, it is also called as circular definition. A recursion can be classified into direct recursion and indirect recursion. In direct recursion, the function calls itself and in the indirect recursion, a function (f1) calls another function (f2), and the called function (f2) calls the calling function (f1).

When a recursive call is made, the parameters and return address gets saved on the stack. The stack gets wrapped when the control is returned back.

Advantages

1. Recursion solves the problem in the most general way as possible.
2. Recursive function is small, simple and more reliable than other coded versions of the program.
3. Recursion is used to solve complicated problems that have repetitive structure.
4. Certain problems can be easily be understood using recursion.

Disadvantages

1. Usage of recursion incurs overhead, since this technique is implemented using function calls.
2. Whenever a recursive call is made, some of the system's memory is consumed.

```
ample
class Fact
{
    int factorial(int x)
    {
        int res;
        if(x = = 1)
        return 1;
        res = factorial(x – 1)*x;
        return res;
    }
}
class Rec
{
    public static void main(String arg[ ])
    {
        fact f = new fact( );
        System.out.println("Factorial of 6 = "+
                                    f.factorial(6));
    }
}
```

Q62. Explain in detail about the concept of access control in Java.

OR

Explain the significance of public, protected and private access specifiers in inheritance.

nswer : **Nov./Dec.-17(R16), Q2(b)**

The methods and variables belonging to a class are metimes limited only to that class. Other classes are not owed to access them. This is called as encapsulation. Such riables can only be accessed by calling the methods of that ss. Java provides four types of access specifiers namely blic, private, protected and a default specifier.

Public Access

The variables and methods when declared as public can accessed by all the other classes. In the concept of inheritance the subclasses can access the public members of its base ss.

ntax

```
public datatype variable_name;
public datatype method_name(args) { }
```

Private Access

The variables and methods when declared as private e accessed only by the class in which they are defined. In the ncept of inheritance, the subclasses are restricted from using e private members of its base class.

ntax

```
private datatype variable_name;
private datatype method_name(args) { }
```

3. Protected Access

The variables and methods when declared as protected are accessed by all the classes belonging to the same package. In the concept of inheritance all the subclasses can be the protected members of its base class.

Syntax

```
protected datatype variable_name;
```

Example

```
class Access
{
    int p;
    public int q;
    private int r;
    void setr(int x)
    {
        r = x;
    }
    int getr( )
    {
        return r;
    }
}
class Demo
{
    public static void main(String args[ ])
    {
        Access object = new Access( );
        object.p = 20;
        object.q = 30;
        object.setr(200);  //OK
        System.out.println("The values of p, q and r:"
        + object.p + " " + object.q + " " object.getr( ));
    }
}
```

Q63. What is meant by static field and static method? Explain.

Answer :

Static Field

A field is said to be static when its declaration is preceded with static keyword. Static fields are also called static variables.

Static member variable allows a common value to be used for the entire class. The properties of static variables are,

(i) The initial value that is assigned (at the first object creation) to the static member variable is 0. No further assignments are allowed.

(ii) Single copy of it is shared among all the objects.

(iii) It is local to the class in which it is declared and remains active till the completion of execution of the entire program.

Syntax

static datatype variable_name;

Static Method

A method is said to be static when its declaration or definition is preceded with static keyword.

Static Member Functions

A static member function is declared using the keyword static and has the following characteristics,

(i) A static function can access only the static members of its class.

(ii) Calling a static member includes the name of the class as shown below.

class_name::Static function_name

Syntax

static datatype method_name( ) { }

Q64. How classes can be nested? Explain.

Answer :

Nested Class

A class which can be defined in another class is called as nested class. A nested class can be classified into two types. They are,

(a) Static nested class

(b) Non-static nested class.

(a) Static Nested Class

A nested class which can be declared as static is called as static nested class.

(b) Non-static Nested Class

A nested class which cannot be declared as static is called non-static nested class. It is also called as inner class.

The nested class can be called as a member of its outer class. The static nested class cannot access all the members of outer class. Whereas, inner class can access all the members of outer class including private members.

Syntax of Nested Class

```
class OuterClass
{
    //members of outer class
    class InnerClass
    {
        // members of inner class
    }
}
```

Example

```
class Outer
{
    static int stdId = 5;
    static class Inner
    {
```

```
        void innerMethod( )
        {
            System.out.println("The inner method in static nested class");
            System.out.println("Student's Id is:" + stdId);
        }
    }
    public static void main(String[ ] args)
    {
    Outer.Inner innerObject = new Outer.Inner( );
    innerObject.innerMethod( );
    }
}
```

Q65. Explain in detail about inner classes.

Answer :

Inner Class

A class which is defined inside another class is known as the inner class. Inner class are used inorder to hide itself from other classes of same package. An inner class object can access the implementation of the object from which it is created. Inner classes are capable of inheriting the class members of the outer class. It manipulates the objects of outer class as if it is created in the inner class itself. In addition to this it can also implement interfaces. It does not care if the object inheriting is used or implemented by the outer class. It can also implement from multiple sources.

Inner classes are considered to be important and mostly used due to the following features,

1. An inner class contains multiple instances each holding the state information of it separately without the concern of outer-class object.
2. The creation of inner class object is not at all related with the outer class object.
3. An outer class can contain multiple inner classes each of which inherits the interface from the outer class in a unique way.
4. An inner class is an independent entity without maintaining any "is-a" relationship.

Example

```
class Outer
{
    private int stdId = 5;
    class Inner
    {
        public void innerMethod( )
        {
            System.out.println("The method in inner class");
            System.out.println("Student's Id which is declared as private is :" + stdId);
        }
    }
    public static void main(String[ ] args)
    {
        Outer.Inner innerObject = new Outer.Inner( );
        innerObject.innerMethod( );
    }
}
```

Q66. Explain how a string can be explored.

Answer :

String

A sequence of characters together is called as a string. Strings in java are class objects. They are implemented through the use of classes such as String and StringBuffer. They are reliable and predictable when compared to that of other languages. A string can be declared as follows,

```
String stringName;
StringName = new String("string");
        or
String stringName = new String("string");
```

Example

```
import java.lang.*;
class Demo
{
    public static void main(String args[ ])
    {
        String S1;
        S1 = new String("Sia Group");
        String S2 = "Computers";
        System.out.println(S1);
        System.out.println(S2);
    }
}
```

Methods of String Class

Some of the methods of string class are as follows,

1. **length( )**

   This method is used to know the length of the string.

   Syntax

   int length( )

2. **equals( )**

   This method is used to compare two strings for equality.

   Syntax

   boolean equals(String obj)

3. **charAt(int index)**

   This method is used to extract the character from the specified index in the string.

   Syntax

   char charAt(int index)

4. **string( )**

   This method is used to initialize an object with specified characters or characters.

   Syntax

   String(char[ ] data)

   String (String str)

5. **concat( )**

   This method is used to append one string at the end of another string.

   Syntax

   String concat(String str)

Q67. What are command line arguments? How are they useful?

Answer :

A command-line argument is the information passed to a program along with a command on the command line when the program is executed. These arguments directly follows the program's name each separated with a space. These are stored as strings in the string array which is passed as parameter to main( ) method. So, they can be easily accessed in a java program from the string array.

Example

The following program accepts the command-line arguments and display the same.

```
class cmdLineArgs
{
    public static void main(String args[ ])
    {
        System.out.println("The command line
                                    argumentsare:");
        for(int c = 0; c<args.length; c++)
        System.out.println(args[c]);
    }
}
```

Output

```
The command line arguments are:
hello
2
all
of
you
```

In the above program the string array args accepts the command line argument. It can be any void name. All the argument passed at command line are stored as strings, to manipulate numeric values they must be converted to their internal forms for example strings must be converted to integer using static method parseInt( ) of an abstract class Integer.

Explain about variable length arguments in Java.

...wer :

...able Length Arguments

The method that accepts the variable number of ...uments is called varargs methods. The variable-length ...uments are in short called as varargs. They are not used ...ently in Java. For example, consider a method to open ...ternet connection. It might accept arguments like name, ...sword, filename, protocol etc. This method supplies default ...e if any of the arguments are not provided. Then it would be ...er to pass arguments for which default does not apply.

Variable-length arguments can be handled in two ways. ...e way is overloaded versions of a method can be created for ...y method call if maximum number of arguments are small ...known. But, this can only be applied to very less situations. ...e second way is to gather the arguments into array and then ...s this array to a method.

...mple

```
class VarArguments
{
static void Test(int ...x)
{
    System.out. print("Arguments:" + x.length);
    for(int i : x)
    System.out.print(i + " ");
    System.out.println( );
}
public static void main(String args[ ])
{
    VarArguments(5);
    VarArguments(6, 7, 8);
    VarArguments( );
}
}
```

In the above program, x in Test( ) is operated as an array. ...re the .... specify to compiler that variable length arguments ...ll be used. They will be stored in c. In main( ) method the ...st(1) method is called with various number of arguments. The ...guments are gathered in x automatically.

...strictions in Varargs

- Normal parameters are also allowed to be used with variable length parameters, but it should be declared at last.
- Attempting to declare a normal parameter after varargs is illegal.
- Only one varags parameter is allowed in an array
- If any ambiguity raises in overloading a method accepting varargs then it is better to use different method names.

1.1.13 String Handling

Q69. Discuss about string handling functions.

Answer : Model Paper-I, Q3(b)

String Handling Methods

The most commonly used string methods provided by Java are discussed below,

1. **String( ):** This method is used to create a string object.
2. **length(str):** This method is used to return the length of the given string.
3. **charAt( ) (int location_index):** This method is used to return the character present at the specified location.
4. **subString(int startindex, int endindex):** This method is used to return the substring starting at startindex and ending at endindex in a string.
5. **endswith(String string_to_match):** This method is used to determine if the end of the string matches with string_to_match.
6. **IndexOf(String substring):** This method is used to return the index of the first occurrence of the substring in the string.
7. **replace(char originalchar, char replacementchar):** This method is used to replace the originalchar with the replacementchar in the string.
8. **Concat(String str):** This method is used to append one string at the end of the another string.
9. **toLowerCase( ):** This method is used to convert the string to lower case.
10. **toUpperCase( ):** This method is used to convert the string to uppercase.
11. **trim( ):** This method is used to remove the leading and trailing spaces.
12. **compareTo(String string_to_match):** This method is used to compare the given two strings.
13. **append(String str):** This method is used to append the given string at the end of string buffer.
14. **delete(int start, int end):** This method is used to delete the string starting at start and ending at end in the string.
15. **insert(int offset, String str):** This method is used to insert the string into the string buffer.
16. **reverse( ):** This method is used to reverse the string in string buffer.
17. **equals( ):** This method is used to compare two strings for equality.

Q70. Explain string comparison functions with examples.

Answer : Nov./Dec.-17(R13), Q6(b)

The String class provides several methods to compare strings or substrings. These methods are discussed here.

1. equals and equalsIgnoreCase( )

Two strings are compared for quality using equals( ) method. equals( ) is a case-sensitive method i.e., upper case letters are not equal to lower case letters e.g. a is not equal to A.

General Form

boolean equals(Object str)

Here, 'str' is the String object to be compared with the invoking String object. It returns true if both the strings are equal and false otherwise.

The equalsIgnoreCase( ) method compares the two strings by ignoring the case i.e., it considers A-Z to be same as a-z.

General Form

boolean equalsIgnoreCase(String str)

It returns true if two strings are equal else returns false.

2. regionMatches( )

This method is used to compare a region of a string with another region of another string.

There are two forms of this method. They are,

boolean regionMatches(int startIndex, String str2, int str2StartIndex, int numChars)

boolean regionMatches(boolean ignoreCase, int StartIndex, String str2, int Str2StartIndex, int numChars)

In both the forms 'startIndex' specifies index at which the region for invoking object begins 'str2' specifies the String object which is being compared and the starting region for this object is specified by 'str2StartIndex'.'numChars' specifies the length of the substring to be compared.

The first form compares two string with case-sensitive and the second form compares without case-sensitive, it is specified using 'ignoreCase'.

3. startsWith( ) and endsWith( )

These two methods are used to determine whether the given string starts with or ends with the string passed as an argument. Both of these returns true if the string matches and false otherwise.

General Form

boolean startsWith(String str)

boolean startsWith(String str, int startIndex)

boolean endsWith(String str)

Here, 'str' is the string being tested. The startsWith( ) method has two forms. The second form also specifies the position of the string from which to start comparing the string.

Example

```
Football.endsWith("all");
Football.startsWith("Foot");
Football.startsWith("all", 5);
```

All of these return true.

4. compareTo( )

This method is used to determine whether the string is less than, equal to or greater than the invoking string.

A string is less than another string if it comes before the other in alphabetical order. And a string is greater than another string if it comes after the other in alphabetical order.

General Form

int compareTo(String str)

Here, 'str' is the string being compared with another string. It returns the values as shown below.

Value	Meaning
Less than zero	It means invoking string is less than str.
Greater than zero	It means invoking string is greater than str.
Zero	It means both the strings are equal.

The two String objects can also be compared using = = operator. It determines whether the two string objects refer to the same instance.

Example

Following is an example program that shows the use of these methods for string comparison.

```
public class CompareStrings
{
    public static void main(String args[ ])
    {
    String s1 = new String("hello");
    String s2 = new String("world");
    String s3 = new String("How Are You");
    String s4 = new String("how are you");
    System.out.println("string s1 =" +s1);
    if(s1.equals("hello"))
    System.out.println("Both are equal");
    else
    System.out.println("They are not equal");
    if(s1 = "hello")
    System.out.println("Both are equal");
    else
    System.out.println("They are not equal");
    if(s3.equalsIgnoreCase(s4))
    System.out.println("s3 equals s4");
    else
    System.out.println("s3 is no equal to s4");
    System.out.println("The returned value is:" + s1.compareTo(s2));
    System.out.println("The value is :" +s2.compareTo(s1));
    }
}
```

1.2 INHERITANCE

1.2.1 Inheritance Concept, Inheritance Basics, Member Access

Q71. Define inheritance. Write about superclass and subclass. Model Paper-II, Q3(a)

OR

What is inheritance and how does it help to create new classes quickly?

Answer : April/May-18(R16), Q2(a)

Inheritance

Inheritance can be defined as a mechanism where in one class inherits the features of another class. The class which acquires the features is called the child class or derived class and the class whose features are acquired is called parent class or base class. The parent class contains only its own features, where as the child class contains the features of both parent and child classes. Moreover, an object created for parent class can access only parent class members. However, child class object can access members of both child class and as well as parent class. The child class can acquire the properties of parent class using the keyword 'extends'.

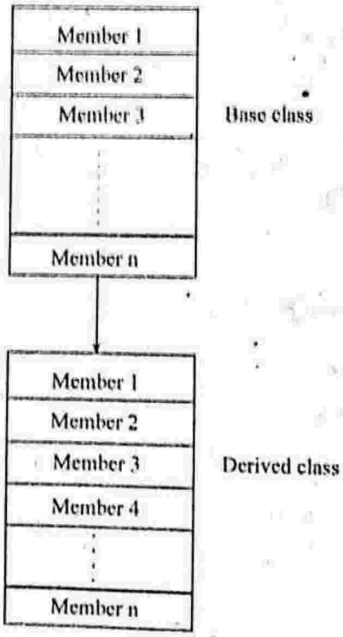
User can create the new classes by using the existing classes. In a new class, the methods and members can be inherited from existing classes. The user can add some extra members or methods in a new class. Thus, in this way the new can be created quickly.

SuperClass

Superclass is a class from which another class is inherited. It is also called as 'superclass' or 'parent class'. A base class does not have knowledge about its subclasses.

Sub Class

Subclass is a class that is inherited from a base class. It contains additional members apart from having base class members. It is also called 'subclass' or a 'child class'.



The rules that are to be followed while using super class and subclasses are as follows,

1. The subclasses are restricted to directly access the private members of its superclass.
2. The subclasses are allowed to access only the public members of its superclass.
3. The subclasses contain either additional data or additional members or both of them.
4. The subclasses are capable of overriding the methods of superclass which are publicly defined.
5. The data member defined in superclass can also be used in subclasses.
6. The methods defined in superclass can also be used in subclasses if they are not overridden.

Example

```
class Student
{
    String sname= "John";
    int sid="20";
}
class studentdetails extends Student
{
    display( )
    {
        System.out.println("Name:" + sname + "studentid" + sid);
    }
}
class Demo
{
    public static void main(String args[ ])
    {
        studentdetails s=new studentdetails( );
        s.display( );
    }
}
```

Q72. Discuss in brief about the different member access specifiers used in Java.

Answer :

For answer refer Unit-I, Q62.

Q73. How a superclass variable can reference a subclass object?

Answer :

A reference variable of a superclass can be assigned a reference to any subclass derived from that superclass. This aspect of inheritance is very useful in different situations. For example, consider the following program. In this, 'weightbox' is reference to 'BoxWeight' objects, and 'plainbox' is reference to 'Box' objects because 'BoxWeight' is a subclass of 'Box', it is permissible to assign 'plainbox' a reference to the 'BoxWeight' object. We must keep one important point in mind that it is the type of reference variable which determines what members can be accessed and not the type of objects which it refers to. When a reference to a subclass object is assigned to a superclass reference variable, we will have access only to those parts of the object defined by the superclass.

```
class Refer
{
public static void main(String args[ ])
    {
    BoxWeight weightbox = new BoxWeight(2, 4, 5, 7.37);
    Box plainbox = new Box( );
    double vol;
    vol = weightbox.volume( );
    System.out.println("Weightbox's volume is" + vol);
    System.out.println("Weightbox's weight is" + weightbox.weight);
    System.out.println( );
    plainbox = weightbox;
    vol = plainbox.volume( );
    System.out.println("Plainbox's volume is" + vol);
    }
}
```

1.2.2 Constructors

Q74. Explain about constructors with an example program.

OR

What is a constructor? What are its special properties?

(Refer Only Topics: Constructor, Properties)

Answer : Nov./Dec.-16(R13), Q2(b)

Constructor

A constructor is a special member function that gets executed on the object creation automatically.

Need for Constructors

A constructor is a special function that is needed to avoid unexpected results caused by un-initialized variables. Generally, variables must be initialized before they are used for processing because, uninitialized variables typically contain garbage values. The constructor for the class gets automatically invoked every time a new object is created.

Properties

1. Constructors are used to initialize objects of its class and allocate suitable memory space to objects
2. They take the same name of the class to which they belong.
3. They doesn't have return type and void, so it cannot return any values.
4. The execution of constructor take place when an object is declared
5. Even on implicit execution of constructors they can be called explicitly.
6. They are not virtual
7. They are usually declared in the public section.

Every class has constructor. It does not have any explicit return type. Java provides a default constructor for every class. When the programmer defines a constructor, the default constructor will not be used.

Example

```
class A
{
    int i;
    A( )
    {
        i = 2;
    }
}
class Demo
{
    public static void main(String[ ] args)
    {
        A a = new A( );
        A b = new A( );
        System.out.println(a.i + " " + b.i);
    }
}
```

Q75. Explain the different types of constructors in Java.

OR

Define constructor. Explain about parameterized Constructors.

(Refer Only Topics: Constructor, Parameterized Constructor)

Answer : Nov./Dec.-17(R13), Q2(a)

Constructor

For answer refer Unit-I, Q74, Topic: Constructor.

There are two different types of constructors in java,

1. Default constructor
2. Parameterized constructor.

1. Default Constructor

A constructor that does not take any argument is known as default constructor. If no constructor is specified then a default constructor is called automatically. It does not accept any argument.

Example

```
class employee_id
{
    employee_id( )
    {
        System.out.println("Default constructor for
                                class is created");
    }
    public static void main(String args[ ])
    {
        employee_id E = new employee_id( );
    }
}
```

2. Parameterized Constructor

A constructor that takes arguments as its parameters is known as parameterized constructor. It is used to initialize the elements that takes different values. Whenever objects are declared then their initial values are passed as arguments in constructor function. This can be done using the following two methods.

(a) Explicit constructor calling

(b) Implicit constructor calling.

Example

```
class A
{
    int i;
    A(int x)
    {
        i = x;
    }
}
```

```
class Demo
{
    public static void main(String[ ] args)
    {
        A a = new A(20);
        A b = new A(30);
        System.out.println(a.i + " " + b.i);
    }
}
```

Q76. Is it possible to define constructors and destructors in base class and derived class. If yes, illustrate it with an example.

Answer :

The base class as well as derived class can contain constructors and destructors.

The constructor belonging to the base class is executed first and then the constructor belonging to derived class is executed. But the destructors are executed in the reverse order. The destructor of the derived class will be executed first followed by the base class destructor. This means the constructors are executed the order in which they are executed and the destructors are executed in the reverse order of their derivation. The same thing will be applied even for multiple inheritance.

Example

```
class X
{
    X( )
    {
        System.out.println("X's constructor");
    }
}
class Y extends X
{
    Y( )
    {
        System.out.println("Y's constructor");
    }
}
class Demo
{
    public static void main (String args[ ])
    {
    Y y = new Y( );
    }
}
```

Output

```
X's constructor
Y's constructor
```

In the above program the constructors get executed in the order of derivation.

1.2.3 Creating Multilevel Hierarchy, Super Uses, Using Final with Inheritance

Q77. Explain about multilevel inheritance.

Answer :

Multilevel Inheritance

The process of deriving a class from another derived class is called multilevel inheritance. In this type of inheritance, a derived class can act as the base class for some other derived class.

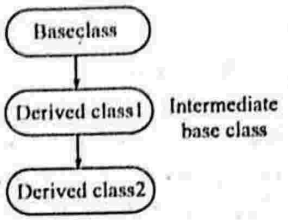


Figure: Multilevel Inheritance

Syntax

```
class A
{
}
class B extends class A
{
}
class C extends class B
{
}
```

Example

```
class Student
{
    String sname = "John";
    int sid = "101";
}
class Department extends Student
{
    String dept = "IT";
}
class Marks extends Department
{
    int sub1, sub2, sub3;
    sub1 = 50;
    sub2 = 80;
    sub3 = 40;
    void total( )
    {
        int total = sub1 + sub2 + sub3;
    }
```

```
    void display( )
    {
        System.out.println("student name:" + sname +
        "\n student id" + sid + "Department" + dept
        + "\nMarks of subject1" + sub1 + "Marks
        of subject2" + sub2 + "Marks of subject3"
        + sub3);
    }
}
class Multilevel
{
    public static void main(Strings args[ ])
    {
        Marks m = new Marks( );
        m.total( );
        m.display( );
    }
}
```

Q78. Define the variable 'super' and its uses with an example. Nov./Dec.-16(R13), Q5(a)

OR

Describe uses of super keyword in inheritance.

Answer : Nov./Dec.-17(R13), Q4(b)

'super'

'super' is a keyword used by subclass to refer to its immediate superclass.

Uses

There are two uses of 'super'-keyword,

(i) It is used to call the superclass constructor.

(ii) It is used to access a member of a superclass.

(i) 'super' Keyword to Call Superclass Constructor

A subclass can call the constructor of superclass by using the following form of 'super'.

```
super(parameter-list);
```

Here, parameter-list are parameters needed by the superclass' constructor.

The call to the 'superclass' constructor is made inside the 'subclass' constructor and hence the super( ) must always be the first statement inside a 'subclass' constructor. The program below illustrates the use of super( ) to call 'superclass' constructor.

Example

```
class A
{
private double width, height, depth;
A(double w, double h, double d)
{
    width = w;
    height = h;
    depth = d;
}
double volume( )
{
    return width * height * depth;
}
}
class B extends A
{
double weight;
B(double w, double h, double d, double m)
{
    super(w, h, d); // calls superclass constructor
    weight = m;
}
}
class FirstUse
{
public static void main(String args[ ])
{
    B b = new B(20.5, 10.2, 15.1, 50.6);
    System.out.println("volume is :" + b.volume( ));
}
}
```

Output

volume is: 3157.41

Here, class B calls super( ) with w, h and d parameters. This causes the class A constructor to be called where the values of width, height and depth are initialized using these values. This makes the class A to declare its variables private if desired.

(ii) 'super' to Access Members

The 'super' keyword can be used to access the members of a superclass, where members can either be a method or an instance variable. To access the members it has the following general form,

```
super.member
```

This 'super' usage is some what similar to 'this', except that it always refers to the superclass.

This second usage is useful in situations where the member names of subclass hides the members of superclass.

Example

```
class A
{
int x;
void show( )
{
System.out.println("This is superclass' method");
}
}
```

```
class B extends A
{
   int x;  // hides x in A
B(int a, int b)
{
   super.x = a;    // x in A
   x = b;          // x in B
}
void show( ) // hides show( ) in A
{
   super.show( );
System.out.println("This is subclass' method");
System.out.println("x in superclass :" + super.x);
System.out.println("x in subclass :" + x);
}
}
class SecondUse
{
public static void main(String args[ ])
{
   B b = new B(10, 20);
   b.show( );
}
}
```

Output

This is superclass' method
This is subclass' method
x in superclass : 10
x in subclass : 20

In the above program, B's constructor initializes the instance variable x of superclass using super keyword, where super refers to superclass' instance variable. An object b of class B calls its show( ) method, where a call to superclass show( ) method is made using the super keyword as follows,

```
super.show( )
```

Q3. Explain about final class and methods.

OR

Explain usage of final with inheritance.

Answer : Nov./Dec.-17(R13), Q5(a)

Final Class

The class which cannot be inherited or extended is called final class. A final class can be created by proceeding the class name with final keyword in its definition. This is done to prevent them from being inherited. If these classes are tried to be inherited then the compiler generates errors. Declaring a class as final will automatically declare its methods as final.

Example

```
final class Baseclass
{
    System.out.println("Baseclass");
}
class Derivedclass extends Baseclass //error
{
    System.out.println("Derivedclass");
}
```

Final classes are used to present the change in implementation resulted by subclassing. Since overriding of methods is not done, performance of final classes is greatly increased.

Final Method

The method which cannot be overridden is called final method. A final method can be created by proceeding the method name with final keyword in its definition. This is done to prevent it from being overridden. If these methods are tried to be overridden the compiler generates an error.

Example

```
class Baseclass
{
    final void display( )
    {
        System.out.println("Hello");
    }
}
class Derivedclass extends Baseclass
{
    void display( ) //error
    {
        System.out.println("Welcome");
    }
}
```

Execution of final method is efficient compared to the execution of non-final methods. This is because the compiler is already aware of the fact that a call to final method will not be overridden by any other method. Moreover, the methods which are private are considered to be final by default.

1.2.4 Polymorphism – Adhoc Polymorphism, Pure Polymorphism

Q80. What is polymorphism? Explain different types of polymorphism with examples.

April/May-18(R16), Q3(b)

OR

How do we implement polymorphism in JAVA? Explain briefly.

Answer : Nov./Dec.-18(R16), Q3(a)

Polymorphism

Polymorphism is an important concept in OOP. It is derived from the Greek word poly (multiple) and morphism (forms) which together mean multiple forms. It is a method through which an operation/function can take several forms based on the type of objects. An individual operator/function can be used in multiple ways.

Typeso f Polymorphism

There are two types of polymorphism, they are,

(i) Adhoc polymorphism

(ii) Pure polymorphism.

(i) Adhoc Polymorphism

Adhoc polymorphism is the concept in which different functions have similar/same name. This is also referred as "method overloading".

In Java several methods are allowed to be defined with same name. But they should differ in their arguments and definitions. All these methods perform similar tasks with different input parameters. This concept is called as method overloading. When a method is called, the compiler will match the method name, number of arguments and their type. Based on this it decides which method should be executed. Sometimes same definition is also allowed for different methods. The following is the example for adhoc polymorphism.

```
class Method
{
    void Sum(int x, int y)
    {
        int res = x + y;
        System.out.println("Sum:" + res);
    }
    void Sum(double x, double y)
    {
        double res = x + y;
        System.out.println("Sum:" + res);
    }
}
```

In the above example, the class method contains the function sum( ) with different arguments passed to it.

(ii) Pure Polymorphism

Pure polymorphism contains only one function with different types of arguments/parameters. This is useful in object-oriented programming. The code is flexible as it is written only one time and can be used in any situation. It supports abstraction. The flexibility can be achieved by forwarding the messages to receiver. The later messages that are received are not related with the class at higher level of polymorphic method and the deferred methods are used in lower classes.

The following is the example for pure polymorphism.

```
public class string
{
    public static string valueOf(Object o)
    {
        if(o == = NULL)
        return "NULL";
        return o.toString( );
    }
}
```

In the above example, the class string is defined to contains the method valueOf( ). It generates the output as a textual description for an object o. Another class object is defined that has a method toString( ) used in various subclasses to produce distinct effects. For example, double is used to generate the textual representation of numbers colour is used to represent the values of a different strings in different colours like red, yellow, green, blue. The method valueOf( ) will also generate the different outputs, even though it is defined only once.

Q81. Illustrate dynamic binding with a Java program.

Answer : Nov./Dec.-16(R13), Q3(b)

Dynamic Binding

The process of connecting the function call with its body is called binding. If there are various functions with same name and same definition. Then the compiler gets confused as to which function to be called while execution. Function to be called is determined only by the object. But object will be existing only at runtime. So, binding should be done at runtime. This type of binding is called as runtime binding or late binding or dynamic binding.

Example

```
class Baseclass
{
    void display( )
    {
        System.out.println("Baseclass");
    }
}
```

```
class Derivedclass
{
    void display( )
    {
        System.out.println("Derivedclass");
    }
    public static void main(String args[ ])
    {
        Baseclass bc= new Baseclass( );
        bc.display( );
    }
}
```

Q82. How does polymorphism promote extensibility? Explain with example.

Answer : **Nov./Dec.-17(R16), Q3(b)**

For answer refer Unit-I, Q80.

The software that invokes polymorphic behavior does not depend upon the object types to which the messages are sent. The new object types that respond to the exiting method calls can be included in the system without the need for base system modification. The client code that initiates the new objects need to be modified so that it accommodates new types.

The polymorphism concept makes the programs more extensible if all the method calls are made generic. When any new class type is added with abstract method to the hierarchy, then no changes are required to be made to the generic calls.

Using the polymorphism, the user can design as well as implement the systems which are easily extensible. The new classes can be added without the need of modification to the general portions of the program when the new classes belong to inheritance hierarchy so that program processes generically.

The parts that direct the new classes knowledge are only needed to be modified. Consider the below example,

```
class flower
{
    class Rose
    {
    =
    }
}
```

In the above program, if class Flower is extended to create a class Rose, then user need to write only one Rose class along with the simulation part that instantiates the Rose object.

The parts of program that indicate every Flower to move generically will remain the same. The class Rose will respond to the move message by crawling one inch.

1.2.5 Method Overriding, Abstract Classes, Object Class

Q83. How a method can be overridden? Explain.

Answer :

Method Overriding

Method overriding is a phenomenon in which a method in subclass is similar to the method in superclass. The return type and signature of a subclass method matches with the return type and signature of superclass method. The advantage of this method is that, it provides an implementation which is already provided by its superclass. This method can be used for runtime polymorphism. When the method of subclass is called the superclass's method is not referred and it is hidden.

Rules to be followed for method overriding are,

1. The argument list of subclass method and super class method must be same.
2. The access modifier of subclass method must not restrict the superclass method.
3. Static and final methods must not be overriden.
4. Constructors and methods which cannot be inherited cannot be overridden
5. Instance methods that are inherited by the subclass can be overridden.

Example

```
class Baseclass
{
    void display( )
    {
        System.out.println("Hello");
    }
}
class Derivedclass extends Baseclass
{
    void display( )
    {
        System.out.println("Welcome");
    }
}
class Method
{
    public static void main(String[ ] args)
    {
        Derivedclass dc = new Derivedclass( )
        dc.display( );
    }
}
```

In the above program, when the display( ) method is called, only the derived class's method is called and the output generated is "welcome". In order to call superclass's method super keyword is used while calling the function in main as shown below.

```
super.display( );
```

Q84. Explain the usage of abstract classes and methods.

OR

Write a Java program to show the use of abstract classes.

(Refer Only Topic: Example)

Answer : Nov./Dec.-16(R13), Q5(b)

Abstract Class and Abstract Method

Java defines abstract classes and methods using the keyword "abstract". The class that doesn't have body is called abstract class. Whereas the method that does not have body is called abstract method. An abstract class can contain subclasses in addition to abstract methods. If the method containing in the subclasses does not override the methods of the baseclass then they will be considered as abstract classes. So, all the subclass must provide an implementation for all the base class's abstract methods. Abstract classes cannot be instantiated.

A method is declared abstract when it has to be overridden in its subclasses. For example, an abstract method "Phone( )" can be defined in "Network" class as there can be various phones with various networks. Hence, each subclass of Network class can override and implement the abstract method phone( ). The syntax for abstract class and method is as follows,

Syntax for Abstract Class

```
abstract class classname{ }
```

Syntax for Abstract Method

```
abstract type methodname(arg1, arg2, - -, argn);
```

Example

```
abstract class Baseclass
{
    abstract display( );
}
class Derivedclass extends Baseclass
{
    void display( )
    {
        System.out.println("Derivedclass");
    }
}
class Demo
{
    public static void main(String args[ ])
    {
        Derived dc = new Derivedclass( );
        dc.display( );
    }
}
```

Q85. What is an object class? Discuss its methods.

Answer :

Object Class

Object class is said to be super class for all the other classes in Java. All the classes are said to be the subclasses of object class. Object class can be used to refer an object whose type is not known a reference variable of type object class is created to refer all the objects of other classes. Arrays can also be implemented as classes. In this case a variable of type object is used to refer the other arrays. It defines the following objects.

1. **object clone( ):** This method is used to create an object with same features of the object that is cloned.
2. **boolean equals(object ob):** This method is used to determine if the object is equal to some other object.
3. **void finalize( ):** This method is called before an object that is not used is recycled.
4. **class<?>getclass( ):** This method is used to access the object's class at runtime.
5. **int hashcode( ):** This method is used to return the hash code of the object being invoked.
6. **void notify( ):** This method is used to resume the execution of the thread that waiting on the object that is being invoked.
7. **void notifyAll( ):** This method is used to resume the execution of all the threads that are waiting on the object that is being invoked.
8. **string toString( ):** This method is used to return the string that defines the object.
9. **void wait( ), void wait(long milliseconds) and void(wait long milliseconds, int nanoseconds):** These methods are used to impose wait on some other thread that is being executed.

1.2.6 Forms of Inheritance – Specialization, Specification, Construction, Extension, Limitation, Combination, Benefits of Inheritance, Costs of Inheritance

Q86. What are the different forms of inheritance? Explain any two of them.

Answer :

The various forms of inheritance are,

1. Specialization
2. Specification
3. Construction
4. Extension
5. Limitation
6. Combination.

Specialization

Inheritance is commonly used for specialization purposes. Here, a child class or a new class is a specialized form of the parent class and it conforms to all specifications of the parent. Thus, a subtype is created using this form and the substitutability is also maintained explicitly.

The subclassification example for specialization is an application window class created using inheritance. The following example class 'MineSweeperGame' inherits the 'Frame' class.

```
public class MineSweeperGame extends Frame
{
...
}
```

It is required to create an instance of the above class 'MineSweeperGame' to run the application. As this class extends the Frame class, several methods are inherited such as setTitle, setSize, show etc., and can thus be invoked. These methods manipulate the instances of the 'MineSweeperGame'class without knowing that they are not the instances of the 'Frame' class.

The methods defined in the child class will be executed by overriding the methods in the parent class when application specific task is to be performed.

2. Specification

Inheritance can also be used to allow the classes to implement those methods that have the same names. The parent class can define operations with or without implementations. The operation whose implementation is not defined in the parent class will be defined in the child class. Such kind of parent class that defines abstract methods is also called as "abstract specification class"

To support inheritance of specification, Java language provides two different techniques. They are,

(a) Using interfaces

(b) Using extensions.

(a) Using Interfaces

A method can describe the characteristics of an ActionListener without describing its implementation, as different applications implement it in different ways. Interfaces can thus be used to define only the requirements without defining the actual behavior.

```
interface ActionListener
{
public void actionPerformed(ActionEvent ev)
}
```

A listener class will be defined each time whenever a button is created. It will define a particular behavior for the method as required by the application.

```
class EmpireEarth extends Frame
{
private class ShootButtonListener implements
                                    ActionListener
{
public void actionPerformed(ActionEvent ev)
{
// Actions to be performed whenever a button is pressed
}
}
}
```

(b) Using Extension

When the classes are inherited using extensions, subclassification for specification is achieved. Subclasses are created using the keyword 'abstract'. The 'new' keyword cannot be used to create instances of an 'abstract' class. Apart from classes, the methods can also be declared as 'abstract'. To create instances these methods will have to be overridden.

Example

'Number' is an abstract class defined in Java Library and has the following description.

```
public abstract class Number
{
public abstract int intValue( );
public abstract long longValue( );
public abstract float floatValue( );
public abstract double doubleValue( );
public byte byteValue( );
{
return (byte) intValue( );
}
public short shortValue( );
{
return (short) intValue( );
}
}
```

The methods declared as abstract must be overridden by the subclasses of this class.

Q87. Explain the characteristics of inheritance for the following,

(a) Construction

(b) Extension

(c) Limitation

(d) Combination.

Answer :

(a) Construction

A child class can inherit most of the properties from its parent class, even if these classes do not have abstract concept in common.

The Java Library describes a vector class which can be used to construct a 'stack' class using inheritance.

```
class Stack extends Vector
{
public Object push(Object item)
{
addElement(item);
return item;
}
public boolean empty( )
{
return isEmpty( );
}
public synchronized Object pop( )
{
Object ob = peek( );
removeElementAT(size( ) –1);
return ob;
}
public synchronized Object peek( )
{
return ElementAT(size( ) –1);
}
}
```

Though, the vector concept and stack concept does not have much similarity, the vector class is made the parent of stack class as it makes the stack implementation more easier.

(b) Extension

The subclassification for extension is achieved if a child class adds an additional behavior to the parent class without modifying the attributes that are inherited from that parent class.

Example

The 'Properties' class defined in Java library inherits 'Hashtable' class. The Hashtable class helps the properties class in storing and retrieving relevant property name or value pairs. It also helps in property management, for example, in reading properties from a file or writing properties to a file.

```
class Properties extends Hashtable
{
...
public synchronized void load(InputStream ins)
                                    throws IOException
{
...
}
public synchronized void save(OutputStream outs,
                                    String header)
{
...
}
public String getProperty(String key)
{
...
}
public Enumeration propertyNames( );
{.....}
public void list(PrintStream outputs)
{.....}
}
```

The parent class functionality is made available to its child class without any modifications. Thus, such classes are also the subtypes because the subclassification for extension also supports the substitutability principle.

(c) Limitation

If a subclass restricts few behaviors to be used that are inherited from the parent class, then the subclassification for limitation occurs.

Example

A set class can be created from vector class. If only the set operations must be used on a set and with no vector operations then this restriction leads to subclassification for limitation. To implement this limitation, those methods should be overridden or a message should be printed denoting that they should not be used. Alternatively, an exception can also be thrown when the undesired methods are used, but the Java compiler does not permit to do so.

```
class Set extends Vector
{
// Methods inherited from vector are,
        addElement, removeElement, isEmpty, size
public int indexOf(Object ob)
{
System.out.println("Do not use set indexOf( )");
return (0);
}
public Object ElementAT(int index)
{
return null;
}
}
```

Subclassification for limitation must be avoided as it does not obey the substitutability principle and also the subclasses build by it are not subtypes.

(d) Combination

Java language does not allow a subclass to inherit more than one class. Thus, solution for this problem is to extend a parent class and implement any number of interfaces.

Example

```
class Hole extends Ball implements PinBallTarget
{
}
```

The above example implements only one interface, but multiple interfaces can also be implemented. For example, RandomAccessFile implements two interfaces, they are DataInput and DataOutput protocols.

Q88. What are the benefits of inheritance?

Answer : Model Paper-II, Q3(b)

The benefits of inheritance are as follows,

1. Increased Reliability

If a code is frequently executed then it will have very less amount of bugs, compared to code that is not frequently executed. When applications have same components within them, then the code exercised is more compared to code of a single application. It is very easy to find the bugs present in this kind of code. Thus, the applications that use this component are more error free. In addition to that the costs required in maintaining the shared components can be divided among multiple projects.

2. Software Reusability

Properties of a parent class can be inherited by a child class. But, it does not require to rewrite the code of the inherited property in the child class. For instance, searching of a string pattern, insertion of a new element into an existing table, etc., are used in several applications and can thus be inherited from one single class. Thus, using object- oriented programming, methods can be written once but can be reused.

3. Code Sharing

There are different levels in which code sharing can occur using object-oriented techniques. At one level of code sharing multiple projects or users can use a single class. If a single project contains more than one class that are inherited from the same parent class, then this is also said to be one level of code sharing. In this level of code sharing, there exist few object types that share the code that is inherited. This kind of code is written once and is used in several applications.

4. Software Components

Programmers can construct software components that are reusable using inheritance. The goal of inheritance is allowing new applications to be developed that requires very little or no coding at all. There is a huge collection of software components defined in Java library that helps in application development.

5. Consistency of Interface

When multiple classes inherit the behavior of a single superclass all those classes will now have the same behavior. Thus, objects that are similar have similar interfaces, as well and the user cannot be confused as all of them have same behavior.

6. Polymorphism and Frameworks

Traditionally, softwares produced were written bottom up mechanism. That is lower abstractions were placed at the bottom increasing gradually and the higher abstractions were placed at the top.

The code at the lower-level is more portable than the code at higher-level. The routines at lower-levels can be used in several applications but the routines at higher-level are very closely tied to a specific application.

In order to produce high-level components that are reusable and can be used with different applications by changing just the low-level routines, polymorphism is employed in many programming languages.

An example of such a large software framework is Java AWT that performs its operations based on substitutability and inheritance.

7. Information Hiding

When a software component is being reused by a programmer, he needs to understand only the interface and nature of that component. There is no need for him to know about the implementation of that component or the techniques used by that component. Thus, the interconnected nature between several software systems is reduced thereby reducing the complexity of software.

8. Rapid Prototyping

When most of the components that are used in building a software are inherited from the parent class or interfaces then only small amount of system requires actual coding. Thus, the software systems are constructed very easily and quickly and this kind of programming is known as Exploratory Programming or Rapid Programming

Q89. What are some of the costs of using inheritance for software development?

Answer :

Following are the costs of object oriented programming techniques or more particularly, the costs of inheritance.

(i) Program Size

If the cost of memory decreases, then the program size does not matter. Instead of limiting the program sizes, there is a need to produce code rapidly that has high quality and is also error-free

(ii) Execution Speed

The specialized code is much faster than the inherited methods that manage the random subclasses.

The efficiency of a system must not be considered in the early stages. Instead, when that system is developed, it should be monitored to find out where the execution time is more and then that part should be improved.

(iii) Program Complexity

Complexity of a program may be increased if inheritance is overused. To understand the program's control flow, the inheritance graph will have to be scanned several times, up and down, which will lead to a problem known as yo-yo problem.

(iv) Message-Passing Overhead

Although, passing of messages is more costlier than procedure invocation, the cost of message passing is very less when execution speed is considered. This increased cost is negligible when compared to the benefits provided by object-oriented programming technique.