

SYLLABUS

Object Oriented Programming - CS3391

UNIT I INTRODUCTION TO OOP AND JAVA

Overview of OOP - Object oriented programming paradigms - Features of Object Oriented Programming - Java Buzzwords - Overview of Java - Data Types, Variables and Arrays - Operators - Control Statements - Programming Structures in Java - Defining classes in Java - Constructors - Methods - Access specifiers - Static members - Java Doc comments. (Chapter - 1)

UNIT II INHERITANCE, PACKAGES AND INTERFACES

Overloading Methods - Objects as Parameters - Returning Objects - Static, Nested and Inner Classes. Inheritance : Basics - Types of Inheritance - Super keyword - Method Overriding - Dynamic Method Dispatch - Abstract Classes - final with Inheritance. Packages and Interfaces : Packages - Packages and Member Access - Importing Packages - Interfaces. (Chapter - 2)

UNIT III EXCEPTION HANDLING AND MULTITHREADING

Exception Handling basics - Multiple catch Clauses - Nested try Statements - Java's Built-in Exceptions - User defined Exception. Multithreaded Programming : Java Thread Model - Creating a Thread and Multiple Threads - Priorities - Synchronization - Inter Thread Communication - Suspending - Resuming and Stopping Threads - Multithreading. Wrappers - Auto boxing. (Chapter - 3)

UNIT IV I/O, GENERICS, STRING HANDLING

I/O Basics - Reading and Writing Console I/O - Reading and Writing Files. Generics : Generic Programming - Generic classes - Generic Methods - Bounded Types - Restrictions and Limitations. Strings : Basic String class, methods and String Buffer Class. (Chapter - 4)

UNIT V JAVAFX EVENT HANDLING, CONTROLS AND COMPONENTS

JAVAFX Events and Controls : Event Basics - Handling Key and Mouse Events. Controls : Checkbox, ToggleButton - RadioButtons - ListView - ComboBox - ChoiceBox - Text Controls - ScrollPane. Layouts - FlowPane - HBox and VBox - BorderPane - StackPane - GridPane. Menus - Basics - Menu - Menu bars - MenuItem. (Chapter - 5)

UNIT I

1

Introduction to OOP and Java

Syllabus

Overview of OOP - Object oriented programming paradigms - Features of Object Oriented Programming - Java Buzzwords - Overview of Java - Data Types, Variables and Arrays - Operators - Control Statements - Programming Structures in Java - Defining classes in Java - Constructors-Methods -Access specifiers - Static members- Java Doc comments.

Contents

Part I : Overview of OOP

1.1	Object Oriented Programming Paradigm.....	Dec.-11, 13, 18, May-12, 14, Marks 16
1.2	Features of Object Oriented Programming	Dec.-11, 13, 18, May-12, 14, Marks 16
1.3	Benefits and Drawbacks of OOP	
1.4	Applications of OOP	
1.5	Introduction to Java.....	Dec.-18, 19, May-19,Marks 7

Part II : Overview of JAVA

1.6	Structure of Java Program	
1.7	Java Tokens	
1.8	Variables	
1.9	Data Types.....	May-19,Marks 6
1.10	Operators	
1.11	Control Statements	May-19,Marks 7
1.12	The break and continue Statement	
1.13	Arrays.....	May-14, 19, Dec.-19,Marks 16
1.14	Defining Classes in Java.....	May-12, Dec.-10, 14,Marks 5
1.15	Constructor	
1.16	Methods	Dec.-14, 18,Marks 6
1.17	Access Specifiers.....	Dec.-11, 12,Marks 8
1.18	Static Members.....	Dec.-10, May-11, 12, 13,Marks 8
1.19	JavaDoc Comments	
1.20	Two Marks Questions with Answers	

Part I : Overview of OOP

1.1 Object Oriented Programming Paradigm

AU : Dec. 11, 13, 18, May-12, 14, Marks 16

- In object oriented programming approach there is a **collection of objects**. Each object consists of corresponding data structures and the required operations (procedures). This is basically the **bottom up problem solving approach**.
- The object oriented programming approach is developed in order to remove some of the flaws of procedural programming.
- OOP has a complete focus on **data** and not on the procedures. In OOP, the data can be protected from accidental modification.
- In OOP the most important entity called **object** is created.
- Each object consists of **data attributes** and the **methods**. The methods or **functions** operate on **data attributes**.

1.2 Features of Object Oriented Programming

AU : Dec. 11, 13, 18, May-12, 14, Marks 16

- There are various characteristics of object oriented programming and those are -
- | | | |
|--------------------|-----------------------|------------------|
| 1) Abstraction | 2) Object and Classes | 3) Encapsulation |
| 4) Inheritance and | 5) Polymorphism. | |

1.2.1 Abstraction

- **Definition :** Abstraction means representing only essential features by hiding all the implementation details. In object oriented programming languages like C++, or Java class is an entity used for data abstraction purpose.

- **Example**

```
class Student
{
    int roll;
    char name [10];
public:
    void input ();
    void display ();
}
```

In main function we can access the functionalities using object. For instance

```
Student obj;
obj.input ();
obj.display ();
```

Thus only abstract representation can be presented, using class.

1.2.2 Object

- Object is an instance of a class.
- Objects are basic run-time entities in object oriented programming.
- In C++ the class variables are called objects. Using objects we can access the member variable and member function of a class.
- Object represent a person, place or any item that a program handles.
- For example - If the class is country then the objects can be India, China, Japan, U.S.A and so on.
- A single class can create any number of objects.
- **Declaring objects -**

The syntax for declaring object is -

Class_Name Object_Name;

- Example

Fruit f1;

For the class **Fruit** the object **f1** can be created.

1.2.3 Classes

- A class can be defined as an entity in which data and functions are put together.
- The concept of class is similar to the concept of **structure** in C.
- **Syntax of class** is as given below

class name_of_class

{

private :

variables declarations;

function declarations;

public :

variable declarations;

function declarations;

} ; do not forget semicolon

- Example

class rectangle

{

private :

int len, br;

public :

void get_data ();

void area();

void print_data ();

};

• Explanation

- The class declared in above example is rectangle.
- The class name must be preceded by the keyword class.
- Inside the body of the class there are two keywords used private and public. These are called access specifiers.

St. No.	Class	Object
1.	For a single class there can be any number of objects. For example - If we define the class as River then Ganga, Yamuna, Narmada can be the objects of the class River.	There are many objects that can be created from one class. These objects make use of the methods and attributes defined by the belonging class.
2.	The scope of the class is persistent throughout the program.	The objects can be created and destroyed as per the requirements.
3.	The class can not be initialized with some property values.	We can assign some property values to the objects.
4.	A class has unique name.	Various objects having different names can be created for the same class.

1.2.4 Encapsulation

- Encapsulation is for the detailed implementation of a component which can be hidden from rest of the system.
- In C++ the data is encapsulated.
- Definition : Encapsulation means binding of data and method together in a single entity called class.
- The data inside that class is accessible by the function in the same class. It is normally not accessible from the outside of the component.

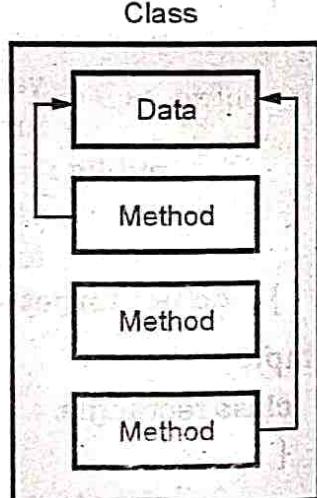


Fig. 1.2.1 Concept of encapsulation

Difference between Encapsulation and Abstraction

Sr. No.	Data encapsulation	Data abstraction
1.	It is a process of binding data members of a class to the member functions of that class.	It is the process of eliminating unimportant details of a class. In this process only important properties are highlighted.
2.	Data encapsulation depends upon object data type.	Data abstraction is independent upon object data type.
3.	It is used in software implementation phase.	It is used in software design phase.
4.	Data encapsulation can be achieved by inheritance.	Data abstraction is represented by using abstract classes.

1.2.5 Inheritance

- Definition :** Inheritance is a property by which the new classes are created using the old classes. In other words the new classes can be developed using some of the properties of old classes.
- Inheritance supports hierarchical structure.
- The old classes are referred as **base classes** and the new classes are referred as **derived classes**. That means the derived classes inherit the properties (data and functions) of base class.
- Example :** Here the **Shape** is a base class from which the **Circle**, **Line** and **Rectangle** are the derived classes. These classes inherit the functionality **draw()** and **resize()**. Similarly the **Rectangle** is a base class for the derived class **Square**. Along with the derived properties the derived class can have its own properties. For example the class **Circle** may have the function like **backgcolor()** for defining the background color.

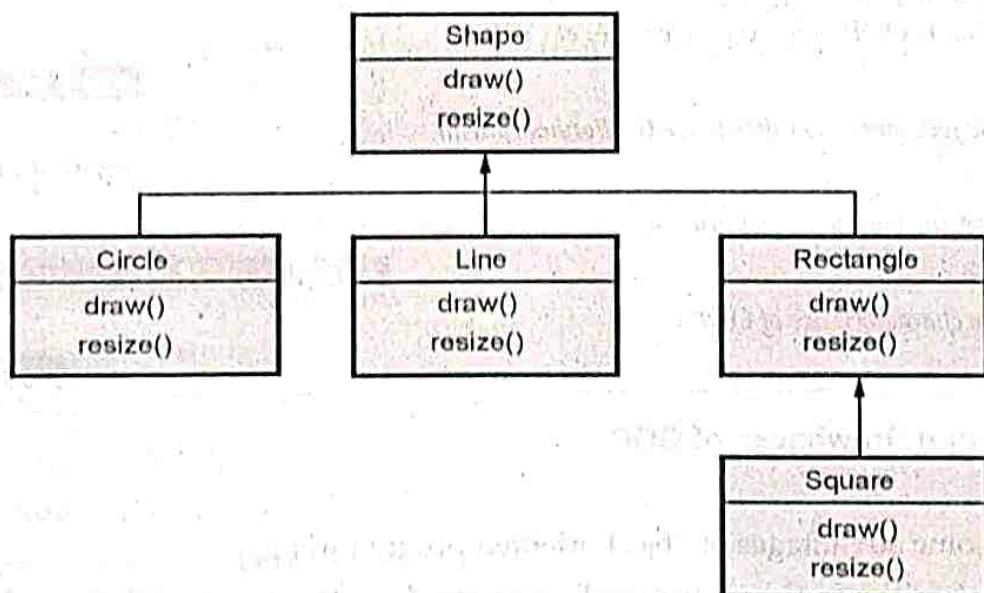


Fig. 1.2.2 Hierarchical structure of inheritance

1.2.6 Polymorphism

- Polymorphism means many structures.
- Definition : Polymorphism is the ability to take more than one form and refers to an operation exhibiting different behavior in different instances (situations).
- The behavior depends on the type of data used in the operation. It plays an important role in allowing objects with different internal structures to share the same external interface.
- Without polymorphism, one has to create separate module names for each method.
- For example the method **clean** is used to clean a **dish** object, one that cleans a **car** object, and one that cleans a **vegetable** object.
- With polymorphism, you create a single "clean" method and apply it for different objects.

Sr. No.	Inheritance	Polymorphism
1.	Inheritance is a property in which some of the properties and methods of base class can be derived by the derived class.	Polymorphism is ability for an object to used different forms. The name of the function remains the same but it can perform different tasks.
2.	Various types of inheritance can be single inheritance, multiple inheritance, multilevel inheritance and hybrid inheritance.	Various types of polymorphism are compile time polymorphism and run time polymorphism. In compile time polymorphism there are two types of overloading possible. - Functional overloading and operator overloading. In run time polymorphism there is a use of virtual function.

Review Questions

1. Explain what is OOPs and explain features of OOPs.

AU : Dec.-11, Marks 8

2. Discuss the following: (i) inheritance (ii) Polymorphism.

AU : May-12, Marks 16

3. Elaborate on the various object oriented concepts with necessary illustrations.

AU : May-14, Marks 16, Dec.-13, Marks 5

4. Explain the characteristics of OOPs.

AU : Dec.18, Marks 6

1.3 Benefits and Drawbacks of OOP

Benefits

Following are some advantages of object oriented programming -

1. Using inheritance the redundant code can be eliminated and the existing classes can be used.
2. The standard working modules can be created using object oriented programming.

- These modules can then communicate to each other to accomplish certain task.
3. Due to **data hiding** property, important data can be kept away from unauthorized access.
 4. It is possible to create **multiple objects** for a given class.
 5. For upgrading the system from small scale to large scale is possible due to object oriented feature.
 6. Due to **data centered nature** of object oriented programming most of the details of the application model can be captured.
 7. **Message passing technique** in object oriented programming allows the objects to communicate to the external systems.
 8. **Partitioning the code** for simplicity, understanding and debugging is possible due to object oriented and modular approach.

Drawbacks

Following are some drawbacks of OOP -

1. The object oriented programming is **complex to implement**, because every entity in it is an object. We can access the methods and attributes of particular class using the object of that class.
2. If some of the members are declared as **private** then those members are **not accessible** by the object of another class. In such a case you have to make use of inheritance property.
3. In Object oriented programming, every thing must be arranged in the forms of **classes** and **modules**. For the lower level applications it is not desirable feature.

1.4 Applications of OOP

Various applications in which object oriented programming is preferred are

- Business logic applications
- Knowledge based and expert systems
- Web based applications
- Real time systems
- Simulation and modeling
- Object oriented databases
- Applications in distributed environment
- CAD/CAM systems
- Office automation system
- Games programming

AU : Dec.-18, 19, May-19, Marks 7**1.5 Introduction to Java**

- Java was invented by James Gosling, Patrick Naughton, Chris Warth, Ed Frank and Mike Sheridan at Sun Microsystems, Inc. in 1991. It took 18 months to complete the first working version of Java.
- This language was initially called as Oak but was renamed as Java in 1995.
- Java is purely an object oriented programming language as it supports various features like Classes, Objects, abstraction, inheritance and polymorphism.

1.5.1 Java Buzzwords

Java is getting popular because of its features. Following is a list of buzzwords that makes Java a popular programming language.

- Simple
- Secure
- Platform independent
- Object oriented
- Distributed
- Portable
- High performance
- Robust
- Multithreaded
- Interpreted
- Dynamic

Let us discuss these features in detail.

1. Java is simple and small programming language

- Java is a simple programming language. Even though you have no programming background you can learn this language comfortably.
- The programmers who worked on C or C++ can learn this language more efficiently because the syntax of Java resembles with C and C++.
- In fact Java is made more clear and adaptable than C++.

2. Java is robust and secure

- Following are reasons that make Java more secured than other programming languages -
 - Java does not support the concept of pointers directly. This makes it impossible to accidentally reference memory that belongs to other programs or the kernel.

2. The output of Java compiler is not executable code but it is a bytecode. The instructions of bytecode is executed by the Java Virtual Machine(JVM). That means JVM converts the bytecode to machine readable form. JVM understand only the bytecode and therefore any infectious code can not be executed by JVM. No virus can infect bytecode. Hence it is difficult to trap the internet and network based applications by the hackers.
3. In programming languages like C or C++ the memory management is done explicitly by the user. That means user allocates or deallocates the memory. Whereas in java its automatically done using garbage collection. Thus user can not perform the memory management directly.
4. If an applet is executing in some browser then it is not allowed to access the file system of local machine.

3. Java is a platform independent and portable programming language

- Platform independence is the most exciting feature of Java program. That means programs in Java can be executed on variety of platforms. This feature is based on the goal - *write once, run anywhere, and at anytime forever.*
- Java supports portability in 2 ways - Java compiler generates the byte code which can be further used to obtain the corresponding machine code. Secondly the primitive data types used in Java are machine independent.

4. Java is known as object oriented programming language

- Java is popularly recognised as an object oriented programming language.
- It supports various object oriented features such as data encapsulation, inheritance, polymorphism and dynamic binding.
- Everything in Java is an object. The object oriented model of Java is simple and can be extended easily.

5. Java is multithreaded and interactive language

- Java supports multithreaded programming which allows a programmer to write such a program that can perform many tasks simultaneously.
- This ultimately helps the developer to develop more interactive code.

6. Java can be compiled and interpreted

- Normally programming languages can be either compiled or interpreted but Java is a language which can be compiled as well as interpreted.

- First, Java compiler translates the Java source program into a special code called bytecode. Then Java interpreter interprets this bytecode to obtain the equivalent machine code.
- This machine code is then directly executed to obtain the output.

7. Java is known for its high performance, scalability, monitoring and manageability

- Due to the use of bytecode the Java has high performance. The use of multi-threading also helps to improve the performance of the Java.
- The J2SE helps to increase the scalability in Java. For monitoring and management Java has large number of Application Programming Interfaces(API).
- There are tools available for monitoring and tracking the information at the application level.

8. Java is a dynamic and extensible language

- This language is capable of dynamically linking new class libraries, methods and objects.
- Java also supports the functions written in C and C++. These functions are called native methods.

9. Java is designed for distributed systems

- This feature is very much useful in networking environment.
- In Java, two different objects on different computers can communicate with each other.
- This can be achieved by **Remote Method Invocation(RMI)**. This feature is very much useful in Client-Server communication.

10. Java can be developed with ease

There are various features of Java such as Generics, static import, annotations and so on which help the Java programmer to create a error free reusable code.

Review Questions

1. Explain the features and characteristics of Java.
2. Discuss the three OOP principles in detail.
3. Explain the various features of java in detail.

AU : Dec.-18, Marks 7

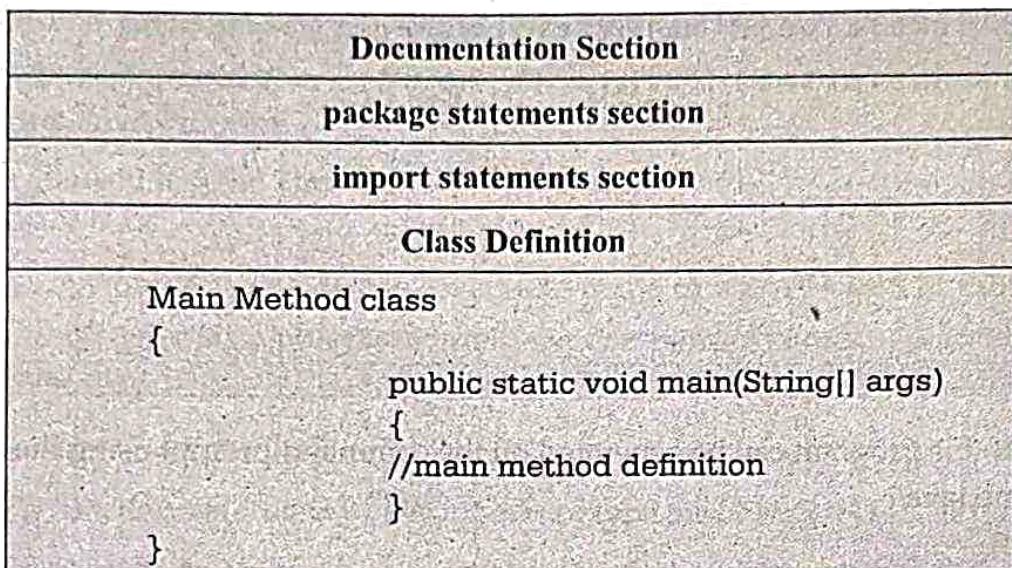
AU : May-19, Marks 7

AU : Dec.-19, Marks 5

Part II : Overview of JAVA

1.6 Structure of Java Program

- The program structure for the Java program is as given in the following figure -



Documentation section : The documentation section provides the information about the source program. This section contains the information which is not compiled by the Java. Everything written in this section is written as comment.

Package section : It consists of the name of the package by using the keyword **package**. When we use the classes from this package in our program then it is necessary to write the package statement in the beginning of the program.

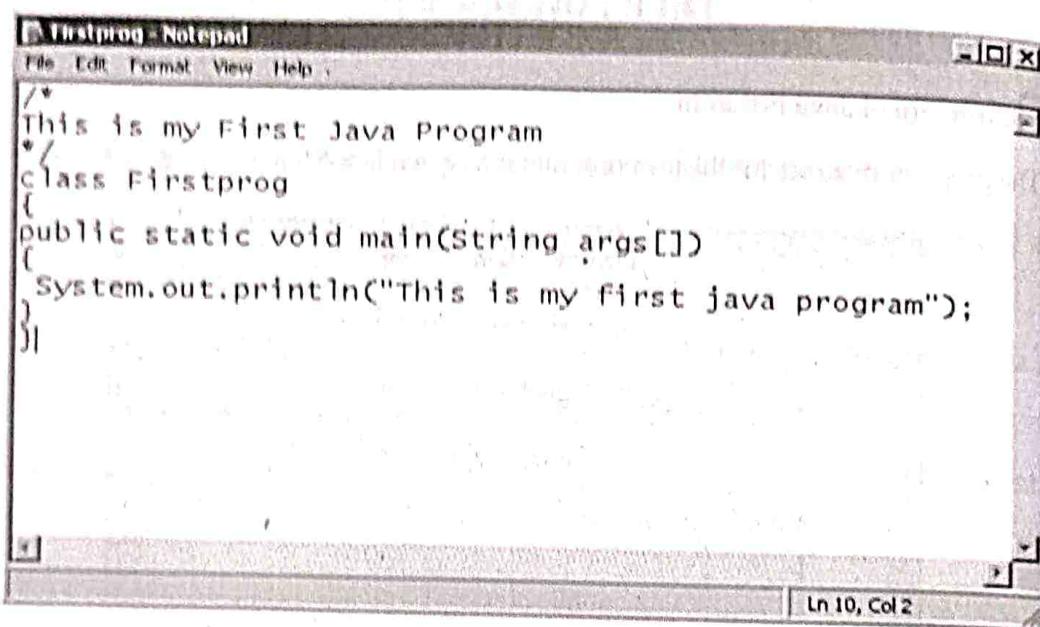
Import statement section : All the required java API can be imported by the import statement. There are some core packages present in the java. These packages include the classes and methods required for java programming. These packages can be imported in the program in order to use the classes and methods of the program.

Class definition section : The class definition section contains the definition of the class. This class normally contains the data and the methods manipulating the data.

Main method class : This is called the main method class because it contains the **main()** function. This class can access the methods defined in other classes.

1.6.1 Writing Simple Java Program

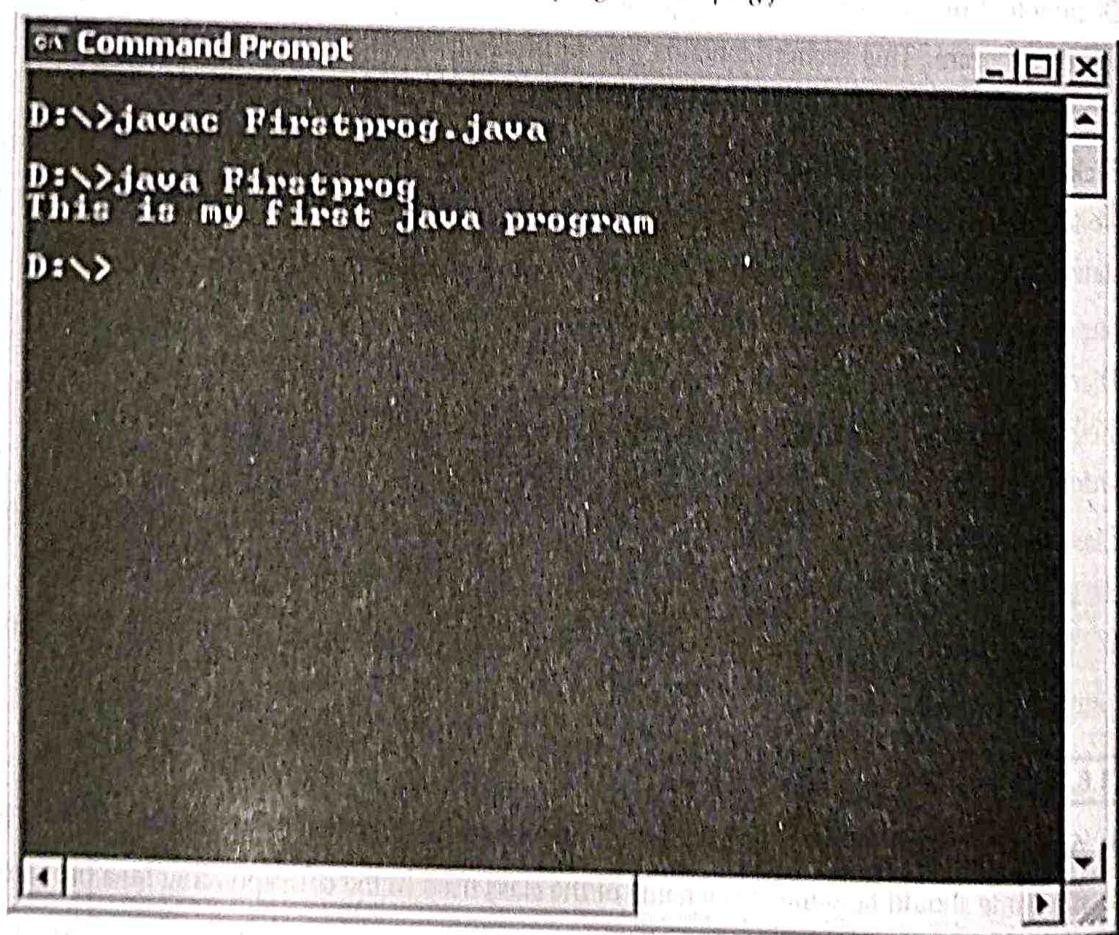
- Any Java program can be written using simple **text editors** like Notepad or Wordpad. The file name should be same as the name of the class used in the corresponding Java program. Note that the name of the program is *Firstprog* and class name is *Firstprog*. The extension to the file name should be **.java**. Therefore we have saved our first program by the name *Firstprog.java*.

Java Program[Firstprog.java]

```
Microsoft Notepad
File Edit Format View Help
/*
This is my First Java Program
*/
class Firstprog
{
public static void main(String args[])
{
System.out.println("This is my first java program");
}
}
Ln 10, Col 2
```

The output of this program can be generated on the command prompt using the commands
javac filename.java
java filename

The sample output is as shown below for the program *Firstprog.java*



```
Command Prompt
D:\>javac Firstprog.java
D:\>java Firstprog
This is my first java program
D:\>
```

Program explanation

- In our First Java program, on the first line we have written a comment statement as

```
/*
This is my First Java Program
*/
```

This is a multi-line comment. Even a single line comment like C++ is also allowed in Java.

Hence if the statement would be

```
//This is my First Java Program
```

- Is perfectly allowed in Java. Then actual program starts with
class Firstprog

Here *class* is a keyword and *Firstprog* is a variable name of class. Note that **the name of the class and name of your Java program should be the same**. The class definition should be within the curly brackets. Then comes

- public static void main(String args[])

This line is for a function **void main()**. The main is of type *public static void*.

The *public* is a access mode of the *main()* by which the class visibility can be defined. Typically *main* must be declared as *public*.

The parameter which is passed to main is *String args[]*. Here *String* is a class name and *args[]* is an array which receives the command line arguments when the program runs.

- System.out.println("This is my first java program");

To print any message on the console *println* is a method in which the string "This is my first java program" is written. After *println* method, the newline is invoked. That means next output if any is on the new line. This *println* is invoked along with *System.out*. The *System* is a predefined class and *out* is an output stream which is related to console. The output as shown in above figure itself is a self explanatory. Also remember one fact while writing the Java programs that it is a **case sensitive** programming language like C or C++.

1.7 Java Tokens

The smallest individual and logical unit of the java statements are called tokens. In Java there are five types of tokens used. These tokens are -

1. Reserved keywords
2. Identifiers
3. Literals
4. Operators
5. Separators

Reserved keywords

Keywords are the reserved words which are enlisted as below-

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

Identifiers

Identifiers are the kind of tokens defined by the programmer. They are used for naming the classes, methods, objects, variables, interfaces and packages in the program.

Following are the rules to be followed for the identifiers-

1. The identifiers can be written using alphabets, digits, underscore and dollar sign.
2. They should not contain any other special character within them.
3. There should not be a space within them.
4. The identifier must not start with a digit, it should always start with alphabet.
5. The identifiers are case-sensitive. For example -

```
int a;  
int A;
```

In above code a and A are treated as two different identifiers.

6. The identifiers can be of any length.

Literals

Literals are the kind of variables that store the sequence of characters for representing the constant values. Five types of literals are -

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

As the name suggests the corresponding data type constants can be stored within these literals.

Operators

Operators are the symbols used in the expression for evaluating them.

Separators

For dividing the group of code and arranging them systematically certain symbols are used, which are known as separators. Following table describes various separators.

Name of the Separator	Description
()	Parenthesis are used to - enclose the arguments passed to it to define precedence in the expression. For surrounding cast type
{ }	Curly brackets are used for - initialising array for defining the block
[]	Square brackets are used for - declaring array types dereferencing array values
;	Used to terminate the statement
,	Commas are used to separate the contents.
.	Period is used to separate the package name from subpackages.

1.8 Variables

A variable is an identifier that denotes the storage location.

Variable is a fundamental unit of storage in Java. The variables are used in combination with identifiers, data types, operators and some value for initialization. The variables must be declared before its use. The syntax of variable declaration will be -

data_type name_of_variable [=initialization][,=initialization][,...];

Following are some rules for variable declaration -

- The variable name should not contain digits.
- No special character is allowed in identifier except underscore.
- There should not be any blank space with the identifier name.
- The identifier name should not be a keyword.
- The identifier name should be meaningful.

For Example :

```
int a,b;
char m='a';
byte k=12,p,t=22;
```

The variables have a scope which defines their visibility and a lifetime.

AU : May-19, Marks 6

1.9 Data Types

Various data types used in Java are byte, short, int, long, char, float, double and boolean.

byte

This is in fact smallest integer type of data type. Its width is of 8-bits with the range -128 to 127. The variable can be declared as byte type as

```
byte i,j;
```

short

This data type is also used for defining the signed numerical variables with a width of 16-bits and having a range from -32,768 to 32,767. The variable can be declared as short as

```
short a,b;
```

int

This is the most commonly used data type for defining the numerical data. The width of this data type is 32-bit having a range 2,147,483,648 to 2,147,483,647. The declaration can be

```
int p,q;
```

long

Sometimes when *int* is not sufficient for declaring some data then *long* is used. The range of *long* is really very long and it is -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807. The declaration can be

```
long x,y;
```

float

To represent the real number(i.e. number that may have decimal point) float data type can be used. The width is 32-bit and range of this data type is 1.4e - 045 to 3.4e+038.

double

To represent the real numbers of large range the double data type is used. Its width is 64-bit having the range 4.9e-324 to 1.8e+308.

Char

This data type is used to represent the character type of data. The width of this data type is 16-bit and its range is 0 to 65,536.

Let us see one simple Java program which makes use of various data types.

boolean

Boolean is a simple data type which denotes a value to be either true or false.

Java Program

Note that for displaying the value of variable, in the *println* statement the variable is appended to the message using + [In C we use comma in *printf* statement but in Java + is used for this purpose]

Java Program [DatatypeDemo.java]

```
/*
This program introduces use of various data type
in the program
*/
class DatatypeDemo
{
    public static void main(String args[])
    {
        byte a=10;
        short b=10*128;
        int c=10000* 128;
        long d=10000*1000*128;
```

```

double e=99.9998;
char f='a';
boolean g=true;
boolean h=false;
}
}

dynamic initialization
}
}

```

Output

The value of a=: 10
The value of b=: 1280
The value of c=: 1280000
The value of d=: 1280000000
The value of e=: 99.9998
The value of f=: a
The value of f after increment=: b
The value of g=: true
The value of h=: false

Dynamic initialization

- Java is a flexible programming language which allows the dynamic initialization of variables. In other words, at the time of declaration one can initialize the variables (*note that in the above program we have done dynamic initialization*). In Java we can declare the variable at any place before it is used.

Review Questions

- What is a data type? Explain various data types available in Java.
- What are literals? Explain the types of literals supported by java.

AU : May-19, Marks 6

1.10 Operators

Various operators that are used in Java are enlisted in following table -

Type	Operator	Meaning	Example
Arithmetic	+	Addition or unary plus	c=a+b
	-	Subtraction or unary minus	d=-a
	*	Multiplication	c=a*b
	/	Division	c=a/b
	%	Mod	c=a%b
Relational	<	Less than	a<4
	>	Greater than	b>10
	<=	Less than equal to	b<=10
	>=	Greater than equal to	a>=5
	==	Equal to	x==100
Logical	!=	Not equal to	m!=8
	&&	And operator	0&&1
		Or operator	0 1
Assignment	=	is assigned to	a=5
Increment	++	Increment by one	++i or i++
Decrement	--	Decrement by one	--k or k--

Arithmetic Operators

The arithmetic operators are used to perform basic arithmetic operations. The operands used for these operators must be of numeric type. The Boolean type operands can not be used with arithmetic operators.

Java Program

```
////////////////////////////////////////////////////////////////
```

```
//Program for demonstrating arithmetic operators
```

```
////////////////////////////////////////////////////////////////
```

```

class ArithOperDemo
{
    public static void main(String[] args)
    {
        System.out.println("\n Performing arithmetic operations ");
        int a=10,b=20,c;
        System.out.println("a= "+a);
        System.out.println("b= "+b);
        c=a+b;
        System.out.println("\n Addition of two numbers is "+c);
        c=a-b;
        System.out.println("\n Subtraction of two numbers is "+c);
        c=a*b;
        System.out.println("\n Multiplication of two numbers is "+c);
        c=a/b;
        System.out.println("\n Division of two numbers is "+c);
    }
}

```

Output

Performing arithmetic operations

a= 10

b= 20

Addition of two numbers is 30

Subtraction of two numbers is -10

Multiplication of two numbers is 200

Division of two numbers is 0

Relational Operators

The relational operators typically used to denote some condition. These operators establish the relation among the operators. The <,>,<=,>= are the relational operators. The result of these operators is a Boolean value.

Java Program

```

//Program for demonstrating relational operators
import java.io.*;
import java.lang.*;
import java.util.*;
public class RelOper

```

```

{
    public static void main(String args[])
    {
        int a,b,c;
        a=10;
        b=20;
        if(a>b)
        {
            System.out.println("a is largest");
        }
        else
        {
            System.out.println("b is largest");
        }
    }
}

```

Logical Operators

The logical operators are used to combine two operators. These two operands are Boolean operators.

Java Program

```

//Program for demonstrating logical operators
import java.io.*;
import java.lang.*;
public class LogicalOper
{
    public static void main(String args[])throws IOException
    {
        boolean oper1,oper2;
        oper1=true;
        oper2=false;
        boolean ans1,ans2;
        ans1=oper1&oper2;
        ans2=oper1|oper2;
        System.out.println("The oper1 is: "+oper1);
        System.out.println("The oper2 is: "+oper2);
        System.out.println("The oper1&oper2 is: "+ans1);
        System.out.println("The oper1|oper2 is: "+ans2);
    }
}

```

Output

The oper1 is: true
 The oper2 is: false
 The oper1&oper2 is: false
 The oper1|oper2 is: true

Special Operators

- There are two special operators used in Java-**instanceof** and **dot** operators.
- For determining whether the object belongs to particular class or not- an **instanceof** operator is used. For example-
- Ganga instanceof River if the object Ganga is an object of class River then it returns true otherwise false.
- The **dot** operator is used to access the instance variable and methods of class objects.

For example -

Customer.name //for accessing the name of the customer
 Customer.ID //for accessing the ID of the customer

Conditional Operator

The conditional operator is "?" The syntax of conditional operator is

Condition?expression1:expression2

Where expression1 denotes the true condition and expression2 denotes false condition.

For example :

a>b?true:false

This means that if a is greater than b then the expression will return the value true otherwise it will return false.

1.11 Control Statements

AU : May-19, Marks 7

Programmers can take decisions in their program with the help of control statements. Various control statements that can be used in java are -

1. if statement
2. if else statement
3. while statement
4. do...while statement
5. switch case statement
6. for loop

Let us discuss each of the control statement in details -

1. if statement

The if statement is of two types

- i) **Simple if statement :** The if statement in which only one statement is followed by that is statement.

Syntax

```
if(apply some condition)
    statement
```

For example

```
if(a>b)
    System.out.println("a is Biiiiig!");
```

- ii) **Compound if statement :** If there are more than one statement that can be executed when if condition is true. Then it is called compound if statement. All these executable statements are placed in curly brackets.

Syntax

```
if(apply some condition)
{
    statement 1
    statement 2
    .
    .
    .
    statement n
}
```

2. if...else statement

The syntax for if...else statement will be -

```
if(condition)
    statement
else
    statement
```

For example

```
if(a>b)
    System.out.println("a is big")
else
    System.out.println("b :big brother")
```

The if...else statement can be of compound type even. For example

```

if(raining==true)
{
    System.out.println("I won't go out");
    System.out.println("I will watch T.V. Serial");
    System.out.println("Also will enjoy coffee");
}
else
{
    System.out.println("I will go out");
    System.out.println("And will meet my friend");
    System.out.println("we will go for a movie");
    System.out.println("Any how I will enjoy my life");
}

```

if...else if statement

The syntax of if...else if statement is

```

if(is condition true?)
    statement
else if(another condition)
    statement
else if(another condition)
    statement
else
    statement

```

For example

```

if(age==1)
    System.out.println("You are an infant");
else if(age==10)
    System.out.println("You are a kid");
else if(age==20)
    System.out.println("You are grown up now");
else
    System.out.println("You are an adult");

```

Let us now see Java programs using this type of control statement-

Java Program [ifelsedemo.java]

```

/*
This program illustrates the use of
if else statement
*/

```

```
class ifelsedemo
```

```

{
public static void main(String[] args)
{
int x=111,y=120,z=30;
if(x>y)
{
if(x>z)
    System.out.println("The x is greatest");
else
    System.out.println("The z is greatest");
}
else
{
if(y>z)
    System.out.println("The y is greatest");
else
    System.out.println("The z is greatest");
}
}
}
}

```

Output

The y is greatest

Java Program [ifdemo.java]

```

/*This program illustrates the use of
if...else if statement
*/
class ifdemo
{
public static void main(String args[])
{
int basic=20000;
double gross,bonus;
if(basic<10000)
{
bonus=0.75*basic;
gross=basic+bonus;
System.out.println("Your Salary is= "+gross);
}
else if(basic>10000&&basic<=20000)
{
bonus=0.50*basic;
gross=basic+bonus;
System.out.println("Your Salary is= "+gross);
}
}

```

```

else if(basic>20000&&basic<=50000)
{
    bonus=0.25*basic;
    gross=basic+bonus;
    System.out.println("Your Salary is = "+gross);
}
else
{
    bonus=0.0;
    gross=basic+bonus;
    System.out.println("Your Salary is = "+gross);
}
}
}
}

```

Output

Your Salary is = 30000.0

3. while statement

This is another form of while statement which is used to have iteration of the statement for the any number of times. The syntax is

```

while(condition)
{
    statement 1;
    statement 2;
    statement 3;
    ...
    statement n;
}

```

For example

```

int count=1;
while(count<=5)
{
    System.out.println("I am on line number "+count);
    count++;
}

```

Let us see a simple Java program which makes the use of while construct.

Java Program [whiledemo.java]

```
/*
```

This is java program which illustrates
while statement

```
*/
```

```

class whiledemo
{
    public static void main(String args[])
    {
        int count=1,i=0;
        while(count<=5)
        {
            i=i+1;
            System.out.println("The value of i= "+i);
            count++;
        }
    }
}

```

Output

```

The value of i= 1
The value of i= 2
The value of i= 3
The value of i= 4
The value of i= 5

```

4. do... while statement

- This is similar to while statement but the only **difference** between the two is that in case of do...while statement the statements inside the do...while must be executed at least once.
- This means that the statement inside the do...while body gets **executed first** and then the while condition is checked for next execution of the statement, whereas in the while statement first of all the condition given in the while is checked first and then the statements inside the while body get executed when the condition is true.

Syntax

```

do
{
    statement 1;
    statement 2;
    ...
    statement n;
}while(condition);

```

For example

```

int count=1;
do
{
    System.out.println("I am on the first line of do-while");
    System.out.println("I am on the second line of do-while");
}

```

```

System.out.println("I am on the third line of do-while");
System.out.println("I am on the forth line of do-while");
System.out.println("I am on the fifth line of do-while");
count++;
}while(count<=5);

```

Java Program [dowhiledemo.java]

```

/*
This is java program which illustrates
do...while statement
*/

```

```

class dowhiledemo
{
    public static void main(String args[])
    {
        int count=1,i=0;
        do
        {
            i=i+1;
            System.out.println("The value of i= "+i);
            count++;
        }while(count<=5);
    }
}

```

Output

```

The value of i= 1
The value of i= 2
The value of i= 3
The value of i= 4
The value of i= 5

```

5. switch statement

You can compare the switch case statement with a Menu-Card in the hotel. You have to select the menu then only the order will be served to you.

Here is a sample program which makes use of switch case statement -

Java Program [switchcasedemo.java]

```

/*
This is a sample java program for explaining
Use of switch case
*/

```

```

class switchcasedemo
{
    public static void main(String args[])
    throws java.io.IOException
    {
        char choice;
        System.out.println("\tProgram for switch case demo");
        System.out.println("Main Menu");
        System.out.println("1. A");
        System.out.println("2. B");
        System.out.println("3. C");
        System.out.println("4. None of the above");
        System.out.println("Enter your choice");
        choice=(char)System.in.read();
        switch(choice)
        {
            case '1':System.out.println("You have selected A");
            break;
            case '2':System.out.println("You have selected B");
            break;
            case '3':System.out.println("You have selected C");
            break;
            default:System.out.println("None of the above choices made");
        }
    }
}

```

Output

Program for switch case demo

Main Menu

- 1. A
- 2. B
- 3. C
- 4. None of the above

Enter your choice

2

You have selected B

Note that in above program we have written main() in somewhat different manner as -

```

public static void main(String args[])
throws java.io.IOException

```

This is IOException which must be thrown for the statement System.in.read(). Just be patient, we will discuss the concept of Exception shortly! The System.in.read() is required for reading the input from the console[note that it is parallel to scanf statement in C]. Thus using System.in.read() user can enter his choice. Also note that whenever System.in.read() is used it is necessary to write the main() with IOException in order to handle the input/output error.

6. for loop

for is a keyword used to apply loops in the program. Like other control statements for loop can be categorized in simple for loop and compound for loop.

Simple for loop :

```
for (statement 1;statement 2;statement 3)
    execute this statement;
```

Compound for loop :

```
for(statement 1;statement 2; statement 3)
{
    execute this statement;
    execute this statement;
    execute this statement;
    that's all;
}
```

Here

Statement 1 is always for initialization of conditional variables,

Statement 2 is always for terminating condition of the for loop,

Statement 3 is for representing the stepping for the next condition.

For example :

```
for(int i=1;i<=5;i++)
{
    System.out.println("Java is an interesting language");
    System.out.println("Java is a wonderful language");
    System.out.println("And simplicity is its beauty");
}
```

Let us see a simple Java program which makes use of for loop.

Java Program [forloop.java]

```
/*
This program shows the use of for loop
*/
class forloop
{
    public static void main(String args[])
    {
        for(int i=0;i<=5;i++)
            System.out.println("The value of i: "+i);
    }
}
```

Output

The value of i: 0
 The value of i: 1
 The value of i: 2
 The value of i: 3
 The value of i: 4
 The value of i: 5

Ex. 1.11.1 : Write a Java program to find factorial of a given number.**Sol. :****Method 1 : Factorial program without recursion i.e. using control statement**

```
class FactorialProgram
{
    public static void main(String args[])
    {
        int i,fact=1;
        int number=5;//The factorial of 5 is calculated
        i=1;
        while(i<number)
        {
            fact=fact*i;
            i++;
        }
        System.out.println("Factorial of "+number+" is: "+fact);
    }
}
```

Output

Factorial of 5 is: 120

Method 2 : Factorial program using recursion

```
class FactorialProgramRec
{
    static int factorial(int n)
    {
        if (n == 0)
            return 1;
        else
            return(n * factorial(n-1)); //recursive call
    }
    public static void main(String args[])
    {
        int i,fact=1;
        int number=5;//Factorial of this number is calculated
        fact = factorial(number);
```

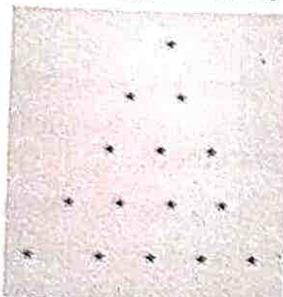
```

        System.out.println("Factorial of "+number+" is: "+fact);
    }
}

```

Output

Factorial of 5 is: 120

Ex. 1.11.2 : Write a Java program to display following pattern.

Sol. :

```

class TriangleDisplay
{
    public static void main(String[] args)
    {
        int i,j,k;
        for(i=1; i<=5; i++)
        {
            for(j=4; j>=i; j--)
            {
                System.out.print(" ");
            }
            for(k=1; k<=(2*i-1); k++)
            {
                System.out.print("*");
            }
            System.out.println("");
        }
    }
}

```

Output

```

*
***
*****
*****
*****
```

Ex. 1.11.3 : Write a Java program to display following number pattern

```

    1
   1 2
  1 2 3
 1 2 3 4
1 2 3 4 5

```

Sol. :

```

class NumberPatternDisplay
{
    public static void main(String[] args)
    {
        int i,j;
        for(i=1; i<=5; i++)
        {
            for(j=1; j<=i; j++)
            {
                System.out.print(j+" ");
            }
            System.out.println("");
        }
    }
}

```

Output

The screenshot shows a Command Prompt window titled "Command Prompt". The user has typed "D:\>javac NumberPatternDisplay.java" and then "D:\>java NumberPatternDisplay". The output displayed is:

```

D:\>javac NumberPatternDisplay.java
D:\>java NumberPatternDisplay
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
D:\>

```

Ex. 1.11.4 : Write a program for following series $1 + 1/2 + 1/2^2 + 1/2^3 + \dots + 1/2^n$

Sol. :

Java Program

```
//////////////////////////////  
//Program for computing  $1 + 1/2 + 1/2^2 + 1/2^3 + \dots + 1/2^n$   
/////////////////////////////  
  
import java.io.*;  
import java.lang.*;  
import java.math.*;  
public class Series1  
{  
    public static void main(String args[]) throws IOException  
    {  
        int n;  
        double val;  
        double sum=1;  
        n=5;  
        System.out.println("\n\t\tProgram for displaying the sum of series");  
        for(int i=1;i<n;i++)  
        {  
            val=1/(Math.pow(2,i));  
            sum=sum+val;  
        }  
        System.out.println("\nSum of the series is "+sum);  
    }  
}
```

Output

Program for displaying the sum of series
Sum of the series is 1.9375

Ex. 1.11.5 : Write a Java application that computes the value of ex by using the following series. $ex = 1+x/1!+x^2/2!+x^3/3!+\dots$

Sol. :

```
//////////////////////////////  
//Program for computing sum of series  
/////////////////////////////  
  
import java.io.*;  
import java.lang.*;  
import java.math.*;  
public class Series2  
{  
    public static int fact(int n)
```

```

{
    int x,y;
    if(n<0)
    {
        System.out.println("The negative parameter in the factorial function");
        return -1;
    }
    if(n==0)
        return 1;
    x=n-1;
    y=fact(x); /*recursive call*/
    return(n*y);
}
public static void main(String args[])throws IOException
{
    int n;
    float val;
    float sum=1;
    int x=2;
    n=3;
    System.out.println("\n\t\tProgram for displaying the sum of series");
    for(int i=1;i<=n;i++)
    {
        val=(float)(Math.pow(x,i))/fact(i);
        sum=sum+val;
    }
    System.out.println("\nSum of the series is "+sum);
}
}

```

Output

Program for displaying the sum of series

Sum of the series is 6.3333335

Ex. 1.11.6 : Write a Java program that reverses the digits of given number.

Sol. :

```

////////// //////////////////////////////////////////////////////////////////
//Program for reversing digits of given number
////////// //////////////////////////////////////////////////////////////////

import java.io.*;
public class Reversenum
{
    public static void main(String[] args)
    {
        int num=1234;

```

```

int rev_num=0;
while(num!=0)
{
    rev_num=(rev_num*10)+(num%10);
    num=num/10;
}
System.out.println(rev_num);
}
}

```

Output

4321

Ex.1.11.7 : Write a Java code using do-while loop that counts down to 1 from 10 printing exactly ten lines of “hello”.

AU : May-19, Marks 6

Sol. :

```

class test
{
    public static void main(String args[])
    {
        int count = 10;
        do
        {
            System.out.println("Hello");
            count--;
        }while(count>=1);
    }
}

```

Output

Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello

Review Question

1. Explain the selection statements in Java using suitable examples.

AU : May-19, Marks 7

1.12 The break and continue Statement

Sometimes when we apply loops we need to come out of the loop on occurrence of particular condition. This can be achieved by break statement. Following program illustrates the use of break in the for loop.

Java Program [breakdemo.java]

```
/*
This program shows the use of break statement
*/
class breakdemo
{
    public static void main(String args[])
    {
        for(int i=1;i<=20;i++)
        {
            if(i%10==0)
            {
                System.out.println("number "+i+" is divisible by 10");
                break;//come out of for loop
            }
            else
                System.out.println("number "+i+" is not divisible by 10");
        }
    }
}
```

Output

The number 1 is not divisible by 10
The number 2 is not divisible by 10
The number 3 is not divisible by 10
The number 4 is not divisible by 10
The number 5 is not divisible by 10
The number 6 is not divisible by 10
The number 7 is not divisible by 10
The number 8 is not divisible by 10
The number 9 is not divisible by 10
The number 10 is divisible by 10

Program explanation : Note that in the above program the numbers after 10 will not be considered at all. This is because when we reach at 10 then if condition becomes true and we encounter break statement. This forces us to come out of for loop. Just look at the output of corresponding program!!!

Similarly we can use the return statement which is useful for returning the control from the current block.

Difference between break and continue

Sr. No.	break	continue
1.	The break is used to terminate the execution of the loop.	The continue statement is not used to terminate the execution
2.	It breaks iteration.	It skips the iteration.
3.	Break is used in loops and in switch.	Continue is used only in loop.
4.	When this break is executed, the control will come out of the loop.	When continue is executed the control will not come out of the loop, in fact it will jump to next iteration of loop.
5.	Example: <pre>public class breakdemo { public static void main(String[] args) { for (int i = 1; i <= 20; i++) { if (i % 2 == 0) { break; // skip the even //numbers } System.out.println(i + " "); } } }</pre> Output 1 3 5 7 9 11 13 15 17	Example: <pre>public class continuedemo { public static void main(String[] args) { for (int i = 1; i <= 20; i++) { if (i % 2 == 0) { continue; // skip the even //numbers } System.out.println(i + " "); } } }</pre> Output 1 3 5 7 9 11 13 15 17

1.13 Arrays

AU : May-14,19, Dec.-19, Marks 16

- Definition : Array is a collection of similar type of elements.
- Using arrays the elements that are having the same data type can be grouped together.
- If we use bunch of variables instead of arrays, then we have to use large number of variables.
- Declaring and handling large number of variables is very inconvenient for the developers.
- There are two types of arrays -
 - One dimensional arrays
 - Two dimensional arrays

1.13.1 One Dimensional Array

- Array is a collection of similar type of elements. Thus grouping of similar kind of elements is possible using arrays.
- Typically arrays are written along with the size of them.
- The syntax of declaring array is -

`data_type array_name[];`

and to allocate the memory -

`array_name=new data_type[size];`

where *array_name* represent name of the array, *new* is a keyword used to allocate the memory for arrays, *data_type* specifies the data type of array elements and *size* represent the size of an array. For example:

`a=new int[10];`

[Note that in C/C++ this declaration is as good as `int a[10]`]

- After this declaration the array *a* will be created as follows

Array a[10]

0	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---

- That means after the above mentioned declaration of array, all the elements of array will get initialized to zero. Note that, always, while declaring the array in Java, we make use of the keyword *new*, and thus we actually make use of dynamic memory allocation.
- Therefore arrays are allocated dynamically in Java. Let us discuss on simple program based on arrays.

Java Program [SampleArray.Java]

```
/*
This is a Java program which makes use of arrays
*/
class SampleArray
{
    public static void main(String args[])
    {
        int a[];
        a=new int[10];
        System.out.println("\tStoring the numbers in array");
        a[0]=1;
        a[1]=2;
        a[2]=3;
        a[3]=4;
        a[4]=5;
        a[5]=6;
        a[6]=7;
        a[7]=8;
        a[8]=9;
        a[9]=10;
        System.out.println("The element at a[5] is: "+a[5]);
    }
}
```

Output

Storing the numbers in array

The element at a[5] is : 6

Program explanation

In above program we have created an array of 10 elements and stored some numbers in that array using the array index. The array that we have created in above program virtually should look like this -

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
1	2	3	4	5	6	7	8	9	10

The `System.out.println` statement is for printing the value present at `a[5]`. We can modify the above program a little bit and i.e instead of writing

```
int a[];
a=new int[10];
```

these two separate statements we can combine them together and write it as -

```
int a[]=new int[10];
```

That means the declaration and initialization is done simultaneously.

Another way of initialization of array is

```
int a[] = {1,2,3,4,5,6,7,8,9,10};
```

That means, as many number of elements that are present in the curly brackets, that will be the size of an array. In other words there are total 10 elements in the array and hence size of array will be 10.

1.13.2 Two Dimensional Arrays

- The two dimensional arrays are the arrays in which elements are stored in rows as well as in columns.
- For example

		Columns		
		0	1	2
Rows	0	10	20	30
	1	40	50	60
		70	80	90

- The two dimensional array can be declared and initialized as follows

Syntax

```
data_type array_name=new data_type[size];
```

For example

```
int a=new int[10];
```

Let us straight away write a Java program that is using two dimensional arrays -

Java Program [Sample2DArray.java]

```
/*
This is a Java program which makes use of 2D arrays
*/
class Sample2DArray
{
    public static void main(String args[])
    {
        int a[][]=new int[3][3];
        int k=0;
        System.out.println("\tStoring the numbers in array");
        for(int i=0;i<3;i++)
        {
            for(int j=0;j<3;j++)
            {
                a[i][j]=k;
                k++;
            }
        }
    }
}
```

```

for(int j=0;j<3;j++)
{
    a[i][j]=k+10;
    k=k+10;
}
System.out.println("You have stored...");
for(int i=0;i<3;i++)
{
    for(int j=0;j<3;j++)
    {
        System.out.print(" "+a[i][j]);
    }
    System.out.println();
}
}
}
}

```

Output

Storing the numbers in array

You have stored...

10 20 30

40 50 60

70 80 90

The typical application of two dimensional arrays is performing matrix operations. Let have some simple Java program in which various matrix operations are performed.

Ex. 1.13.1 : Write a Java program to perform matrix addition, subtraction and multiplication.

Java Program[Matrix.java]

```

/*
This is a Java program which performs matrix operations
*/
class Matrix
{
    public static void main(String args[])
    throws java.io.IOException
    {
        int a[][]={

            {10,20,30},
            {40,50,60},
            {70,80,90}
        };
        int b[][]={

            {1,2,3},

```

```

{4,5,6},
{7,8,9}
};

int c[][]=new int [3][3];
char choice;
System.out.println("\n Main Menu");
System.out.println("1. Addition");
System.out.println("2. Subtraction");
System.out.println("3. Multiplication");
System.out.println("Enter your choice");
choice=(char)System.in.read();
switch(choice)
{
case '1':for(int i=0;i<3;i++)
{
    for(int j=0;j<3;j++)
    {
        c[i][j]=a[i][j]+b[i][j];
    }
}
System.out.println("The Addition is...");
for(int i=0;i<3;i++)
{
    for(int j=0;j<3;j++)
    {
        System.out.print(" "+c[i][j]);
    }
    System.out.println();
}
break;
case '2':for(int i=0;i<3;i++)
{
    for(int j=0;j<3;j++)
    {
        c[i][j]=a[i][j]-b[i][j];
    }
}
System.out.println("The Subtraction is...");
for(int i=0;i<3;i++)
{
    for(int j=0;j<3;j++)
    {
        System.out.print(" "+c[i][j]);
    }
    System.out.println();
}
}

```

```

        break;
case '3':for(int i=0;i<3;i++)
{
    for(int j=0;j<3;j++)
    {
        c[i][j]=0;
        for(int k=0;k<3;k++)
        {
            c[i][j]=c[i][j]+a[i][k]*b[k][j];
        }
    }
}
System.out.println("The Multiplication is...");
for(int i=0;i<3;i++)
{
    for(int j=0;j<3;j++)
    {
        System.out.print(" "+c[i][j]);
    }
    System.out.println();
}
break;
}
}
}

```

Output (Run1)

Main Menu
 1. Addition
 2. Subtraction
 3. Multiplication
 Enter your choice

1

The Addition is...

11 22 33

44 55 66

77 88 99

Output (Run2)

Main Menu
 1. Addition
 2. Subtraction
 3. Multiplication
 Enter your choice

2

The Subtraction is...

9 18 27
36 45 54
63 72 81

Output (Run3)

Main Menu
1. Addition
2. Subtraction
3. Multiplication
Enter your choice
3
The Multiplication is...
300 360 420
660 810 960
1020 1260 1500

Ex. 1.13.2 : Write a Java Program to implement Linear Search.

Sol.

```
public class LinearSearch
{
    public static void main(S
        int[] a={1,5,-2,8,7,11,40,3
        System.out.println("The e
        System.out.println("The e
    }
    public static int Find(int[
    {
        int i;
        for(i=0;i<a.length;i++)
        {
            if(a[i]==key)
                return i+1;
        }
        return -1;
    }
}
```

Output

The element is at location 5
The element is at location 6

arraycopy Command

Using array copy we can copy entire an to another array. The arraycopy command is applicable to one dimensional array. The syntax of arraycopy command is -

```
public static void arraycopy(Object src,
                           int srcPos,
                           Object dest,
                           int destPos,
                           int length) (snuff) bugfix
```

But it can be extended to copy two dimensional array as well. Following example illustrates this idea.

Ex. 1.13.3 : Write a program for transposition of matrix using arraycopy command.

Sol. :

Java Program[ArrayTranspose.java]

```
public class ArrayTranspose
{
    public static void main(String args[])
    {
        int A[][]={{1,2,3},{4,5,6},{7,8,9}};
        int B[][]=new int[3][3];
        int i,j,From[],To[];
        From =new int[9];
        To =new int[9];
        int k=0;
        System.out.println("The original matrix is...");
        for(i=0;i<3;i++)
        {
            for(j=0;j<3;j++)
                System.out.print(A[i][j]+" ");
            System.out.println();
        }
        for(i=0,k=0;i<3;i++)
        {
            for(j=0;j<3;j++)
                From[k++]=A[i][j];//making it 1-D
            for(j=0,j=0,k=0;i<9;i++)
            {
                System.arraycopy(From,j,To,i,1);//copying it to another 1-D array
                j=j+3;
                if(j>=9)//first row completion
                    j=++k;
            }
            System.out.println("The Transposed Matrix is...");
            for(i=0,k=0;i<3;i++)
            {
                for(j=0;j<3;j++)
                    To[i][j]=From[j];
            }
        }
    }
}
```

```

{
    B[i][j]=To[k];
    k=k+1;
    System.out.print(B[i][j]+ " ");
}
System.out.println();
}
}
}

```

Output

The original matrix is...

1 2 3
4 5 6
7 8 9

The Transposed Matrix is...

1 4 7
2 5 8
3 6 9

Ex. 1.13.4 : Write a program to find the transpose of a given matrix.

Sol. :

```

import java.util.Scanner;
class Transpose
{
    public static void main(String args[])
    {
        int m, n, i, j;
        Scanner in = new Scanner(System.in);
        System.out.println("Enter the number of rows and columns of matrix");
        m = in.nextInt();
        n = in.nextInt();

        int A[][] = new int[m][n];//original matrix
        int B[][] = new int[n][m];//transpose matrix
        System.out.println("Enter the elements of matrix");
        for (i = 0; i < m; i++)
            for (j = 0; j < n; j++)
                A[i][j] = in.nextInt(); //enter matrix elements through keyboard
        for (i = 0; i < m; i++)
        {

```

```

for ( j = 0 ; j < n ; j++ )
    B[j][i] = A[i][j];
}
System.out.println("Transpose of matrix is ");

for (i=0;i<n;i++)
{
    for (j=0;j<m;j++)
        System.out.print(" " + B[i][j]);
    System.out.print("\n");
}
}
}

```

Output

Enter the number of rows and columns of matrix

3 4

Enter the elements of matrix

1 2 3 4

5 6 7 8

9 10 11 12

Transpose of matrix is

1	5	9
2	6	10
3	7	11
4	8	12

Ex. 1.13.5 : Write a program to perform the following functions on a given matrix

- i) Find the row and column sum ii) Interchange the rows and columns.

AU : May-14, Marks 16

Sol. :

```

i)
class RowColSum
{
    public static void main(String args[])throws java.io.IOException
    {
        int a[][]={

{10,20,30},
{40,50,60},
{70,80,90}

};

        int rsum[]=new int[3];
        int csum[]=new int[3];
        int i,j;
    }
}

```

```

/* Sum of Rows */
for(i=0;i<3;i++)
{
    rsum[i]=0;
    for(j=0;j<3;j++)
        rsum[i]=rsum[i]+a[i][j];
}
/* Sum of Column */
for(i=0;i<3;i++)
{
    csum[i]=0;
    for(j=0;j<3;j++)
        csum[i]=csum[i]+a[j][i];
}
System.out.println("The sum of rows and columns of the matrix is :");
for(i=0;i<3;i++)
{
    for(j=0;j<3;j++)
        System.out.print(" "+a[i][j]);/*displaying each row*/
    System.out.print(" = "+rsum[i]);/*displaying row sum*/
    System.out.println("");
}
System.out.println("-----");
for(j=0;j<3;j++)
{
    System.out.print(" "+csum[j]);/*displaying col sum*/
}
}
}

```

Output

The sum of rows and columns of the matrix is :

10 20 30 = 60

40 50 60 = 150

70 80 90 = 240

120 180

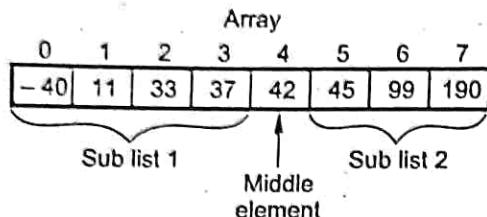
ii) Refer example 1.13.4.

Ex. 1.13.6 : Explain binary search method with necessary illustration and write a Java program to implement it.

Sol. : The necessity of this method is that all the elements should be sorted. So let us take an array of sorted elements.

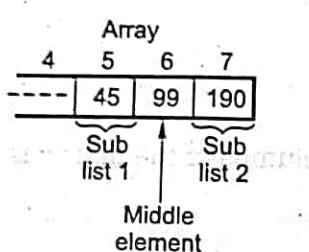
array

0	1	2	3	4	5	6	7
-40	11	33	37	42	45	99	100



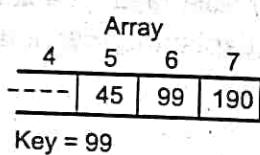
Step 1 : Now, the key element which is to be searched is = 99 key = 99.

Step 2 : Find the middle element of the array. Compare it with the key



if middle ? key
 i.e. if 42 ? 99
 if $42 < 99$ search the sublist 2

Now handle only sublist 2. Again divide it, find mid of sublist 2



if middle ? key
 i.e. if $99 = 99$

So match is found at 7th position of array i.e. at array [6]

Thus by binary search method we can find the element 99 present in the given list at array [6]th location.

Java Program[BinSearch.java]

```
public class BinSearch {
    public static void main(String args[]) {
        int[] a={10,20,30,40,50,60};
        int key=40;
        Find(a,0,5,key);
    }
    public static void Find(int[] a,int low,int high,int key)
    {
        int mid;
        if(low>high)
        {
            System.out.println("Error!!The element is not present in the list");
            return;
        }
    }
}
```

```

mid=(low+high)/2;
if(key==a[mid])
    System.out.println("\n The element is present at location "+(mid+1));
else if(key<a[mid])
    Find(a,low,mid-1,key);
else if(key>a[mid])
    Find(a,mid+1,high,key);
}
}

```

Output

F:\test>javac BinSearch.java

F:\test>java BinSearch

The element is present at location 4

Ex. 1.13.7 : Develop a Java program to find a smallest number in the given array by creating a one dimensional array and two dimensional array using new operator.

AU : Dec.- 19, Marks 15

Sol. :

```

import java.util.*;
public class SmallestElement
{
    public static void main(String[] args)
    {
        int rows;
        int columns;
        Scanner scanner = new Scanner (System.in);
        System.out.println("\t\t **** TWO DIMENSIONAL ARRAY ****");
        System.out.println("Enter number of rows: ");
        rows = scanner.nextInt();
        System.out.println("Enter number of columns: ");
        columns = scanner.nextInt();
        //creating two dimensional array using new operator
        int[][] matrix = new int [rows][columns];
        //storing the elements in the two dimensional array
        System.out.println("Enter matrix numbers: ");
        for (int i = 0; i < rows; i++)
        {
            for (int j = 0; j < columns; j++)
            {
                matrix[i][j] = scanner.nextInt();
            }
        }
    }
}

```

```

// Displaying entered matrix
System.out.println("Matrix as entered");
for (int i = 0; i < matrix.length; i++)
{
    System.out.println();
    for (int j = 0; j < matrix[i].length; j++)
    {
        System.out.print(matrix[i][j] + " ");
    }
}
System.out.println();
//finding smallest element from 2D array
findMin(matrix);
int size;
System.out.println("\t\t***** ONE DIMENSIONAL ARRAY *****");
System.out.println("Enter total number of elements: ");
size = scanner.nextInt();
//creating one dimensional array using new operator
int[] a = new int[size];
//storing the elements in one dimensional array
System.out.println("Enter the elements: ");
for(int i = 0;i< size; i++)
{
    a[i] = scanner.nextInt();
}
//displaying one dimensional array
System.out.println("The one dimensional array which you have entered is");
for(int i = 0;i< size; i++)
{
    System.out.println(a[i]+ " ");
}
System.out.println();
//finding smallest element from array
findSmall(a);
}

private static void findMin(int[][] matrix)
{
    int minNum = matrix[0][0];
    for (int i = 0; i < matrix.length; i++)
    {
        for (int j = 0; j < matrix[i].length; j++)
        {
            if(minNum > matrix[i][j])
            {
                minNum = matrix[i][j];
            }
        }
    }
}

```

```

        }
    }

    System.out.println("Smallest number: " + minNum);
}

private static void findSmall(int[] a)
{
    int smallNum = a[0];
    for(int i = 0; i < a.length; i++)
    {
        if(smallNum > a[i])
        {
            smallNum = a[i];
        }
    }
    System.out.println("Smallest number: " + smallNum);
}
}

```

Output******* TWO DIMENSIONAL ARRAY *******

Enter number of rows:

3

Enter number of columns:

3

Enter matrix numbers:

5 2 4

3 1 6

9 8 7

Matrix as entered

5 2 4

3 1 6

9 8 7

Smallest number: 1

******* ONE DIMENSIONAL ARRAY *******

Enter total number of elements:

5

Enter the elements:

30 10 20 50 40

The one dimensional array which you have entered is

30

10

20

50

40

Smallest number : 10

Ex. 1.13.8 : Define Java classes of your own without using any library classes to represent linked lists of integers. Provide it with methods that can be used to reverse a list and to append two lists.

AU : May-19, Marks 15

Sol. :

```

public class SinglyLinkedList{
    Node head;

    static class Node {
        private int data;
        private Node next;

        Node(int data) {
            this.data = data;
            this.next = null;
        }
    }

    public void insert(Node node) {
        if (head == null) {
            head = node;
        } else {
            Node temp = head;
            while (temp.next != null)
                temp = temp.next;
            temp.next = node;
        }
    }

    public void display(Node head) {
        Node temp = head;
        while (temp != null) {
            System.out.format("%d ", temp.data);
            temp = temp.next;
        }
        System.out.println();
    }

    // Reverse linkedlist using this function
    public static Node reverse(Node currentNode)
    {

```

```

    // For first node, previousNode will be null
    Node previousNode=null;
    Node nextNode;
    while(currentNode!=null)
    {
        nextNode=currentNode.next;
        // reversing the link
        currentNode.next=previousNode;
        // moving ahead currentNode and previousNode by 1 node
        previousNode=currentNode;
        currentNode=nextNode;
    }
    return previousNode;
}

public static Node Append(Node head1,Node head2) {
    Node previousNode =null;
    Node temp = head1;
    while (temp != null) {
        previousNode = temp;
        temp = temp.next;//reaching to last node of first list
    }
    previousNode.next = head2;//appending second list
    return head1;
}

```

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

```
    SinglyLinkedList list = new SinglyLinkedList();
```

```
    // Creating a linked list
```

```
    Node head=new Node(10);
    list.insert(head);
    list.insert(new Node(20));
    list.insert(new Node(30));
    list.insert(new Node(40));
    list.insert(new Node(50));
```

```
    list.display(head);
```

```
    //Reversing LinkedList
```

```
    Node reverseHead=reverse(head);
```

```

        System.out.println("After reversing");
        list.display(reverseHead);
        //Creating First LinkedList
        Node head1=new Node(1);
        list.insert(head1);
        list.insert(new Node(2));
        list.insert(new Node(3));
        list.insert(new Node(4));
        list.insert(new Node(5));
        System.out.println("The first linked list is");
        list.display(head1);
        //Creating Second LinkedList
        Node head2=new Node(6);
        list.insert(head2);
        list.insert(new Node(7));
        list.insert(new Node(8));
        list.insert(new Node(9));
        list.insert(new Node(10));
        System.out.println("The second linked list is");
        list.display(head2);
        System.out.println("The concatenated linked list is");
        Node head3 = Append(head1,head2);
        list.display(head3);
    }
}

```

1.14 Defining Classes in Java

AU : IT : May-12, CSE : Dec.-10, 14, Marks 5

Each class is a collection of data and the functions that manipulate the data. The data components of the class are called data fields and the function components of the class are called member functions or methods.

- Thus the state of an object is represented using data fields (data members) and behaviour of an object is represented by set of methods (member functions). The class template can be represented by following Fig. 1.14.1.

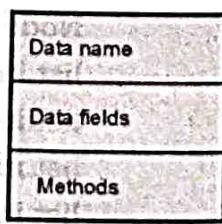


Fig. 1.14.1 Class template

- The Java Class can written as follows

```
class Customer {
    int ID;
    int Age;
    String Name;

    Customer() {
    }
    Customer(int ID) {
    }
}
```

```
double withdraw_money() {
    ...
}
```

- Encapsulation means binding of data and method together in a single entity called class. Thus a class is used to achieve an **encapsulation** property.
- This class is not having the main function. The class that contains main function is called **main class**.
- Following is a simple Java program that illustrates the use of class

```
////////////////////////////////////////////////////////////////////////
//Program for demonstrating the concept of class
////////////////////////////////////////////////////////////////////////
import java.io.*;
import java.lang.*;
import java.math.*;
public class MyClass
{
    String name;
    int roll;
    double marks;
    public void display(String n,int r,double m)
    {
        System.out.println();
        System.out.println("Name: "+n);
        System.out.println("Roll number: "+r);
        System.out.println("Marks: "+m);
    }
}
```

```

public static void main(String args[])
{
    int a,b;
    MyClass obj1=new MyClass();
    MyClass obj2=new MyClass();
    MyClass obj3=new MyClass();
    obj1.display("Amar",10,76.65);
    obj2.display("Akbar",20,87.33);
    obj3.display("Anthony",30,96.07);
}
}

```

Output

Name: Amar
 Roll number: 10
 Marks: 76.65

Name: Akbar
 Roll number: 20
 Marks: 87.33

Name: Anthony
 Roll number: 30
 Marks: 96.07

Program Explanation

- In above program we have declared one simple class. This class displays small database of the students. The data members of this class are name, roll and marks. There is one member function named **display** which is for displaying the data members. In the **main** function we have created three objects **obj1**, **obj2** and **obj3**. These objects represent the real world entities. These entities are actually the instances of the class. The class representation of the above program can be as follows -

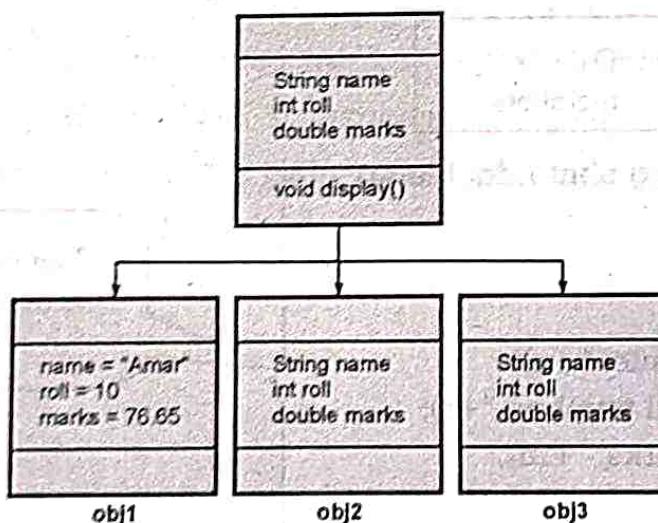


Fig. 1.14.2

Review Question

1. What is class? How do you define a class in Java ?

1.15 Constructor

- **Definition :** The constructor is a specialized method for **initializing objects**.
- Name of the constructor is same as that of its class name. In other words, the name of the constructor and class name is same.

```
class Test
{
    Test()
    {
        //body of constructor
    }
}
```

- Whenever an object of its associated class is created, the constructor is invoked automatically.
- The constructor is called constructor because it creates values for data fields of class.
- Example -

Java Program[Rectangle1.java]

```
public class Rectangle1 {
    int height;           Data fields
    int width;
    Rectangle1()          Simple constructor
    {
        System.out.println(" Simple constructor: values are initialised...");

        height=10;
        width=20;
    }
    Rectangle1(int h,int w) Parameterised constructor
    {
        System.out.println(" Parameterised constructor: values are initialised...");

        height=h;
        width=w;
    }
    void area() Method defined
    {
        System.out.println("Now, The function is called...");

        int result=height*width;

        System.out.println("The area is "+result);
    }
}
```

```

class Constr_Demo {
    public static void main(String args[])
    {
        Rectangle1 obj=new Rectangle1();
        obj.area(); //call the to method
        Rectangle1 obj1=new Rectangle1(11,20);
        obj1.area(); //call the to method
    }
}

```

Object created and simple constructor is invoked

Object created and parameterised constructor is invoked

Output

F:\test>javac Rectangle1.java
F:\test>java Constr_Demo

Note the method of running the program

Simple constructor: values are initialised...

Now, The function is called...

The area is 200

Parameterised constructor: values are initialised...

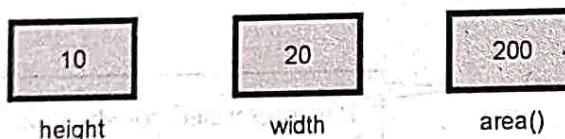
Now, The function is called...

The area is 220

F:\test>

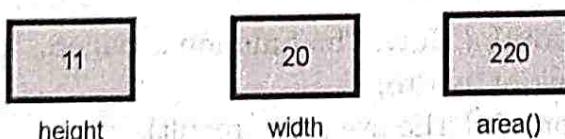
Program Explanation

In above program, there are two classes - **Rectangle1** and **Constr_Demo**. In the **Rectangle1** class, data fields, method and constructor is defined. In the **Constr_Demo** class the objects are created. When an object **obj** gets created, then the simple constructor **Rectangle1()** gets invoked and the data fields height and width gets initialized



Hence the **area** function will compute the area= $10*20=200$.

When an object **obj1** gets created, then the parameterised constructor **Rectangle1(11, 20)** gets invoked and the data fields height and width gets initialised.



Hence the **area** function will compute the area = $11 * 20 = 220$.

The class **Constr_Demo** is called **main class** because it consists of **main** function.

1.15.1 Properties of Constructor

1. Name of constructor must be the same as the name of the class for which it is being used.
2. The constructor must be declared in the **public** mode.
3. The constructor gets invoked automatically when an object gets created.
4. The constructor should not have any return type. Even a **void** data type should not be written for the constructor.
5. The constructor cannot be used as a member of union or structure.
6. The constructors can have default arguments.
7. The constructor **cannot be inherited**. But the derived class can invoke the constructor of base class.
8. Constructor can make use of **new** or **delete** operators for allocating or releasing memory respectively.
9. Constructor can not be **virtual**.
10. Multiple constructors can be used by the same class.
11. When we declare the constructor explicitly then we must declare the object of that class.

1.16 Methods

AU : Dec.-14, 18, Marks 6

Methods are nothing but the functions defined by the particular class.

class classname

{

declaration of member variables;

definition of method

{

...

...

}

}

There are four parts of the method -

- Return type of the method

- name of the method
- parameter passed to the method
- body of the method

For example

Return type Method's name

```

void get_data(int a,int b)
{
    int c=a+b;
}

```

Parameters passed to the method

Body of method

1.16.1 Parameter Passing

- Parameter can be passed to the function by two ways -
 1. Call by value
 2. Call by reference
- In the **call by value** method the value of the actual argument is assigned to the formal parameter. If any change is made in the formal parameter in the subroutine definition then that change does not reflect the actual parameters.
- Following Java program shows the use of parameter passing by value -

Java Program

```

//Program implementing the parameter passing by value
public class ParameterByVal
{
    void Fun(int a,int b)
    {
        a=a+5;
        b=b+5;
    }
    public static void main(String args[])
    {
        ParameterByVal obj1=new ParameterByVal();
        int a,b;
        a=10;b=20;
        System.out.println("The values of a and b before function call");
    }
}
```

```

    System.out.println("a= "+a);
    System.out.println("b= "+b);
    obj1.Fun(a,b);
    System.out.println("The values of a and b after function call");
    System.out.println("a= "+a);
    System.out.println("b= "+b);
}

}

```

Output

The values of a and b before function call

a = 10
b = 20

The values of a and b after function call

a = 10
b = 20

Program Explanation

- The above Java program makes use of the parameter passing by value. By this approach of parameter passing we are passing the actual values of variables a and b.
- In the function Fun we are changing the values of a and b by adding 5 to them. But these incremented values will remain in the function definition body only. After returning from this function these values will not get preserved. Hence we get the same values of a and b before and after the function call.
- On the contrary the parameter passing by reference allows to change the values after the function call. But use of variables as a parameter does not allow to pass the parameter by reference.
- For passing the parameter by reference we pass the object of the class as a parameter.
- Creating a variable of class type means creating reference to the object. If any changes are made in the object made inside the method then those changes get preserved and we can get the changed values after the function call.

Review Question

1. What is method ? How method is defined ? Give example. ✓

AU : Dec.-14, 18, Marks 6

1.17 Access Specifiers

AU : IT : Dec.-11, CSE : Dec.-12, Marks 8

Access modifiers control access to data fields, methods, and classes. There are three modifiers used in Java -

- public
- private
- default modifier

public allows classes, methods and data fields accessible from any class

private allows classes, methods and data fields accessible only from within the own class,

If public or private is not used then by default the classes, methods, data fields are assessable by any class in the same package. This is called **package-private** or **package-access**. A package is essentially grouping of classes.

For example :

```
package Test;
public class class1
{
    public int a;
    int b;
    private int c;
    public void fun1() {
    }
    void fun2() {
    }
    private void fun3() {
    }
}
public class class2
{
    void My_method() {
        class1 obj=new class1();
        obj.a;//allowed
        obj.b;//allowed
        obj.c;//error:cannot access
        obj.fun1();//allowed
        obj.fun2();//allowed
        obj.fun3();//error:cannot access
    }
}
```

```
package another_Test
public class class3
{
    void My_method() {
        class1 obj=new class1();
        obj.a;//allowed
        obj.b;// error:cannot access
        obj.c;//error:cannot access
        obj.fun1();//allowed
        obj.fun2();//error:cannot access
        obj.fun3();//error:cannot access
    }
}
```

In above example,

- We have created two packages are created namely - **Test** and **another_Test**.
- Inside the package **Test** there are two classes defined - **class1** and **class2**
- Inside the package **another_Test** there is only one class defined and i.e. **class3**.
- There are three data fields - **a**, **b** and **c**. The data field **a** is declared as **public**, **b** is defined as **default** and **c** is defined as **private**.

- The variable **a** and method **fun1()** both are accessible from the classes **class2** and **class3**(even if it is in another package). This is because they are declared as **public**.
- The variable **b** and method **fun2()** both are accessible from **class2** because **class2** lies in the same package. But they are not accessible from **class3** because **class3** is defined in another package.
- The variable **c** and method **fun3()** both are not accessible from any of the class, because they are declared as **private**.

Protected mode is another access specifier which is used in inheritance. The **protected** mode allows accessing the members to all the classes and subclasses in the same package as well as to the subclasses in other package. But the non subclasses in other package can not access the protected members.

The effect of access specifiers for class, subclass or package is enlisted below -

Specifier	class	subclass	package
private	Yes	-	-
protected	Yes	Yes	Yes
public	Yes	Yes	Yes

For example, if some variable is declared as protected, then the class itself can access it, its subclass can access it, and any class in the same package can also access it. Similarly if the variable is declared as private then that variable is accessible by that class only and its subclass can not access it.

Review Questions

- Write short note on access specifiers in Java.
- What are access specifiers ? Discuss them in context of Java.

AU : IT : Dec.-10, Marks 6

AU : IT : Dec.-11, Marks 8, CSE : Dec.-12

1.18 Static Members

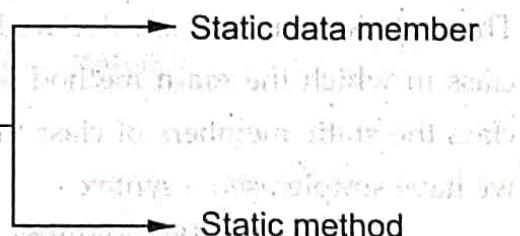
AU : CSE : Dec.-10, IT : May-11, 12, 13, Marks 8

The static members can be **static data member** or **static method**.

The static members are those members which can be accessed

without using object. Following

program illustrates the use of static members.



Java Program[StaticProg.java]

```
*****
Program to introduce the use of the static method and
static variables
*****/ class StaticProg
{
    static int a=10;
    static void fun(int b)
    {
        System.out.println("b= "+b);
        System.out.println("a= "+a);
    }
}
class AnotherClass
{
    public static void main(String[] args)
    {
        System.out.println("a= "+StaticProg.a);
        StaticProg.fun(20);
    }
}
```

Output

a= 10
b= 20
a= 10

Program Explanation

In above program, we have declared one static variable **a** and a static member function **fun()**. These static members are declared in one class **StaticProg**. Then we have written one more class in which the **main** method is defined. This class is named as **AnotherClass**. From this class the static members of class **StaticProg** are accessed *without using any object*. Note that we have simple used a syntax

ClassName.staticMember

Hence by using `StaticProg.a` and `StaticProg.fun` we can access static members

Points to remember about static members

- In Java the `main` is always static method.
- The static methods must access only static data.
- The static method can call only the static method and can not call a non static method.
- The static method can not refer to `this` pointer.
- The static method can not refer to `super` method.

Review Questions

1. Give the syntax for the static method and its initialization. ✓

AU : CSE : Dec.-10, Marks 4

2. Discuss the working and meaning of the "static" modifier with suitable example.

AU : IT : May-11, Marks 8

3. Explain about static variables and static methods used in Java, with examples.

AU : IT : May-12, Marks 8

4. Why do we need static members and how to access them ? Explain it with clear example.

AU : IT : May-13, Marks 8

1.19 JavaDoc Comments

Javadoc is a convenient, standard way to document your Java code. Javadoc is actually a special format of comments. There are some utilities that read the comments, and then generate HTML document based on the comments. HTML files give us the convenience of hyperlinks from one document to another.

There are two types of Javadoc comments -

- Class level comments
- Member level comments

The class level comments describe the purpose of classes and member level comments describe the purpose of members.

The Javadoc comments start with `/**` and end with `*/` For example

`/** This is a Javadoc comment statement*/`

1.19.1 Class Level Comments

The class level comments provide the description and purpose of the classes. For example –

```
/**  
 * @author XYZ  
 * The Employee class contains salary of each employee the organisation.  
 */
```

```
public class Employee
{
    //Employee class code
}
```

1.19.2 Member Level Comments

The member level comments describe the data members, methods and constructors used in particular class. In this type of comments special tags are used to describe the things. The most commonly used tags are -

Tags	Description
@author	This can be used in the class level comment. It describes the name of the author who is writing the document/
@param	This tag describes the name of the method used in the class
@return	This tag describes the return tag of the method.
@throws	This tag describes the exertion that can be thrown by the method.
@exception	This tag describes the exception

The example of member level comments for the Employee class is as shown below -

```
/*
 * @author XYZ
 * The Employee class contains salary of each employee the organisation
 */
public class Employee
{
    /**
     * Employee information for knowing the salary
     */
    private int amtSalary;
    /**
     * The constructor defined to initialise the salary amount
     */
    public Employee()
    {
        this.salary=0;
    }
    /**
     * This method returns the salary amount of particular employee
     * @return int
     */
    public int getAmtSalary()
    {
        return salary;
    }
}
```

```


    * @param int No_of_Days_Worked is for total number of working days
    * @param int Payscale is for payment scale as per the designation of the employee
    */
    public void setAmtSalary(int No_of_Days_Worked, int Payscale)
    {
        this.salary=No_of_Days_Worked* Payscale;
    }
}


```

Along with the above mentioned tags some HTML tags can also be included in Javadoc comments. For example we can use <table> tags in Javadoc. Also we can include special tags like < in the Javadoc. Some words like true, false and null can also be included in Javadoc.

1.20 Two Marks Questions with Answers

Q.1 Mention some of the separators used in Java programming .

AU : IT : Dec.-12

Ans. : Some of the separators that are used in Java are as given below -

Name	Description
.	The period or dot is used to select a field or method from an object. It is also used to separate out the package name from sub-packages or class names.
:	The colon is used in loops after the loop label.
,	The comma is used to separate out variables or to separate out the parameters.
;	The semicolon is used to terminate the statement in Java.
()	The round opening and closing parenthesis are used to enclose the parameters of the method definition or call. It adjusts the precedence in arithmetic expression. For type casting purpose also these parenthesis are used.
{ }	For enclosing the function or the class block or for array initialization these parentheses are used.
[]	These parenthesis are used in array declaration.

Q.2 How dynamic initialization of variables is achieved in java ?

AU : IT : Dec.-12

Ans. : The variables in Java are allowed to get initialized at run time. Such type of initialization is called dynamic initialization.

For example

```

double a=5.0;
double b=20.0
double c=Math.sqrt((a+b));

```

Here the expression is evaluated at run time and its square root value is calculated and then the variable c is initialized dynamically.

AU : CSE : Dec.-13

Q.3 What is the output of the main method in the given code ?

```
public static void main(String[] args)
{
    screen.write(sum());
}
static int sum()
{
    int A=12;
    int B=13;
    return =A+B;
}
```

Ans. : It will generate error at the statement `return =A+B` as "Illegal start of expression".

AU : Dec.-13

Q.4 What are the features of Java ?

Ans. :

- 1) Java is simple to implement.
- 2) Java is platform independent.
- 3) It is an object oriented programming language.
- 4) It is a robust programming language.
- 5) Java is designed for distributed system.

AU : CSE : Dec.-10, 18

Q.5 Define objects and classes in Java.

Ans. : An object is an instance of a class. The objects represent the real world entity. The objects are used to provide a practical basis for the real world. Each class is a collection of data and the functions that manipulate the data. The data components of class are called data fields and the function components of the class are called member functions or methods.

Q.6 Why are classes important in OO technology ?

Ans. : Classes bind together the relative data and methods in a cohesive unit. Due to this arrangement, only certain methods are allowed to access the corresponding data. Secondly, if any modification is needed, then the class can be viewed as one module and the changes made in one class does not spoil rest of the code. Moreover, finding error from such a source code becomes simple.

Hence use of class is very important thing in OO technology.

Q.7 What is the difference between object and class ?**Ans. :**

Following are some differences between the class and the object -

Sr. No.	Class	Object
1.	For a single class there can be any number of objects. For example - If we define the class as River then Ganga, Yamuna, Narmada can be the objects of the class River.	There are many objects that can be created from one class. These objects make use of the methods and attributes defined by the belonging class.
2.	The scope of the class is persistent throughout the program.	The objects can be created and destroyed as per the requirements.
3.	The class cannot be initialized with some property values.	We can assign some property values to the objects.
4.	A class has unique name.	Various objects having different names can be created for the same class.

Q.8 What is the difference between structure and class ?**Ans. :**

Sr. No.	Structure	Class
1.	By default the members of structure are public.	By default the members of class are private.
2.	The structure can not be inherited.	The class can be inherited.
3.	The structures do not require constructors.	The classes require constructors for initializing the objects.
4.	A structure contains only data members.	A class contains the data as well as the function members.

Q.9 What is the difference between static and non static variables ?**AU : IT : Dec.-10**

Ans. : A static variable is shared among all instances of class, whereas a non static variable (also called as instance variable) is specific to a single instance of that class.

Q.10 Define encapsulation.**AU : IT : Dec.-11**

Ans. : Encapsulation means binding of data and method together in a single entity called class.

Q.11 What is an abstract class ?**AU : IT : Dec.-11,12, May-12**

Ans. : When apply inheritance, the class becomes more and more specific as we move down. Sometimes a situation may occur that the superclass becomes more general and less specific. Such a class lists out only common features of other classes. Such a super class is called as **abstract class**.

Q.12 Define class. Give example.

AU : Comp : Dec.-11, IT : May-12

Ans. : Each class is a collection of data and the functions that manipulate the data. The data components of class are called data fields and the function components of the class are called member functions or methods.

For example

```
class Customer
{
    int ID;
    String Name; //Data Field
    Customer() //Constructor
    {
    }
    double withdraw_money() //method
    {
    ...
}
```

Q.13 What are the benefits of encapsulation ? Should abstractions be user centric or developer centric ?

Ans. : Due to encapsulation the corresponding data and the methods get bound together by means of class. The data inside the class is accessible by the function in the same class. It is normally not accessible from outside component. Thus the unwanted access to the data can be protected.

The abstraction should be user centric. While developing the system using the OO principles it is important to focus the user and not the developer.

Q.14 How would you declare an object of type animal named lion that takes a weight of 500 and length of 45 as parameters ?

Ans. :

```
public class Animal
{
    int weight,length;
    Animal(int wt,int len)
    {
        weight=wt;
        length=len;
    }
    void Show()
    {
```

```

    System.out.println("\n The weight of animal is: "+weight);
    System.out.println("\n The length of animal is: "+length);
}
}
class AnimalMain
{
    public static void main(String args[])
    {
        Animal Lion=new Animal(500,45);
        Lion.Show();
    }
}

```

Output

The weight of animal is: 500

The length of animal is: 45

Q.15 What is meant by private access specifier ?

Ans.: The private access specifier allows classes, methods and data fields accessible only from within the own class. The functions outside the class cannot access the data members or the member functions.

Q.16 What do you mean by instance variable ?

AU : CSE : May-12

Ans.: An Object is an instance of a class. The variables that the object contains are called instance variables. For example consider a class Student

```

class Student
{
    int RollNo;
    char name[10];
}
Student S; //Creation of object of type Student class.
S= new Student();

```

If we create an object of the class Student say S, then using this object the variables RollNo and name can be accessed. These variables that belong to object S are then called as instance variables. Thus the Instance variables are any variables, without "static" field modifier, that are defined within the class body.

Q.17 Define constructor.

AU : CSE : Dec.-12

Ans. : Constructor is a specialized method used for initializing the objects. The name of this method and the name of the class must be the same. The constructor is invoked whenever an object of its associated class is created.

Q.18 What is Static in Java ?**AU : IT : May-13**

Ans. : In java, static is a member of a class that is not associated with an instance of a class. If there is a need for a variable to be common to all the objects of a single java class, then the static keyword should be used in the variable declaration.

Q.19 What is the difference between a constructor and a method ?**AU : IT : May-13****Ans. :**

Constructor	Method
Name of the constructor must be the same as the name of the class.	There is no such restriction on the name of the method.
Constructor does not have any return type.	Method has return type. If the method does not return anything then it must have void data type.
Constructors are chained, in the sense that they are called in some specific order.	Methods are not chained. That means - invoking of methods depend upon the logic of the program.

Q.20 What are key characteristics of objects ?**AU : CSE : May-13****Ans. :**

1. Object is an instance of a class.
2. Objects are runtime entities.
3. Using object of a class the member variable and member functions of a class can be accessed.

Q.21 What is meant by parameter passing constructors ? Give example.**AU : CSE : Dec.-13**

Ans. : The constructor methods to which the parameter are passed are called parameter passing constructors

Example -

```
Rectangle(int h,int w)
{
    height=h;
    weight=w;
}
```

Q.22 Enumerate two situations in which static methods are used.**AU : May-14****Ans. :**

- 1) The situation in which object of belonging class is not created and we want to use the method of that class, then the method must be static.

- 2) For calling another static method, one such static method is used.
 3) For accessing static data the static method must be used.

Q.23 List any four JavaDoc comments.

AU : CSE : Dec.-11

Ans. :

Tags	Description
@author	This can be used in the class level comment. It describes the name of the author who is writing the document/
@param	This tag describes the name of the method used in the class
@return	This tag describes the return tag of the method.
@throws	This tag describes the exertion that can be thrown by the method.
@exception	This tag describes the exception

Q.24 What is the need for javadoc multiline comments ?

Ans. : Javadoc multiline comments can be used to document all java source code. Comments follow a standard format consisting of a description followed by block tags. The first sentence of the description should be clear and concise as it is used in the summary of the API item.

Q.25 Define access specifier.

AU : Dec.-18, 19

Ans. : Access Specifier is used to set the accessibility of class members. There are three access specifiers in Java - (1) Public (2) Private (3) Protected.

Q.26 What is Javadoc ?

AU : Dec.-19

Ans. : The Javadoc is a standard way to comment the java code. It has a special format of commenting the java code.

Q.27 Can Java source file be saved using a name other than the class name. Justify. **AU : May-19**

Ans. : Yes, we can save the java source file using a name other than the class name. But we should compile the program with file name and should run the program with the class name in which the main method exist. And there must not be any public class defined inside this java source file. For example, consider following Java code which is saved using the file name **test.java**

class A

```
{
  public static void main(String args[])
  {
    System.out.println("Class A");
  }
}
```

```
}

class B
{
    public static void main(String args[])
    {
        System.out.println("Class B");
    }
}
```

Output

Step 1 : Compile the above program using following command
javac test.java

Step 2 : Now execute the above program using
java A

The output will be
Class A

Step 3 : If you execute the above code as
java B

The output will be
ClassB

UNIT II

2

Inheritance, Packages and Interfaces

Syllabus

Overloading Methods - Objects as Parameters - Returning Objects - Static, Nested and Inner Classes. Inheritance : Basics - Types of Inheritance - Super keyword -Method Overriding - Dynamic Method Dispatch - Abstract Classes - final with Inheritance, Packages and Interfaces: Packages - Packages and Member Access - Importing Packages - Interfaces.

Contents

2.1	Overloading Methods	
2.2	Objects as Parameters	
2.3	Returning Objects	May-12
2.4	Static, Nested and Inner Classes.....	Dec.-10, 11, 13, May-13,Marks 8
2.5	Inheritance : Basics	
2.6	Types of Inheritance.....	May-11, 12, Dec.-19,Marks 16
2.7	What is Protected Member ?	
2.8	Implementation of Different Types of Inheritance	May-13, 15, Dec.-14, 18,Marks 16
2.9	Super Keyword.....	Dec.-14,Marks 8
2.10	Method Overriding.....	May-12, Dec.-13,Marks 16
2.11	Polymorphism	
2.12	Abstract Classes.....	Dec.-10, 12, 13, 19, May-13Marks 8
2.13	The Final with Inheritance	Dec.-13,Marks 8
2.14	The finalize() method	Dec.-18,Marks 7
2.15	Packages	May-12, 13, 14, Dec.-11, 12, Dec.-13,Marks 16
2.16	Interfaces	May-19,Marks 6
2.17	Implementing Interface.....	Dec.-14,Marks 8
2.18	Applying Interfaces.....	May-13, 19,Marks 16
2.19	Multiple Inheritance	
2.20	Two Marks Questions with Answers	

2.1 Overloading Methods

Overloading is a mechanism in which we can use many methods having the same function name but can pass different number of parameters or different types of parameter.

For example :

```
int sum(int a,int b);
double sum(double a,double b);
int sum(int a,int b,int c);
```

That means, by overloading mechanism, we can handle different number of parameters or different types of parameter by having the same method name.

Following Java program explains the concept overloading -

Java Program [OverloadingDemo.java]

```
public class OverloadingDemo {
    public static void main(String args[]) {
        System.out.println("Sum of two integers");
        Sum(10,20); <----- line A
        System.out.println("Sum of two double numbers");
        Sum(10.5,20.4); <----- line B
        System.out.println("Sum of three integers");
        Sum(10,20,30); <----- line C
    }
    public static void Sum(int num1,int num2)
    {
        int ans;
        ans=num1+num2;
        System.out.println(ans);
    }
    public static void Sum(double num1,double num2)
    {
        double ans;
        ans=num1+num2;
        System.out.println(ans);
    }
    public static void Sum(int num1,int num2,int num3)
    {
        int ans;
        ans=num1+num2+num3;
        System.out.println(ans);
    }
}
```

Output

```
F:\test>javac OverloadingDemo.java
```

```
F:\test>java OverloadingDemo
```

Sum of two integers

30

Sum of two double numbers

30.9

Sum of three integers

60

Program Explanation

In above program, we have used three different methods possessing the same name. Note that,

on line A

We have invoked a method to which two integer parameters are passed. Then compiler automatically selects the definition **public static void Sum(double num1,double num2)** to fulfill the call. Hence we get the output as 30 which is actually the addition of 10 and 20.

on line B

We have invoked a method to which two double parameters are passed. Then compiler automatically selects the definition **public static void Sum(double num1,double num2)** to fulfill the call. Hence we get the output as 30.9 which is actually the addition of 10.5 and 20.4.

on line C

We have invoked a method to which the three integer parameters are passed. Then compiler automatically selects the definition **public static void Sum(int num1,int num2,int num3)** to fulfill the call. Hence we get the output as 60 which is actually the addition of 10, 20 and 30.

2.2 Objects as Parameters

The object can be passed to a method as an argument. Using dot operator the object's value can be accessed. Such a method can be represented syntactically as -

```
Data_Type name_of_method(object_name)
{
    //body of method
}
```

Following is a sample Java program in which the method **area** is defined. The parameter passed to this method is an **object**.

Java Program[ObjDemo.java]

```

public class ObjDemo {
    int height;
    int width;
    ObjDemo(int h,int w)
    {
        height=h;
        width=w;
    }
    void area(ObjDemo o)
    {
        int result=(height+o.height)*(width+o.width);
        System.out.println("The area is "+result);
    }
}
class Demo {
    public static void main(String args[])
    {
        ObjDemo obj1=new ObjDemo(2,3);
        ObjDemo obj2=new ObjDemo(10,20);
        obj1.area(obj2);
    }
}

```

Method with object as parameter

Output

F:\test>javac ObjDemo.java

F:\test>java Demo

The area is 276

height=2 and o.height=10
width=3 and o.width=20
Hence,
result=(2+10)*(3+20)
=12*23
=276

Program Explanation

- The method, **area** has object **obj2** as argument. And there is another object named **obj1** using which the method **area** is invoked. Inside the method **area**, the height and width values of invoking object are added with the height and width values of the object to be passed.
- When an object needs to be passed as a parameter to the function then constructor is built. Hence we have to define **constructor** in which the values of the object are used for initialization.

2.3 Returning Objects

AU : IT : May-12

- We can return an object from a method. The data type for such method is a **class type**.

Java Program[ObjRetDemo.java]

```

public class ObjRetDemo {
    int a;
    ObjRetDemo(int val)
    {
        a=val;
    }
    ObjRetDemo fun()
    {
        ObjRetDemo temp=new ObjRetDemo(a+5); //created a new object
        return temp; //returning the object from this method
    }
}
class ObjRet {
    public static void main(String args[])
    {
        ObjRetDemo obj2=new ObjRetDemo(20);
        ObjRetDemo obj1;
        obj1=obj2.fun(); //obj1 gets the value from object temp
        System.out.println("The returned value is = "+obj1.a);
    }
}

```

Data type of this method is
name of the class

Output

F:\test>javac ObjRetDemo.java

F:\test>java ObjRet
The returned value is = 25

Ex. 2.3.1 : Write Java program for computing Fibonacci series.

AU : IT : May-12

Sol. :

```

*****
Program for computing the number in the fibonacci series at
certain location. For example the eighth number in fibonacci
series will be 21. The fibonacci series is 1 1 2 3 5 8 13 21 34...
*****
import java.io.*;
import java.util.*;
class Fibonacci
{
    public int fib(int n)
    {
        int x,y;

```

```

if(n<=1)
    return n;
x=fib(n-1);
y=fib(n-2);
return (x+y);
}
}//end of class
class FibonacciDemo
{
public static void main(String[] args) throws IOException
{
Fibonacci f=new Fibonacci();
int n=8;
System.out.println("\nThe number at "+n+" is "+f.fib(n));
}
}

```

Output

D:\>javac FibonacciDemo.java

D:\>java FibonacciDemo

The number at 8 is 21

AU : CSE : Dec.-10, 11, 13, May-13, Marks 8

Inner classes are the nested classes. That means these are the classes that are defined inside the other classes. The syntax of defining the inner class is -

```

Access_modifier class OuterClass
{
    //code
    Access_modifier class InnerClass
    {
        //code
    }
}

```

Following are some **properties of inner class** -

- The outer class can inherit as many number of inner class objects as it wants.
- If the outer class and the corresponding inner class both are **public** then any other class can create an instance of this inner class.
- The inner class objects do not get instantiated with an outer class object.
- The outer class can call the private methods of inner class.
- Inner class code has free access to all elements of the outer class object that contains it.

- If the inner class has a variable with same name then the outer class's variable can be accessed like this -

outerclassname.this.variable_name

There are four types of inner classes -

1. Static member classes

- This inner class is defined as the static member variable of another class.
- Static members of the outer class are visible to the **static inner class**.
- The non-static members of the outer class are not available to inner class.

Syntax

Access_modifier class OuterClass

```
{
    //code
    public static class InnerClass
    {
        //code
    }
}
```

2. Member classes

- This type of inner class is non-static member of outer class.

3. Local classes

- This class is defined within a Java code just like a local variable.
- Local classes are never declared with an access specifier.
- The scope inner classes is always restricted to the block in which they are declared.
- The local classes are completely hidden from the outside world.

Syntax

Access_modifier class OuterClass

```
{
    //code
    Access_modifier return_type methodname(arguments)
    {
        class InnerClass
        {
            //code
        }
        //code
    }
}
```

4. Anonymous classes

- Anonymous class is a local class without any name.
- Anonymous class is a one-shot class- created exactly where needed
- The anonymous class is created in following situations
 - When the class has very short body.
 - Only one instance of the class is needed.
 - Class is used immediately after defining it.
- The anonymous inner class can extend the class, it can implement the interface or it can be declared in method argument.

Example

```
class MyInnerClass implements Runnable
{
    public void run()
    {
        System.out.println("Hello");
    }
}
class DemoClass
{
    public static void main(String[] args)
    {
        MyInnerClass my=new MyInnerClass();
        Thread th=new Thread(my);
        my.start();
    }
}
```

Review Questions

1. What is static inner class ? Explain with example.
2. Discuss in detail about inner class with its usefulness.
3. Define Inner classes. How to access object state using inner classes ? Give an example.
4. Discuss the object and inner classes with examples.

AU : CSE : Dec.-10, Marks 8

AU : CSE : Dec.-11, Marks 8

AU : CSE : May-13, Marks 8

AU : CSE : Dec.-13, Marks 8

2.5 Inheritance: Basics

- **Definition :** Inheritance is a mechanism in Java by which derived class can borrow the properties of base class and at the same time the derived class may have some additional properties.
- The inheritance can be achieved by incorporating the definition of one class into another using the keyword extends.

Advantages of Inheritance

One of the key benefits of inheritance is to minimize the amount of duplicate code in an application by sharing common code amongst several subclasses.

1. **Reusability :** The base class code can be used by derived class without any need to rewrite the code.
2. **Extensibility :** The base class logic can be extended in the derived classes.
3. **Data hiding :** Base class can decide to keep some data private so that it cannot be altered by the derived class.
4. **Overriding :** With inheritance, we will be able to override the methods of the base class so that meaningful implementation of the base class method can be designed in the derived class

2.5.1 Concept of Base and Derived Class

- The inheritance is a mechanism in which the child class is derived from a parent class.
- This derivation is using the keyword extends.
- The parent class is called **base class** and child class is called **derived class**.
- For example

```

Class A           ← This is Base class
{
    ...
}

Class B extends A ← This is Derived class
{
    ...
    ... // uses properties of A
}

```

- Inheritance is represented diagrammatically as follows

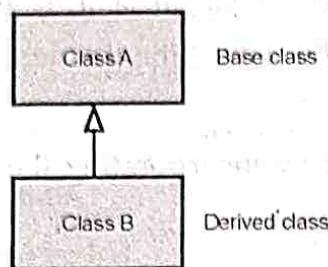


Fig. 2.5.1

2.6 Types of Inheritance

AU : May-11, 12, Dec.-19, Marks 16

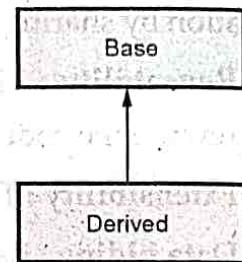


Fig. 2.6.1 Single inheritance

2. Multiple inheritance :

In multiple inheritance the derived class is derived from more than one base class.

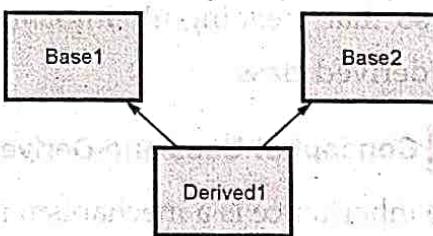


Fig. 2.6.2 Multiple inheritance

3. Multilevel inheritance :

When a derived class is derived from a base class which itself is a derived class then that type of inheritance is called multilevel inheritance.

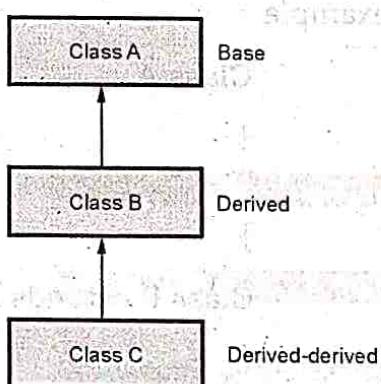


Fig. 2.6.3 Multilevel inheritance

Refer Fig. 2.6.3.

4. Hybrid inheritance :

When two or more types of inheritances are combined together then it forms the hybrid inheritance. The following Fig. 2.6.4 represents the typical scenario of hybrid inheritance.

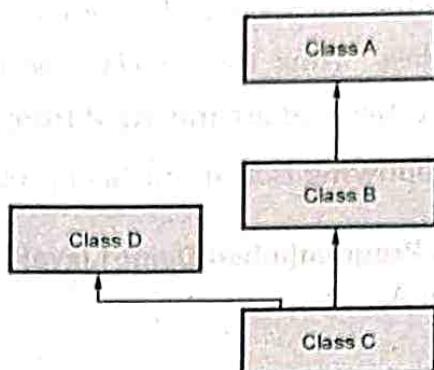


Fig. 2.6.4 Hybrid inheritance

Review Questions

1. Explain in detail as how inheritance is supported in Java with necessary examples

AU : May-11, Marks 8

2. Give an elaborate discussion on inheritance.

AU : May-12, Marks 16

3. Explain in detail about various types of inheritance in java with neat diagram.

AU : Dec.-19, Marks 13

2.7 What is Protected Member ?

- Protected mode is another access specifier which is used in inheritance.
- The protected mode allows accessing the members to all the classes and subclasses in the same package as well as to the subclasses in other package. But the non subclasses in other package can not access the protected members.
- For example, if some variable is declared as protected, then the class itself can access it, its subclass can access it, and any class in the same package can also access it.
- Similarly if the variable is declared as private then that variable is accessible by that class only and its subclass or package can not access it.

2.8 Implementation of Different Types of Inheritance

AU : May-13, 15, Dec.-14, 18, Marks 16

2.8.1 Single Inheritance

- The class which is inherited is called the **base class** or the **superclass** and the class that does the inheriting is called the **derived class** or the **subclass**.
- The method defined in base class can be used in derived class. There is no need to redefine the method in derived class. Thus inheritance promotes **software reuse**.
- The subclass can be defined as follows -

```

class nameofSubclass extends superclass
{
    variable declarations
    method declarations
}
  
```

(A) A works in JCB A
 (B) A works in JCB B
 (C) A works in JCB C
 (D) A works in JCB D
 (E) A works in JCB E

Note that the keyword **extends** represents that the properties of superclass are extended to the subclass. Thus the subclass will now have both the properties of its own class and the properties that are inherited from superclass.

- Following is a simple Java program that illustrates the concept of single inheritance.

Java Program [InheritDemo1.java]

```

class A
{
    int a;
    void set_a(int i)
    {
        a=i;
    }
    void show_a()
    {
        System.out.println("The value of a=" +a);
    }
}

class B extends A //extending the base class A
{
    int b;
    void set_b(int i)
    {
        b=i;
    }
    void show_b()
    {
        System.out.println("The value of b=" +b);
    }
    void mul()
    {
        int c;
        c=a*b;
        System.out.println(" The value of c=" +c);
    }
}

class InheritDemo1
{
    public static void main(String args[])
    {
        A obj_A=new A();
        B obj_B=new B();
        obj_B.set_a(10);
        obj_B.set_b(20);
        obj_B.show_a();
    }
}

```

Note that object of class B is accessing method of class A

```

        obj_B.show_b();
        obj_B.mul();
    }
}

```

Output

F:\test>javac InheritDemo1.java

F:\test>java InheritDemo1

The value of a = 10

The value of b = 20

The value of c = 200

Program Explanation

In above program, we have created two classes : class A and B. In class A we have declared one integer a and in class B we have declared an integer b. There are two methods defined in class A namely: set_a and show_a. Similarly, in class B there are two methods defined namely: set_b and show_b. As the name suggests these methods are for setting the values and for showing the contents.

In the class InheriDemo1, in the main function we have created two objects for class A and class B. The program allows us to access the variable a (belonging to class A) and the variable b (belonging to class B) using the object for class B. Thus it is said that class B has inherited value of variable a .

The class A is called **Superclass** and the class B is called **Subclass**. A

Superclass is also called as **parent class** or **base class**. Similarly, the Subclass is also called as **child class** or **derived class**.

Ex. 2.8.1 : Write a java program to calculate area of rectangle using the single inheritance.

Sol. :

class Shape

```

{
    int len,br;
    void setValues(int a,int b)
    {
        len=a;
        br=b;
    }
}

```

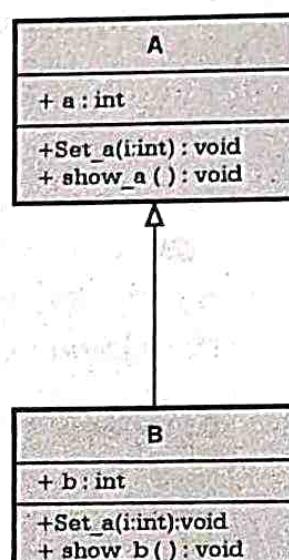


Fig. 2.8.1 Single inheritance

```

}
class Rectangle extends Shape
{
    int area()
    {
        return len*br;
    }
}
class test
{
    public static void main(String args[])
    {
        Rectangle r=new Rectangle();
        r.setValues(10,20);
        System.out.println("\n Area of rectangle is "+r.area());
    }
}

```

Output

```

D:\>javac test.java
D:\>java test
Area of rectangle is 200
D:\>

```

2.8.2 Multilevel Inheritance

The multilevel inheritance is a kind of inheritance in which the derived class itself derives the subclasses further.

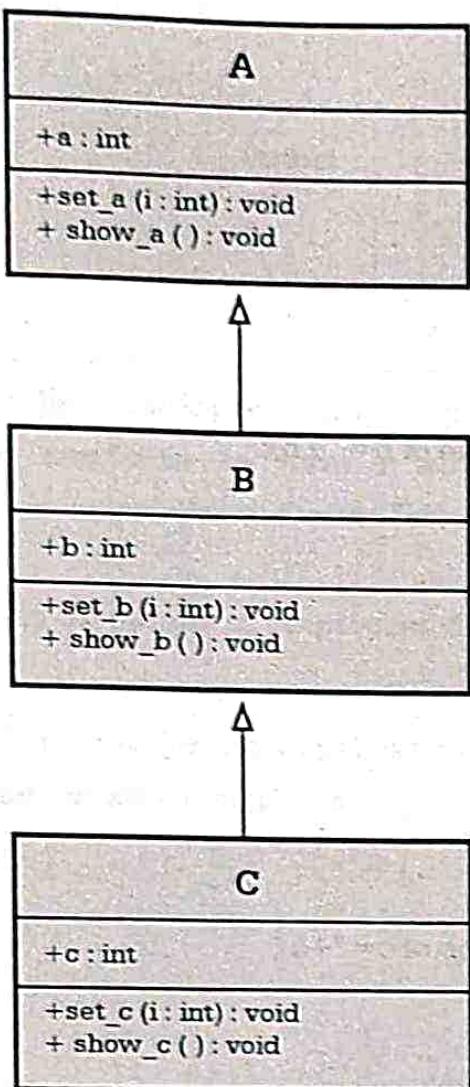


Fig. 2.8.2 Multilevel inheritance

In the following program, we have created a base class A from which the subclass B is derived. There is a class C which is derived from class B. In the function main we can access any of the field in the class hierarchy by creating the object of class C.

Java Program[MultilevInherit.java]

```

class A
{
    int a;
    void set_a(int i)
    {
        a=i;
    }
    void show_a()
    {
        System.out.println("The value of a= "+a);
    }
}
  
```

```
}

class B extends A
{
    int b;
    void set_b(int i)
    {
        b=i;
    }
    void show_b()
    {
        System.out.println("The value of b= "+b);
    }
}

class C extends B
{
    int c;
    void set_c(int i)
    {
        c=i;
    }
    void show_c()
    {
        System.out.println("The value of c= "+c);
    }
    void mul()
    {
        int ans;
        ans=a*b*c;
        System.out.println(" The value of ans= "+ans);
    }
}

class MultiLvlInherit
{
    public static void main(String args[])
    {
        A obj_A=new A();
        B obj_B=new B();
        C obj_C=new C();
        obj_C.set_a(10);
        obj_C.set_b(20);
        obj_C.set_c(30);
        obj_C.show_a();
        obj_C.show_b();
        obj_C.show_c();
    }
}
```

```

    obj.C.mul();
}
}

```

Output

```

The value of a= 10
The value of b= 20
The value of c= 30
The value of ans= 6000

```

Ex. 2.8.2 : Declare a class called employee having employee_id and employee_name as members. Extend class employee to have a subclass called salary having designation and monthly_salary as members. Define following :

- Required constructors.
- A method to find and display all details of employees drawing salary more than 20000/-.
- Method main for creating an array for storing these details given as command line arguments and showing usage of above methods.

Sol. :

```

class employee
{
    int employee_id;
    String employee_name;
}
class salary extends employee //derived class
{
    String designation;
    double monthly_salary;
    salary() //default constructor
    {}
    salary(int employee_id, String employee_name, String designation,
          double monthly_salary) //parameterised construct.
    {
        this.employee_id = employee_id;
        this.employee_name = employee_name;
        this.designation = designation;
        this.monthly_salary = monthly_salary;
    }
    void fun(String emp[][]) //Function displaying salary>20000
    {
        for(int i=0;i<emp.length;i++)
        {
            if( (Double.parseDouble(emp[i][3])) > 20000 )
            {

```

```

        System.out.println("\nEmployee Drawing Salary More than ' 20000/- : ");
        System.out.println("Id = "+emp[i][0]);
        System.out.println("Name = "+emp[i][1]);
        System.out.println("Designation = "+emp[i][2]);
        System.out.println("Salary = "+emp[i][3]);
        System.out.println("-----");
    }
}
}

public class employeetest
{
    public static void main(String args[])
    {
        salary obj[] = new salary[5];
        String Details[][];
        obj[0] = new salary(1,"AAA","Accountant",12000);
        obj[1] = new salary(2,"BBB","Manager",30000);
        obj[2] = new salary(3,"CCC","Executive",2000);
        obj[3] = new salary(4,"DDD","CEO",50000);
        obj[4] = new salary();
        try
        {
            obj[4].employee_id = Integer.parseInt(args[0]);
            obj[4].employee_name = args[1];
            obj[4].designation = args[2];
            obj[4].monthly_salary = Double.parseDouble(args[3]);
        }
        catch(NumberFormatException e)
        {
            System.out.println("Exception : "+ e);
        }
        Details = new String[obj.length][4];
        for(int i=0;i<obj.length;i++)
        {
            Details[i][0] = String.valueOf(obj[i].employee_id);
            Details[i][1] = obj[i].employee_name;
            Details[i][2] = obj[i].designation;
            Details[i][3] = String.valueOf(obj[i].monthly_salary);
        }
        obj[4].fun(Details);
    }
}

```

Output

E:\test>javac employeetest.java

E:\test>java employetest 5 EEE Manager 20000

Employee Drawing Salary More than ` 20000/- :

Id = 2

Name = BBB

Designation = Manager

Salary = 30000.0

Employee Drawing Salary More than Rs. 20000/- :

Id = 4

Name = DDD

Designation = CEO

Salary = 50000.0

Ex. 2.8.3 : Create a Java class Shape with constructor to initialize the one parameter "dimension". Now create three subclasses of Shape with following methods :

(i) n"Circle" with methods to calculate the area and circumference of the circle with dimension as radius.

(ii) "Square" with methods to calculate the area and length of diagonal of square with dimension as length of one side.

(iii) "Sphere" with methods to calculate the volume and surface area of sphere with dimension as radius of the sphere. Write appropriate main method to create object of each class and test every method

AU : May-15, Marks 16

Sol. :

```
class Shape
{
    double dimension;
    Shape()
    {
        dimension=0;
    }
}

class Circle extends Shape
{
    Circle(double r)
    {
        dimension = r;
    }
}
```

```

void display()
{
    System.out.println("-----");
    System.out.println("The radius of circle is: "+dimension);
}

double area()
{
    System.out.print("Area Of Circle : ");
    return (3.14*dimension*dimension);
}

double circum()
{
    System.out.print("Circumference Of Circle : ");
    return (2*3.14*dimension);
}

}

class Square extends Shape
{
    Square(double d)
    {
        dimension = d;
    }

    void display()
    {
        System.out.println("-----");
        System.out.println("The side of square is: "+dimension);
    }

    double area()
    {
        System.out.print("Area Of Square : ");
        return (dimension*dimension);
    }

    double LenDiagonal()
    {
        System.out.print("Length of Diagonal of Square : ");
        return (dimension*Math.sqrt(2));
    }
}

class Sphere extends Shape
{
    Sphere(double r)
    {
        dimension = r;
    }

    void display()
}

```

```

{
    System.out.println("-----");
    System.out.println("The radius of sphere is: " + dimension);
}
double area()
{
    System.out.print("Surface Area Of Sphere : ");
    return (4 * 3.14 * dimension * dimension);
}
double volume()
{
    System.out.print("Volume of Sphere : ");
    return ((4 / 3) * 3.14 * dimension * dimension * dimension);
}
}

class InheritanceDemo
{
    public static void main(String args[])
    {
        Circle cir = new Circle(10);
        cir.display();
        System.out.println(cir.area());
        System.out.println(cir.circum());
        Square sq = new Square(10);
        sq.display();
        System.out.println(sq.area());
        System.out.println(sq.LenDiagonal());
        Sphere sph = new Sphere(10);
        sph.display();
        System.out.println(sph.area());
        System.out.println(sph.volume());
    }
}

```

Output

The radius of circle is: 10.0
 Area Of Circle : 314.0
 Circumference Of Circle : 62.800000000000004

The side of square is: 10.0
 Area Of Square : 100.0
 Length of Diagonal of Square : 14.142135623730951

The radius of sphere is: 10.0
 Surface Area Of Sphere : 1256.0
 Volume of Sphere : 3140.0

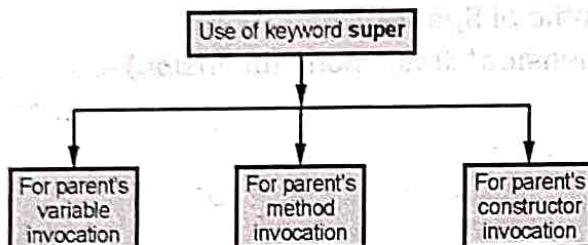
Review Questions

- What is inheritance? Write a program for inheriting a class. AU : May-13, Marks 8
- What is inheritance? With diagrammatic illustrations and Java programs illustrate different types of inheritance. Give self explanatory comments in your program. AU : Dec.-14, Marks 16
- Define Inheritance. With diagrammatic illustration and java programs illustrate the different types of inheritance with an example. AU : Dec.-18, Marks 7

2.9 Super KeywordAU : Dec.-14, Marks 8

Super is a keyword used to access the immediate parent class from subclass.

There are three ways by which the keyword **super** is used.

**Fig. 2.9.1 Use of super**

Let us understand these uses of keyword super with illustrative Java programs.

1. The super() is used to invoke the class method of immediate parent class.**Java Program[B.java]**

```

class A
{
    int x=10;
}

class B extends A
{
    int x=20;
    void display()
    {
        System.out.println(super.x);
    }
    public static void main(String args[])
    {
        B obj=new B();
        obj.display();
    }
}
  
```

Output

10

Program Explanation : In above program class A is a immediate parent class of class B. Both the class A and Class B has variables x. In class A, the value of x variable is 10 and in class B the value of variable x is 20. In display function if we would write

```
System.out.println(x);
```

The output will be 20 but if we user `super.x` then the variable `x` of class `A` will be referred. Hence the output is 10.

2. The `super()` is used to access the class variable of immediate parent class.

Java Program[B.java]

```
class A
{
    void fun()
    {
        System.out.println("Method: Class A");
    }
}

class B extends A
{
    void fun()
    {
        System.out.println("Method: Class B");
    }

    void display()
    {
        super.fun();
    }

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

Output

Method: Class A

Program Explanation : In above program, the derived class can access the immediate parent's class method using `super.fun()`. Hence is the output. You can change `super.fun()` to `fun()`. Then note that in this case, the output will be invocation of subclass method `fun`.

3. The `super()` is used to invoke the immediate parent class constructor.

Java Program[B.java]

```
class A
{
    A()
    {
        System.out.println("Constructor of Class A");
    }
}
```

```

}
class B extends A
{
    B()
    {
        super();
        System.out.println("Constructor of Class B");
    }
    public static void main(String args[])
    {
        B obj=new B();
    }
}

```

Output

Constructor of Class A
Constructor of Class B

Program Explanation : In above program, the constructor in class B makes a call to the constructor of immediate parent class by using the keyword **super**, hence the print statement in parent class constructor is executed and then the print statement for class B constructor is executed.

Ex. 2.9.1 : What is inheritance ? How will you call parameterized constructor and overrided method from parent class in sub class ?

AU : Dec.-14, Marks 8

Sol. : Inheritance: - Inheritance is a mechanism in Java by which derived class can borrow the properties of base class and at the same time the derived class may have some additional properties.

Calling Parameterized constructor from parent class in subclass

```

class Parentclass
{
    //no-arg constructor
    Parentclass()
    {
        System.out.println("no-arg constructor of parent class");
    }
    //arg or parameterized constructor
    Parentclass(String str)
    {
        System.out.println(str+" This is parameterized constructor of parent class");
    }
}

```

```

void display(String str)
{
    System.out.println(str+" This is overridden method");
}

class Subclass extends Parentclass
{
    Subclass()
    {
        super("Welcome");//calling super class's parameterized constructor
        System.out.println("Constructor of child class");
    }

    void display()//overridden method
    {
        super.display("Hi"); //calling super class overridden method
    }

    public static void main(String args[])
    {
        Subclass obj= new Subclass();
        obj.display();
    }
}

```

Output

Welcome This is parameterized constructor of parent class
 Constructor of child class
 Hi This is overridden method

Review Question

- With suitable program segments describe the usage of super keyword.

2.10 Method Overriding

SPPU : CSE : May-12, Dec.-13, Marks 16

Method overriding is a mechanism in which a subclass inherits the methods of superclass and sometimes the subclass modifies the implementation of a method defined in superclass.

The method of superclass which gets modified in subclass has the same name and type signature. The overridden method must be called from the subclass. Consider following Java Program, in which the method(named as fun) in which a is assigned with some value is modified in the derived class. When an overridden method is called from within a subclass, it will always refer to the version of that method re-defined by the subclass. The version of the method defined by the superclass will be hidden.

Java Program[OverrideDemo.java]

```

class A
{
    int a=0;
    void fun(int i)
    {
        this.a=i;
    }
}
class B extends A
{
    int b;
    void fun(int i)
    {
        int c;
        b=20;
        super.fun(i+5);
        System.out.println("value of a:"+a);
        System.out.println("value of b:"+b);
        c=a*b;
        System.out.println("The value of c= "+c);
    }
}
class OverrideDemo
{
    public static void main(String args[])
    {
        B obj_B=new B();
        obj_B.fun(10);//function re-defined in derived class
    }
}

```

Output

```

F:\test>javac OverrideDemo.java
F:\test>java OverrideDemo
value of a:15
value of b:20
The value of c= 300

```

Program Explanation

In above program, there are two class - class A and class B. The class A acts as a superclass and the class B acts as a subclass. In class A, a method **fun** is defined in which the variable **a** is assigned with some value. In the derived class B, we use the same function name **fun** in which, we make use of **super** keyword to access the variable **a** and then it is multiplied by **b** and the result of multiplication will be printed.

Rules to be followed for method overriding

1. The private data fields in superclass are not accessible to the outside class. Hence the method of superclass using the private data field cannot be overridden by the subclass.
2. An instance method can be overridden only if it is accessible. Hence private method can not be overridden.
3. The static method can be inherited but can not be overridden.
4. Method overriding occurs only when the name of the two methods and their type signatures is same.

Difference between Method Overloading and Method Overriding

Method Overloading	Method Overriding
The method overloading occurs at compile time.	The method overriding occurs at the run time or execution time.
In case of method overloading different number of parameters can be passed to the function.	In function overriding the number of parameters that are passed to the function are the same.
The overloaded functions may have different return types.	In method overriding all the methods will have the same return type.
Method overloading is performed within a class.	Method overriding is normally performed between two classes that have inheritance relationship.

Review Questions

1. Differentiate between method overloading and method overriding. Explain both with an example.

AU : CSE : May-12, Marks 16

2. What is meant by overriding method ? Give example.

AU : CSE, Dec.-13, Marks 5

2.11 Polymorphism

Basic concept : Polymorphism is a mechanism which allows to have many forms of the method having the same name. That means, a method A() may take an instance of an object of one class and may execute its method or the same method A() may take another instance of an object and may execute the method belonging to another class. In the following Java program we have taken the superclass A and build a multilevel inheritance.

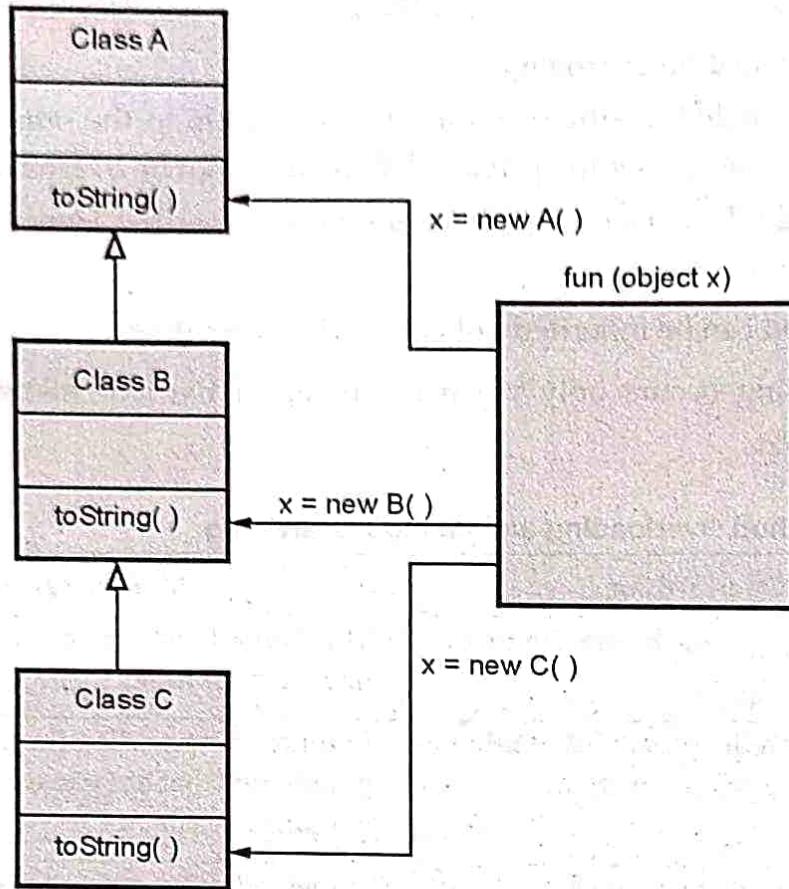


Fig. 2.11.1 : Polymorphism

Using object of corresponding class corresponding method is invoked by the same function fun.

The implementation is given by following Java program -

Java Program[PolymorphismDemo.java]

```

class A extends Object
{
    public String toString()
    {
        return "A";
    }
}

class B extends A
{
    public String toString()
    {
        return "B";
    }
}

class C extends B
{
    public String toString()
    {
```

```

        return "C";
    }

}

public class PolymorphismDemo
{
    public static void main(String[] args) {
        fun(new C());//invokes the method toString() of class C
        fun(new B());//invokes the method toString() of class B

        fun(new A());//invokes the method toString() of class A
    }

    public static void fun(Object x) {
        System.out.println(x.toString());
    }
}

```

Output

F:\test>javac PolymorphismDemo.java

F:\test>java PolymorphismDemo

C
B
A

Program Explanation

The same function fun is used to execute the methods of various class. The selection of the method of the class (i.e. toString method) depends upon the instance of the object which is passed to the function fun. Which implementation is to be used is determined dynamically by the Java Virtual Machine. This mechanism is called **dynamic binding**.

2.11.1 Dynamic Method Dispatch

The dynamic method dispatch is also called as **runtime polymorphism**. During the run time polymorphism, a call to overridden method is resolved at run time. The overridden method is called using the reference variable of a super class(or base class). The determination of which method to invoke is done using the object being referred by the reference variable.

Following is a simple Java program that illustrates the Run time polymorphism.

```

class Base
{
    void display()
    {
        System.out.println("\n Base Method Called");
    }
}

```

```

}
class Derived extends Base
{
void display() //overridden method
{
    System.out.println("\n Derived Method Called");
}
}
public class RunPolyDemo
{
public static void main(String args[])
{
    Base obj=new Derived(); //obj is reference to base class
                           // which is referred by the derived class
    obj.display(); //method invocation determined at run time
}
}

```

Output

D:\test>javac RunPolyDemo.java

D:\test>java RunPolyDemo

Derived Method Called

Program Explanation :

In above program, the display method is an **overridden method** because with the same signature we are modifying it in its derived class. This method is invoked in the main() function using the reference obj. This reference variable is of superclass type. It is assigned with the reference of Derived class. At run time it is decided that the display method of derived class should be invoked. Hence it is known as run time polymorphism.

Ex. 2.11.1 : Declare a class called book having author_name as private data member. Extend book class to have two sub classes called book_publication and paper_publication. Each of these classes have private member called title. Write a complete program to show usage of dynamic method dispatch (dynamic polymorphism) to display book or paper publications of given author. Use command line arguments for inputting data.

Sol. :

```

import java.util.Scanner;
class book
{
    private String[] author_name = {"Puntambekar", "Godse"};
    void display()
    {
        System.out.println("Author Name : " + author_name[0]);
        System.out.println("Author Name : " + author_name[1]);
    }
}

```

```

{
    System.out.println("Author names: ");
    for(int i=0;i<author_name.length;i++)
    {
        System.out.println("\t"+author_name[i]);
    }
}

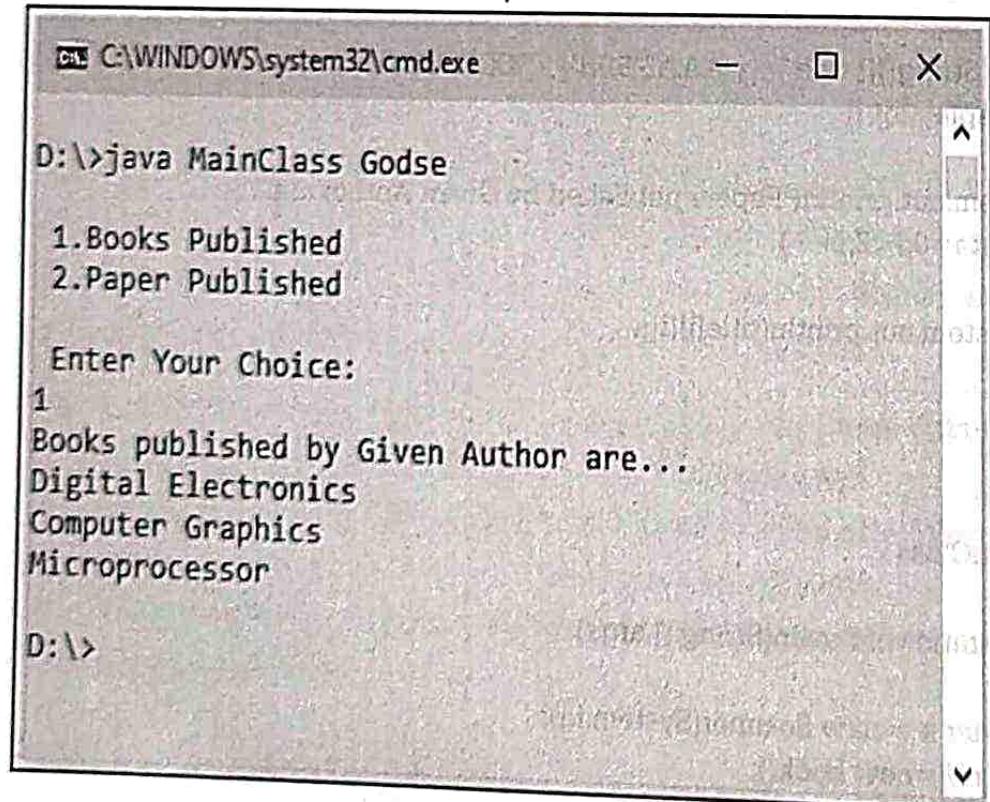
class book_publication extends book
{
    private String[][] title = {"Java Programming","Compiler Design","DAA"}, {"Digital Electronics","Computer Graphics","Microprocessor"};
    void display(int j)
    {
        System.out.println("Books published by Given Author are...");
        for(int i=0;i<3;i++)
        {
            System.out.println(title[j][i]);
        }
    }
}

class paper_publication extends book
{
    private String[][] title = {"AAA","BBB"}, {"XXX","YYY"};
    void display(int j)
    {
        System.out.println("Papers published by Given Author are...");
        for(int i=0;i<2;i++)
        {
            System.out.println(title[j][i]);
        }
    }
}

class MainClass
{
    public static void main(String [] args)
    {
        Scanner s = new Scanner(System.in);
        book obj=new book();
        book_publication bookpub=new book_publication();
        paper_publication paperpub=new paper_publication();
        int author_name=0,choice;
        if(args[0].equals("Puntambekar"))
            author_name=0;
    }
}

```

Output



2.12 Abstract Classes

AU : CSE : Dec. 10, 12, 13, 19, IT : May-13, Marks 8

- In inheritance hierarchy, the superclass is very general and less specific. This class does nothing but only specifies the member functions that can be used in hierarchy. Such a class is called **abstract class**.

For example - In the following Java program we have created three classes - class A is a superclass containing two methods, the class B and class C are inherited from class A. The class A is an abstract class because it contains one abstract method **fun1()**. We have defined this method as abstract because, its definition is overridden in the subclasses B and C, another function of class A that is **fun2()** is a normal function. Refer Fig. 2.12.1.

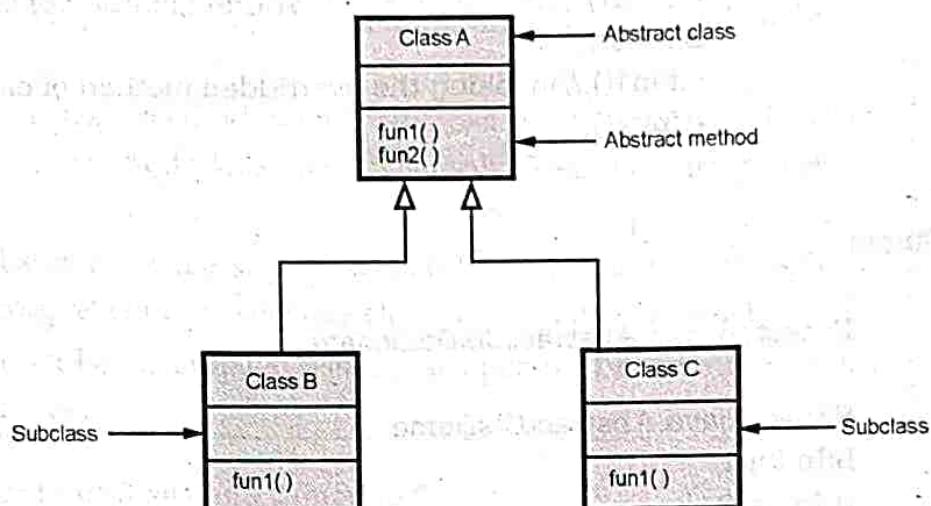


Fig. 2.12.1 Abstract class

Java Program [AbstractClsDemo.java]

```

abstract class A
{
    abstract void fun1();
    void fun2()
}

System.out.println("A:Infun2");

}

class B extends A
{
    void fun1()
    {
        System.out.println("B:In fun1");
    }
}

class C extends A
{
    void fun1()
    {
        System.out.println("C:In fun1");
    }
}
  
```

This method is so abstract that it has not definition body.
This is an **abstract method**.

```

}
}

public class AbstractClsDemo
{
    public static void main(String[] args)
    {
        B b=new B();
        C c=new C();
        b.fun1(); //invoking the overridden method of class B
        b.fun2();
        c.fun1(); //invoking the overridden method of class C
        c.fun2();
    }
}

```

Output

```
F:\test>javac AbstractClsDemo.java
```

```
F:\test>java AbstractClsDemo
B:In fun1
A:In fun2
C:In fun1
A:In fun2
```

Program Explanation

- In above program, the class A is a superclass. It is an abstract class as well. The name of this class is preceded by the keyword **abstract**. This class is abstract because it contains an abstract method **fun1**. The method is called abstract because it does not have any definition body. Note that the abstract method should be declared with the keyword **abstract**.
- There are two classes B and C which are subclasses of superclass A. The function definition **fun1** is overridden in these classes.
- In the **main function** we can access the methods of the subclasses by instantiating their objects. That is why we have created **b** as an **object of class B** and **c** as an **object of class C**. Using these objects appropriate **fun1** can be invoked. Note that the **fun2** will always be from class A even if we call it using the object of class B or C.
- If we write a statement **A a=new A()** i.e. if we instantiate the abstract class then it will generate compiler error. That means the **abstract classes can not be instantiated**.

Points to Remember about abstract classes and abstract methods

1. An abstract method must be present in an abstract class only. It should not be present in a non-abstract class.
2. In all the non-abstract subclasses extended from an abstract superclass all the abstract methods must be implemented. An un-implemented abstract method in the subclass is not allowed.
3. Abstract class cannot be instantiated using the new operator.
4. A constructor of an abstract class can be defined and can be invoked by the subclasses.
5. A class that contains abstract method must be abstract but the abstract class may not contain an abstract method. This class is simply used as a base class for defining new subclasses.
6. A subclass can be abstract but the superclass can be concrete. For example the superclass Object is concrete but the subclass class A of it can be abstract.
7. The abstract class cannot be instantiated using new operator but an abstract class can be used as a data type.

2.12.1 Difference between Abstract Class and Concrete Class

Sr. No.	Abstract class	Concrete class
1.	The abstract keyword is used to define the abstract class.	The keyword class is used to define the concrete class.
2.	The abstract class have partial or no implementation at all.	The concrete class contains the data members and member functions defined within it.
3.	Abstract class can not be instantiated.	Concrete class are usually instantiated in order to access the belonging data members and member functions.
4.	Abstract classes may contain abstract methods.	Concrete classes contain concrete method i.e. with code and functionality. Concrete class can not contain abstract method.
5.	Abstract classes need to be extended in order to make them useful.	Concrete classes may or may not be extended.
6.	Abstract classes always act as a parent class.	Concrete class can be parent or child or neither of the two.

Ex. 2.12.1 : Create an abstract base class shape with two members base and height, a member function for initialization and a pure virtual function to compute area().

Derive two specific classes triangle and rectangle which override the function area(). Use these classes in a main function and display the area of a triangle and a rectangle.

Sol. :

Test.java

```
abstract class shape
```

```
{
```

```
    int base,height;
```

```
    double a;
```

```
    void initFun()
```

```
{
```

```
    base=5;height=6;
```

```
}
```

```
    abstract void compute_area(); //pure virtual function
```

```
}
```

```
class triangle extends shape
```

```
{
```

```
    public void compute_area()
```

```
{
```

```
    a=(base*height)/2;
```

```
    System.out.println("\n Area of triangle is "+a);
```

```
}
```

```
}
```

```
class Rectangle extends shape
```

```
{
```

```
    public void compute_area()
```

```
{
```

```
    a=(base*height);
```

```
    System.out.println("\n Area of rectangle is "+a);
```

```
}
```

```
}
```

```
public class Test
```

```
{
```

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

```
{
```

```
    triangle obj1=new triangle();
```

```
    obj1.initFun();
```

```
    obj1.compute_area();
```

```
    Rectangle obj2=new Rectangle();
```

```

    obj2.initFun();
    obj2.compute_area();

}

}

Output

```

D:\test>javac Test.java

D:\test>java Test

Area of triangle is 15.0

Area of rectangle is 30.0

Ex. 2.12.2 : What are the conditions to be satisfied while declaring abstract classes ?

Sol. : Following are the conditions that must be satisfied while declaring the abstract classes -

1. The class name must be preceded by the keyword **abstract class**.
2. The abstract class must have one abstract method. This abstract method must also be preceded by the keyword **abstract** and it should not have definition body.

Ex. 2.12.3 : Can an abstract class in Java be instantiated ? Give the reason.

Sol. : The abstract class can not be instantiated (i.e. we can not create the object of this class using new operator) because the abstract class is very much general and less specific. It does nothing and simply lists out only common features of other classes.

Ex. 2.12.4 : Write a program to define abstract class, with two methods addition() and subtraction(), addition() is abstract method. Implement the abstract method and call that method using a program(s).

Sol. :

Java Program[AbstractDemo.java]

```

abstract class A
{
int a=10;
int b=20;
int c;
abstract void addition(int x,int y);
void subtraction()
{
c=b-a;
System.out.println("Subtraction of 20 and 10: "+c);
}
class B extends A
{
void addition(int x,int y)

```

```

{
int z=x+y;
System.out.println("addtion function in class B: "+z);
}
}

class C extends A
{
void addition(int x,int y)
{
int z=x+y;
System.out.println("addtion function in class C: "+z);
}
}

public class AbstractDemo
{
public static void main(String[] args)
{
B b=new B();
C c=new C();
System.out.println("-----");
b.addition(10,20);
b.subtraction();
System.out.println("-----");
c.addition(100,200);
c.subtraction();
System.out.println("-----");
}
}

```

Output

addtion function in class B: 30
Subtraction of 20 and 10: 10

addtion function in class C: 300
Subtraction of 20 and 10: 10

Review Questions

1. Write briefly on abstract classes with an example.
2. Write in detail about : Abstract classes.
3. What does it mean that a method or class is abstract ? Can we make an instance of an abstract class ? Explain it with example.
4. What is abstract class ? Write a program for abstract example.
5. Explain the concept of abstract class with example.
6. What is an abstract class ? Illustrate with an example to demonstrate abstract class.

AU : CSE : Dec.-10, Marks 6

AU : CSE : Dec.-12, Marks 8

AU : IT : May-13, Marks 8

AU : CSE: May-13, Marks 8

AU : CSE : Dec.-13, Marks 8

AU : Dec.-19, Marks 8

AU : Dec.-13, Marks 8

2.13 The Final with Inheritance

The final keyword can be applied at three places -

- For declaring variables
- For declaring the methods
- For declaring the class

2.13.1 Final Variables and Methods

A variable can be declared as final. If a particular variable is declared as final then it cannot be modified further. The final variable is always a constant. For example -

```
final int a=10;
```

The final keyword can also be applied to the method. When final keyword is applied to the method, the method overriding is avoided. That means the methods those are declared with the keyword final cannot be overridden.

Consider the following Java program which makes use of the keyword final for declaring the method -

```
class Test
{
    final void fun()
    {
        System.out.println("\n Hello, this function declared using final");
    }
}

class Test1 extends Test
{
    final void fun()
    {
        System.out.println("\n Hello, this another function");
    }
}
```

Output

```
D:\>javac Test.java
Test.java:10: fun() in Test1 cannot override fun() in Test; overridden method is final
    final void fun()
                           ^
1 error
```

Program Explanation

The above program, on execution shows the error. Because the method fun is declared with the keyword final and it cannot be overridden in derived class.

2.13.2 Final Classes to Stop Inheritance

If we declare particular class as final, no class can be derived from it. Following Java program is an example of final classes.

```
final class Test
{
    void fun()
    {
        System.out.println("\n Hello, this function in base class");
    }
}

class Test1 extends Test
{
    final void fun()
    {
        System.out.println("\n Hello, this another function");
    }
}
```

Output

```
D:\>javac Test.java
Test.java:8: cannot inherit from final Test
class Test1 extends Test
^
1 error
```

Review Question

- Write a note on final keyword.

AU : Dec.-13, Marks 8

2.14 The finalize() method

AU : Dec.-18, Marks 7

- Java has a facility of automatic garbage collection. Hence even though we allocate the memory and then forget to deallocate it then the objects that are no longer used get freed.
- Inside the `finalize()` method you will specify those actions that must be performed before an object is destroyed.
- The garbage collector runs periodically checking for objects that are no longer referenced by any running state or indirectly through other referenced objects.
- To add finalizer to a class simply define the `finalize` method. The syntax to `finalize()` the code is -

```
void finalize()
{
    finalization code
}
```

Note that `finalize()` method is called just before the garbage collection. It is not called when an object goes out-of-scope.

- The finalize method of an object is called when garbage collector is about to clean up the object.
- The purpose of finalization is to perform some action before the objects get cleaned up.
- The clean up code can be kept in the finalize method block.

Review Question

1. State the purpose of finalize () method in java. With an example explain how finalize () method can be used in java program.

AU : Dec.-18, Marks 7**2.15 Packages****AU : IT : May-12, 13, 14, Dec.-11, 12, CSE : Dec.-13, Marks 16****2.15.1 Defining Package**

- Purpose :** In Java, packages are used to achieve the **code reusability**. That means, the classes from other programs can be easily used by a class without physically copying it to current location.
- Definition :** Package is a mechanism in which variety of classes and interfaces can be grouped together.
- Importance :** Following are the benefits of organizing classes into packages-
 - The classes defined in the packages of other program can be easily reused.
 - Two classes from two different packages can have the **same name**. By using the package name the particular class can be referred.
 - Packages provide the complete **separation** between the two phases- **design and coding**.
In the design phase, we can design the classes and decide their relationship and then during the coding phase we can develop the Java code for corresponding classes and can group them in several packages according to their relationship with each other.
 - Using packages it is possible to **hide the classes**. This feature prevents other programs to access the classes that are developed for internal purpose only.
- Creating a package is very simple. Just include the command package at the beginning of the program. The syntax for declaring the package is

```
package name_of_package
```

- This package statement defines the name space in which the classes are stored. If we omit the package then the default classes are put in the package that has no name.
- Basically Java creates a directory and the name of this directory becomes the package name.

For example - In your program, if you declare the package as -

```
package My_Package;
```

then we must create the directory name **My_Package** in the current working directory and the required classes must be stored in that directory. Note that the name of the package and the name of the directory must be the same. This name is case sensitive.

- We can create hierarchy of packages. For instance if you save the required class files in the subfolder **MyPkg3** and the path for this subfolder is **C:\MyPkg1\MyPkg2\MyPkg3** then the declaration for the package in your java program will be -

```
package MyPkg1.MyPkg2.MyPkg3;
```

2.15.2 Creating and Accessing Package

In this section we discuss how to develop a program which makes use of the classes from other package.

Step 1 : Create a folder named **My_Package**.

Step 2 : Create one class which contains two methods. We will store this class in a file named **A.java**. This file will be stored in a folder **My_Package**. The code for this class will be as follows-

Java Program[A.java]

```
package My_Package; //include this package at the beginning
public class A
{
    int a;
    public void set_val(int n)
    {
        a=n;
    }
    public void display()
    {
        System.out.println("The value of a is: "+a);
    }
}
```

Note that the class, and the methods defined must be **public**

Note that this class contains two methods namely- **set_val** and **display**. By the **set_val** method we can assign some value to a variable. The **display** function displays this stored value.

Step 3 : Now we will write another java program named **PackageDemo.java**. This program will use the methods defined in class **A**. This source file is also stored in the subdirectory **My_Package**. The java code for this file is -

Java Program[PackageDemo.java]

```
import My_Package.A; //The java class A is referenced here by import statement
```

```

class PackageDemo
{
    public static void main(String args[]) throws NoClassDefFoundError
    {
        A obj=new A(); //creating an object of class A
        obj.set_val(10); //Using the object of class A, the methods present
        obj.display(); //in class A are accessed
    }
}

```

Step 4 : Now, open the command prompt and issue the following commands in order to run the package programs

D:\>set CLASSPATH=.;D:\;

Setting the class path

D:\>cd My_Package

D:\My_Package>javac A.java

Creating the classes A.class and
PackageDemo.class files

D:\My_Package>javac PackageDemo.java

D:\My_Package>java PackageDemo

The value of a is: 10

Running the test program which uses the
class A.class stored in another file

D:\My_Package>

2.15.3 CLASSPATH

The packages are nothing but the directories. For locating the specified package the java run time system makes use of current working directory as its starting point. Thus if the required packages is in the current working directory then it will found. Otherwise you can specify the directory path setting the **CLASSPATH** statement. For instance- if the package name My_Package is present at prompt D:\> then we can specify

set CLASSPATH=.;D:\;

D:\>cd My_Package

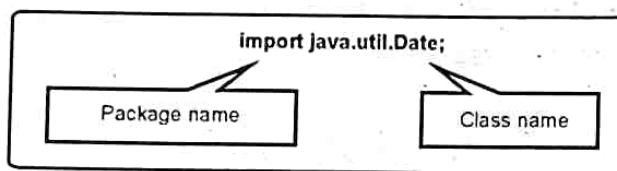
D:\My_Package\> Now you can execute the required class files from this location

2.15.4 Importing Package

- All the standard classes in Java are stored in named packages.
- There is no standard class present in Java which is unnamed. But it is always complicated to write the class name using a long sequence of packages containing dot operator. Hence the **import** statement is needed.
- The import statement can be written at the beginning of the Java program, using the keyword **import**.

- There are two ways of accessing the classes stored in the core package.

1. Method 1 : We import the java package class using the keyword **import**. Suppose we want to use the Data class stored in the **java.util** package then we can write the import statement at the beginning of the program. It is as follows -



2. Method 2 : There are some situations in which we want to make use of several classes stored in a package. Then we can write it as

import java.util.*;

Here * means any class in the corresponding package.

Ex. 2.15.1 : Write a java program to maintain the books details like BookId, accession number; book name, author, publication in books package and keep the journal details such as journal Id; journal name; in journal package in main class use these two packages details for staff and student classes and display the books and journals information as requested by the user.

AU : IT : May-13, Marks 16

Sol. :

Step 1 : Create a folder named MyLibrary and save following two Java programs namely **books.java** and **journals.java** within it.

books.java

```

package MyLibrary;
import java.io.*;

public class books
{
    int BookId,AccessionNumber;
    String BookName, Author, Publication;
    DataInputStream input=new DataInputStream(System.in);
    public void ReadData()
    {
        try
        {
            System.out.println("Enter BookId: ");
            BookId=Integer.parseInt(input.readLine());
            System.out.println("Enter Accession Number: ");
            AccessionNumber=Integer.parseInt(input.readLine());
        }
    }
}
  
```

```

        System.out.println("Enter Book Name: ");
        BookName=input.readLine();

        System.out.println("Enter Author Name: ");
        Author=input.readLine();

        System.out.println("Enter Publication: ");
        Publication=input.readLine();
    }
    catch(Exception e)
    {
        System.out.println("You have entered wrong data!!");
    }
}
public void Display()
{
    System.out.println("BookId: "+BookId);
    System.out.println("Accession Number: "+AccessionNumber);
    System.out.println("Book Name: "+BookName);
    System.out.println("Author Name: "+Author);
    System.out.println("Publication: "+Publication);
}
}

```

journals.java

```

package MyLibrary;
import java.io.*;

public class journals
{
    int JournalId;
    String JournalName;
    DataInputStream input=new DataInputStream(System.in);
    public void ReadData()
    {
        try
        {
            System.out.println("Enter Journal Id: ");
            JournalId=Integer.parseInt(input.readLine());

            System.out.println("Enter Journal Name: ");
            JournalName=input.readLine();
        }
        catch(Exception e)
        {

```

```

        System.out.println("You have entered wrong data!!!");
    }
}

public void Display()
{
    System.out.println("Journal Id: "+JournalId);
    System.out.println("Journal Name: "+JournalName);
}
}

```

Step 2 : Compile the above two programs using following javac command

```

D:\MyLibrary>javac books.java
D:\MyLibrary>javac journals.java

```

Due to above commands the **books.class** and **journals.java** get generated within the folder MyLibrary/

Step 3 : Come out of MyLibrary directory and create following program -

MainClass.java

```

import MyLibrary.*;
import java.io.*;

class Staff {
    int StaffId;
    String StaffName;
    String Department;
    DataInputStream input=new DataInputStream(System.in);
    void ReadData()
    {
        try
        {
            System.out.println("Enter Staff Id");
            StaffId= Integer.parseInt(input.readLine());
            System.out.println("Enter Staff Name");
            StaffName= input.readLine();
            System.out.println("Enter Department of Staff");
            Department= input.readLine();
        }
        catch(Exception e)
        {
            System.out.println("You have entered wrong data!!!");
        }
    }
    void Display()
    {
        System.out.println("Staff Id: "+StaffId);
    }
}

```

```

        System.out.println("Staff Name: " + StaffName);
        System.out.println("Department: " + Department);
    }

}

class Student {
    int RollNumber;
    String StudentName;
    DataInputStream input=new DataInputStream(System.in);
    void ReadData()
    {
        try
        {
            System.out.println("Enter Student Roll Number");
            RollNumber= Integer.parseInt(input.readLine());
            System.out.println("Enter Student Name");
            StudentName= input.readLine();
        }
        catch(Exception e)
        {
            System.out.println("You have entered wrong data!!!");
        }
    }
    void Display()
    {
        System.out.println("Student Roll Number: "+RollNumber);
        System.out.println("Student Name: "+StudentName);
    }
}

public class MainClass
{
    public static void main(String[] args) throws IOException
    {
        Staff Staff_obj=new Staff();
        Student Student_obj=new Student();
        books Book_obj=new books();
        journals journals_obj=new journals();
        int choice;
        char ans;
        DataInputStream input=new DataInputStream(System.in);
        do
        {
            System.out.println("Enter your choice: ");
            System.out.print("\n 1.Staff \n 2.Student");
            choice= Integer.parseInt(input.readLine());
            switch (choice)

```

```

    {
        case 1: System.out.println("\n\t Enter The data for Staff..");
        Staff_obj.ReadData();
        Book_obj.ReadData();
        journals_obj.ReadData();
        System.out.println("\n\t Displaying Record..");
        Staff_obj.Display();
        Book_obj.Display();
        journals_obj.Display();

        break;
    case 2: System.out.println("\n\t Enter The data for Student..");
        Student_obj.ReadData();
        Book_obj.ReadData();
        journals_obj.ReadData();
        System.out.println("\n\t Displaying Record..");
        Student_obj.Display();
        Book_obj.Display();
        journals_obj.Display();
        break;
    }
}

System.out.println("\n Do you want to continue?");
String s1 = input.readLine();
ans = s1.charAt(0);
} while (ans == 'y');
}
}

```

Step 3 : Execute above program using the following command

D:\>javac MainClass.java

D:\>java MainClass

Enter your choice:

1.Staff

2.Student

1

Enter The data for Staff...

Enter Staff Id

1

Enter Staff Name

Archana

Enter Department of Staff

Computer

Enter BookId:-

101
Enter Accession Number:

1010
Enter Book Name:

Software Engineering

Enter Author Name:

Puntambekar

Enter Publication:

Technical

Enter Journal Id:

5001

Enter Journal Name:

IEEE

Displaying Record...

Staff Id: 1

Staff Name: Archana

Department: Computer

BookId: 101

Accession Number: 1010

Book Name: Software Engineering

Author Name: Puntambekar

Publication: Technical

Journal Id: 5001

Journal Name: IEEE

Do you want to continue?

n

Review Questions

1. Give a brief overview of java packages. Write necessary code snippets.

AU : IT : Dec.-11, 12, Marks 16

2. What are packages ? Explain how will you import a package in Java. Give example.

AU : IT : May-12, IT : Dec.-12, Marks 16

3. What is meant by package ? How it is created and implemented in JAVA.

AU : CSE : Dec.-13, Marks 8

4. With suitable examples explain how packages can be created, imported and used. Also elaborate on its scope.

AU : May-14, Marks 16

2.16 Interfaces

AU : May-19, Marks 6

- The interface can be defined using following syntax

access_modifier **interface** name_of_interface

{

return_type method_name1(parameter1,parameter2,...parametern);

...

```

return_type method_name1(parameter1,parameter2,...parametern);
type static final variable_name = value;
...
}

```

- The access_modifier specifies whether the interface is public or not. If the access specifier is not specified for an interface then that interface will be accessible to all the classes present in that package only. But if the interface is declared as public then it will be accessible to any of the class.
- The methods declared within the interface have no body. It is expected that these methods must be defined within the class definition.

2.16.1 Difference between Class and Interface

An interface is similar to a class but there lies some difference between the two.

Class	Interface
The class is denoted by a keyword class	The interface is denoted by a keyword interface
The class contains data members and methods. But the methods are defined in class implementation. Thus class contains an executable code.	The interfaces may contain data members and methods but the methods are not defined. The interface serves as an outline for the class.
By creating an instance of a class the class members can be accessed.	You can not create an instance of an interface.
The class can use various access specifiers like public , private or protected .	The interface makes use of only public access specifier.
The members of a class can be constant or final .	The members of interfaces are always declared as final .

2.16.2 Difference between Abstract Class and Interface

Following table presents the difference between the abstract class and interface.

Sr. No.	Abstract class	Interface
1.	The class can inherit only one abstract class	The class can implement more than one interfaces.
2.	Members of abstract class can have any access modifier such as public , private and protected .	Members of interface are public by default.
3.	The methods in abstract class may or may not have implementation.	The methods in interface have no implementation at all. Only declaration of the methods is given.

4.	Java abstract classes are comparatively efficient.	The interfaces are comparatively slow and implies extra level of indirection.
5.	Java abstract class is extended using the keyword extends .	Java interface can be implemented by using the keyword implements .
6.	The member variables of abstract class can be non final.	The member variables of interface are by default final.

Review Question

1. Present a detailed comparison between classes and interfaces.

AU : May-19 , Marks 6

2.17 Implementing Interface

AU : Dec.-14, Marks 8

- It is necessary to create a class for every interface.
- The class must be defined in the following form while using the interface

```
class Class_name extends superclass_name
implements interface_name1,interface_name2,...
```

```
{
    //body of class
}
```

- Let us learn how to use interface for a class

Step 1 : Write following code and save it as my_interface.java

```
public interface my_interface
{
    void my_method(int val);
}
```

Do not compile this program. Simply save it.

Step 2 : Write following code in another file and save it using InterfaceDemo.java

Java Program[InterfaceDemo.java]

```
class A implements my_interface
{
    public void my_method(int i)
    {
        System.out.println("\n The value in the class A: "+i);
    }
    public void another_method() //Defining another method not declared in interface
    {
        System.out.println("\nThis is another method in class A");
    }
}
class InterfaceDemo
{
```

Defining the method declared in the interface

```

public static void main(String args[])
{
    my_interface obj=new A();
    //or A obj=new A() is also allowed
    A obj1=new A();
    obj.my_method(100);
    obj1.another_method();
}
}

```

Object of class A is of type my_interface. Using this object method can be accessed

Step 3 : Compile the program created in step 2 and get the following output

F:\test>javac InterfaceDemo.java
F:\test>java InterfaceDemo

The value in the class A: 100

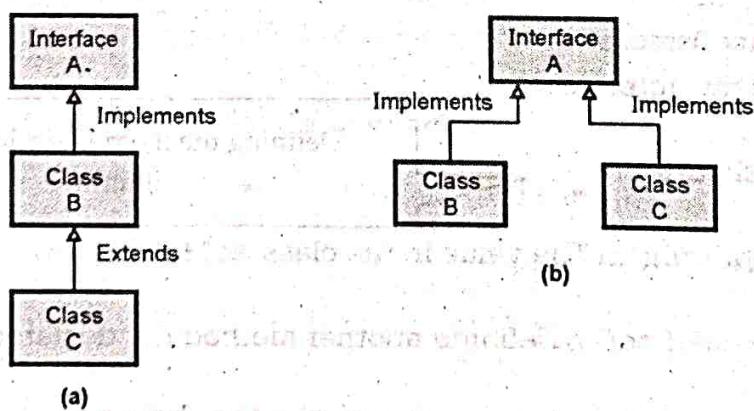
This is another method in class A

Program Explanation :

In above program, the interface my_interface declares only one method i.e. my_method. This method can be defined by class A. There is another method which is defined in class A and that is another_method. Note that this method is not declared in the interface my_interface. That means, a class can define any additional method which is not declared in the interface.

Various ways of interface implementation

- The design various ways by which the interface can be implemented by the class is represented by following Fig. 2.17.1.



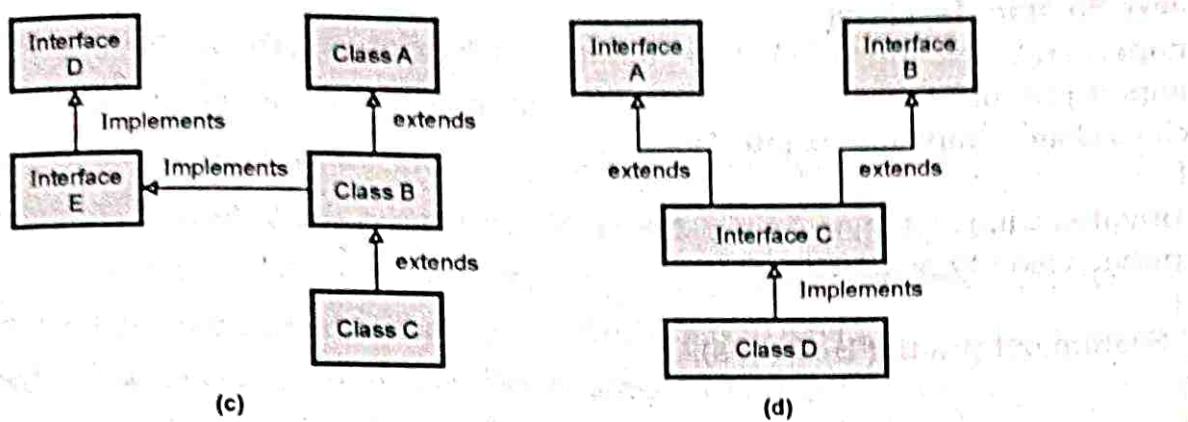


Fig. 2.17.1 Various ways of Interface implementation

- The variables can be assigned with some values within the interface. They are implicitly **final** and **static**. Even if you do not specify the variables as **final** and **static** they are assumed to be **final** and **static**.
- Interface variables are **static** because Java interfaces cannot be instantiated. Hence the value of the variable must be assigned in a static context in which no instance exists. The **final** modifier indicates that the value assigned to the interface variable is a true constant that cannot be re-assigned by program code.

Review Question

- What is an interface ? Write a Java program to illustrate the use of interface. Give self explanatory comments in your program.*

AU : Dec.-14, Marks 8**2.18 Applying Interfaces****AU : May-13, 19, Marks 16**

The interface is a powerful tool. The same interface can be used by different classes for some method. Then this method can be implemented by each class in its own way. Thus same interface can provide variety of implementation. The selection of different implementations is done at the run time. Following is a simple Java program which illustrates this idea.

Step 1 : Write a simple interface as follows

Java Program [interface1.java]

```
interface interface1
{
    void MyMsg(String s);
}
```

Step 2 : Write a simple Java Program having two classes having their own implementation of MyMsg method.

Java Program[Test.java]

```

import java.io.*;
import java.util.*;
class Class1 implements interface1
{
    private String s;
    public void MyMsg(String s)
    {
        System.out.println("Hello "+s);
    }
}
class Class2 implements interface1
{
    private String s;
    public void MyMsg(String s)
    {
        System.out.println("Hello "+s);
    }
}
class Test
{
    public static void main(String[] args)
    {
        interface1 inter;
        Class1 obj1=new Class1();
        Class2 obj2=new Class2();
        inter=obj1;
        inter.MyMsg("User1");
        inter=obj2;
        inter.MyMsg("User2");
    }
}

```

Step 3 : Execute the program.



Program Explanation :

- The selection of method **MyMsg** is done at the runtime and it depends upon the object which is assigned to the reference of the interface.
- We have created reference **Inter** for the interface in which the method **MyMsg** is declared.
- Then the objects **obj1** and **obj2** are created for the classes **class1** and **class2** respectively. When the **obj1** is assigned to the reference **Inter** then the message "Hello User1" will be displayed because the string passed to the method **MyMsg** is "User1". Similarly, when the **obj2** is assigned to the reference **Inter** then the message "Hello User2" will be displayed because the string passed to the method **MyMsg** is "User2".
- Thus using the same interface different implementations can be selected.

Ex. 2.18.1 : Write a program to create Interface method named customer. In this keep the methods called information(), show() and also maintain the tax rate. Implement this interface in employee class and calculate the tax of the employee based on their income.

Income	Tax percentage	
	Male	Female
>=1,90,000	Nil	Nil
>=2,00,000	10 %	Nil
>=5,00,000	20 %	10 %
<5,00,000	25 %	20 %

AU : May-13, Marks 16

Sol. :

Step 1 : Create an interface file named Customer.java. It is as follows -

```
interface Customer { }
```

```
double rate1=0.10;
double rate2=0.20;
double rate3=0.25;
double income=0.0;
void information();
void show();
```

Step 2 : Create a Java program named Employee.java. It is as follows -

Employee.java

```

import java.lang.System;
import java.util.Scanner;
class TestClass implements Customer
{
    private char sex;
    private double tax=0;
    private double netpay;
    private double income=0.0;
    public void information()
    {
        Scanner in=new Scanner(System.in);
        System.out.print("\n Enter Sex(M or F)");
        sex = in.next().charAt(0);
        System.out.println("\n Enter your income");
        income=in.nextDouble();
        if(sex=='M')
        {
            if((income>=190000)&&(income<200000))
            {
                netpay=income;
            }
            else if((income>=200000)&&(income<500000))
            {
                netpay=income-(income*rate1);
            }
            else if((income>=500000)&&(income<1000000))
            {
                netpay=income-(income*rate2);
            }
            else if(income>1000000)
                netpay=income-(income*rate3);
        }
        else
        {
            if((income>=190000)&&(income<200000))
            {
                netpay=income;
            }
            else if((income>=200000)&&(income<500000))
            {
                netpay=income;
            }
            else if((income>=500000)&&(income<1000000))
            {

```

```

        netpay=income-(income*rate1);
    }
    else if(income>=1000000)
        netpay=income-(income*rate2);
    }

}

public void show()
{
    System.out.print("\n The net Income of the person is\n"+netpay);
}
}

class Employee
{
    public static void main(String[] args)
    {
        Customer cust;
        TestClass obj=new TestClass();
        cust=obj;
        cust.information();

        cust.show();
    }
}

```

Step 3 : Open the command prompt and issue the following commands to execute the above program

D:\>javac Employee.java
D:\>java Employee

Enter Sex(M or F)F

Enter your income

500000

The net Income of the person is

450000.0

Ex. 2.18.2 : Define an interface using JAVA that contains a method to calculate the perimeter of object. Define two classes - circle and rectangle with suitable fields and methods. Implement the interface "perimeter" in these classes. Write the appropriate main() method to create object of each class and test all methods.

Sol. :

Step 1 : We will write the simple interface. The name of this interface is **perimeter**. In this interface the method **calculate** is declared.

```
perimeter.java
interface perimeter
{
    void calculate();
}
```

Step 2 : Following is a java program in which two classes are defined- circle and rectangle.

The perimeter for each of this class is calculated using the function declared in the interface.

Main.java

```
import java.io.*;
import java.util.*;
import java.util.Scanner;
class circle implements perimeter
{
    private double r;
    public void calculate()
    {
        System.out.print("Enter radius:");
        Scanner input=new Scanner(System.in);
        double r=input.nextDouble();
        double p=2*3.14*r;
        System.out.println("Perimeter of Circle: "+p);
    }
}
class rectangle implements perimeter
{
    private double length,breadth;
    public void calculate()
    {
        System.out.print("Enter length:");
        Scanner input=new Scanner(System.in);
        double length=input.nextDouble();
        System.out.print("Enter breadth:");
        double breadth=input.nextDouble();
        double p=2*(length+breadth);
        System.out.println("Perimeter of rectangle: "+p);
    }
}
class Main
{
    public static void main(String[] args)
    {
        perimeter obj;
        circle obj1=new circle();
```

```

rectangle obj2=new rectangle();
System.out.println("\n\tCalculating Perimeter of Circle....\n");
obj=obj1;
obj.calculate();

System.out.println("\n\t Calculating Perimeter of Rectangle....\n");
obj=obj2;
obj.calculate();
}
}

```

Step 3 : The output for this program will be as follows

```

D:\>javac perimeter.java
D:\>javac Main.java
D:\>java Main
Calculating Perimeter of Circle....
Enter radius:100
Perimeter of Circle: 628.0
Calculating Perimeter of Rectangle....
Enter length:20
Enter breadth:30
Perimeter of rectangle: 100.0
D:\>

```

Ex. 2.18.3 : Write an interface called **shape** with methods **draw()** and **getArea()**. Further design two classes called **Circle** and **Rectangle** that implements **Shape** to compute area of respective shapes. Use appropriate getter and setter methods. Write a Java program for the same.

Sol. :

```

interface Shape
{
    void draw();
    double getArea();
}

class Circle implements Shape
{
    private double radius;
    public void setRadius(double r)
    {
        radius=r;
    }
}

```

```
        }
        public double getRadius()
        {
            return radius;
        }
        public double getArea()
        {
            return (3.14*radius*radius);
        }
        public void draw()
        {
            System.out.println("Area of Circle is :" +getArea());
        }
    }
    class Rectangle implements Shape
    {
        private double length,breadth;
        public void setLength(double l)
        {
            length=l;
        }
        public double getLength()
        {
            return length;
        }
        public void setBreadth(double b)
        {
            breadth=b;
        }
        public double getBreadth()
        {
            return breadth;
        }
        public double getArea()
        {
            return (length*breadth);
        }
        public void draw()
        {
            System.out.println("Area of Rectangle is :" +getArea());
        }
    }
    class Test
    {
        public static void main(String[] args)
```

```

    {
        Circle c=new Circle();
        c.setRadius(10);
        System.out.println("The Radius of Circle is "+c.getRadius());
        c.draw();
        Rectangle r=new Rectangle();
        r.setLength(10);
        r.setBreadth(20);
        System.out.println("Length = "+r.getLength()+" , Breadth = "+r.getBreadth());
        r.draw();
    }
}

```

Output

The Radius of Circle is 10.0

Area of Circle is :314.0

Length = 10.0, Breadth = 20.0

Area of Rectangle is :200.0

2.18.1 Nested Interface

Nested interface is an interface which is declared within another interface. For example -

```

interface MyInterface1 {
    interface MyInterface2 {
        void show();
    }
}

```

Nested Interface with declaration of one method.

```
class NestedInterfaceDemo implements MyInterface1.MyInterface2 {
```

```
    public void show() {
```

```
        System.out.println("Inside Nested interface method");
```

```
}
```

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

```
    MyInterface1.MyInterface2 obj= new NestedInterfaceDemo();
```

```
    obj.show();
```

Defining nested interface method

Output

Inside Nested interface method

Ex. 2.18.4 : The transport interface declares a deliver() method. The abstract class Animal is the super class of Tiger, Camel, Deer and Donkey classes. The transport interface is implemented by the Camel and Donkey classes. Write a test program to

initialize an array of four animal objects. If the object implements the transport interface, the deliver() method is invoked.

Sol. :

```

interface Transport
{
    public void deliver();
}

abstract class Animal
{
    String item;
    abstract void deliver();
}

class Tiger extends Animal
{
    public void deliver(){
    }
}

class Camel extends Animal implements Transport
{
    Camel(String s)
    {
        item=s;
    }
    public void deliver()
    {
        System.out.println("\tCamels are used in deserts to carry "+item+".");
    }
}

class Deer extends Animal
{
    public void deliver(){}
}

class Donkey extends Animal implements Transport
{
    Donkey(String s){
        item=s;
    }
    public void deliver()
    {
        System.out.println("\tDonkeys can carry "+item+".");
    }
}

class testclass
{
    public static void main(String args[]) throws NullPointerException
    {
        Transport[] t=new Transport[4];
    }
}

```

```
t[1]=new Camel("goods and people");
t[2]=new Donkey("heavy load");
System.out.println("Transport interface is implemented by following animals...");
t[1].deliver();
t[2].deliver();
}
```

Output

Transport interface is implemented by following animal...
Camels are used in deserts to carry goods and people.
Donkeys can carry heavy load.

Review Question

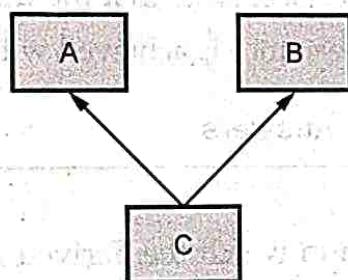
- Explain simple interfaces and nested interfaces with examples.

AU : May-19, Marks 7

2.19 Multiple Inheritance

Definition: Multiple inheritance is a mechanism in which the child class inherits the properties from more than one parent classes.

For example



- Java does not support multiple inheritance because it creates ambiguity when the properties from both the parent classes are inherited in child class or derived class. But it is supported in case of interface because there is no ambiguity as implementation is provided by the implementation class.

- Implementation

interface A

{

 void display1();

}

interface B

{

 void display2();

}

class C implements A,B

{

 public void display1()

{

```

        System.out.println("From Interface A");
    }
    public void display2()
    {
        System.out.println("From Interface B");
    }
    public static void main(String args[])
    {
        C obj = new C();
        obj.display1();
        obj.display2();
    }
}

```

Output

From Interface A

From Interface B

Program Explanation :

In above program, we have created both the parent interfaces and the derived class is using method declared ion the parent interface. Note that the definition of these methods is present in the child class. Thus multiple inheritance is achieved with the help of interface.

2.20 Two Marks Questions with Answers

Q.1 Define the term inheritance.

Ans. : Inheritance is a mechanism in which the derived class borrows the properties of the base class.

Q.2 Explain the use of extend keyword with suitable example.

Ans. :

- The extend keyword is used in inheritance.
- When the child class is derived from its parent class then the keyword extends is used.

Q.3 What is the difference between superclass and subclass ?

Ans. :

- The superclass is a parent class or base class from which another class can be derived.
- The subclass is always a derived class which inherits the properties of the base class or superclass.
- The superclass is normally a generalized class whereas the subclass is normally for specific purpose.

Q.4 Enlist various forms of inheritance.

Ans. : Various forms of inheritance are –

1. Single level inheritance
2. Multi-level inheritance
3. Multiple inheritance
4. Hierarchical inheritance
5. Hybrid inheritance

Q.5 What is the use of super keyword?

Ans.: The super class is used to access immediate parent class from the subclass. It is used to

1. access parent's variable,
2. access parent's method
3. access parent's constructor invocation

Q.6 Differentiate between inheritance and polymorphism.

Ans.:

Sr. No.	Inheritance	Polymorphism
1.	Inheritance is a property in which some of the properties and methods of base class can be derived by the derived class.	Polymorphism is ability for an object to used different forms. The name of the function remains the same but it can perform different tasks.
2.	Various types of inheritance can be single inheritance, multiple inheritance, multilevel inheritance and hybrid inheritance.	Various types of polymorphism are compile time polymorphism and run time polymorphism. In compile time polymorphism there are two types of overloading possible. - Functional overloading and operator overloading. In run time polymorphism there is a use of virtual function.

Q.7 Distinguish between static and dynamic binding.

AU : IT : May-11

Ans.: Static binding is a type of binding in which the function call is resolved at compile time.

The dynamic binding is a type of binding in which the function call is resolved at run time.

The static binding is called the early binding and the dynamic binding is called late binding.

Q.8 What is the purpose of 'final' keyword?

AU : IT : Dec-11, CSE : Dec-12

Ans. : The keyword 'final' is used to avoid further modification of a variable, method or a class. For instance if a variable is declared as final then it can not be modified further, similarly, if a method is declared as final then the method overriding is avoided.

Q.9 Define inheritance hierarchy.

AU : CSE : Dec-11, 18

Ans. : The inheritance hierarchy represents the collection of classes the inherit from their base classes and thereby make use of the code present in the base class. The data reusability can be achieved by inheritance hierarchy. For example

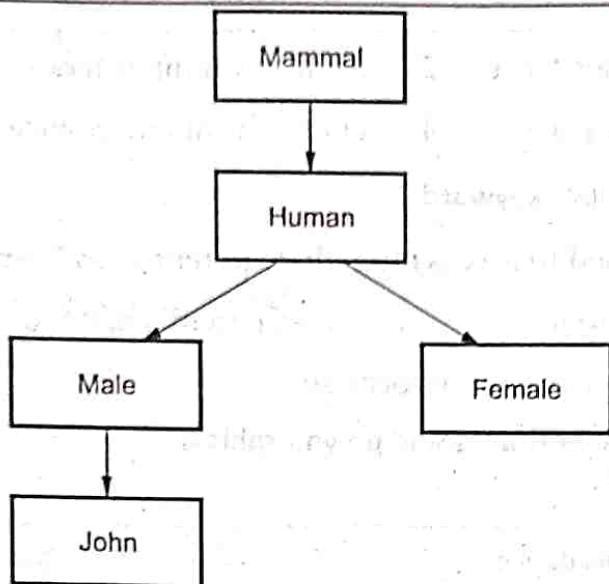


Fig. 2.20.1

Q.10 How to prevent inheritance ?

Ans. : If we declare particular class as final then no class can be derived from it. For example
final class Test

```

{
...
...
...
}
class Test1 extends Test
{
...
...
...
}
  
```

The above code will produce an error "cannot inherit from final Test".

Thus the use of keyword final for the class prevents the inheritance.

**Q.11 Write a class declarations for the following relationship, assuming that all classes are public:
a Bulldog is a kind of dog, and a Dog is a kind of Animal.**

Ans. :

```

class Animal
{
...
}
class dog extends Animal
{
...
}
class Bulldog extends dog
  
```

{
...
}**Q.12 What is meant by dynamic method dispatch ?**

Ans. : The dynamic method dispatch is run time polymorphism in which a call to overridden function is resolved at runtime instead of at compile time. Hence is the name. For example

```
class Test1
{
    public void method1()
    {
    }
}

class Test2 extends Test1
{
    public void method1()
    {
    }

    void main()
    {
        Test1 obj=new Test2();
        obj.method1();
    }
}
```

The **method1** of class **Test2** will be invoked here, thus the code supports the dynamic method dispatch.

Q.13 Can an abstract class be final ? Why ?**AU : IT : Dec.-10**

Ans. : The abstract class cannot be final because once we declared the abstract base class with the keyword final then it can not be inherited.

Q.14 Can an abstract class in Java be instantiated ? Give the reason.**AU : IT : Dec.-13**

Ans. : The abstract class can not be instantiated(i.e. we can not create the object of this class using new operator) because the abstract class is very much general and less specific. It does nothing and simply lists out only common features of other classes.

Q.15 Why is multiple inheritance using classes a disadvantage in Java ?**AU : IT : May-13**

Ans. : In multiple inheritance, the child class is derived from two or more parent classes. It can lead to ambiguity when two base classes implement a method with the same name.

Q.16 State the condition for method overriding in Java**AU : May 19**

Ans. : Method overriding occurs only when the name of the two methods(method in super class and method in subclass) are same and their type signature is same.

Q.17 What is package ?

Ans. : Package represent a collection of classes,methods and interfaces. The name of the package must be written as the first statement in the Java source program. The syntax of specifying the package in the Java program is
package name_of_package

Due to package the systematic arrangement of classes is possible. All standard classes in Java are stored in named packages.

Q.18 Mention the necessity for import statement.

AU : IT : May-12

Ans. : The import statement is used to refer the classes and methods that are present in particular package. This statement is written at the beginning of any Java program. For example -

```
import java.io.*;
```

This statement allows to use the useful functionalities for performing input and output operations.

Q.19 List any four packages in java and highlight their features.

Ans. :

Package	Feature
java.io	This package provide useful functionalities for performing input and output operations through data streams and through file system.
java.net	This package is for providing useful classes and interfaces for networking applications which are used in sockets and URL.
java.lang	This package provides core classes and interfaces used in design of Java language.
java.util	Some commonly used utilities such as random number generation, event model, date, time and some of the system properties are represented using java util package.

Q.20 What is API package ?

Ans. : • The API packages are application programming interface packages used by java.
 • These packages include important classes and interfaces which are used by the programmer while developing the java applications.
 • For example java.io, java.util, java.net and so on are the commonly used API.

Q.21 Explain any three uses of package.

Ans. :

1. The classes defined in the packages of other program can be easily reused.
2. Two classes from two different packages can have the same name. By using the package name the particular class can be referred.
3. Packages provide the complete separation between the two phases- design and coding.

Q.22 Explain the term CLASSPATH.

Ans. : The packages are nothing but the directories. For locating the specified package the java run time system makes use of current working directory as its starting point. This directory path is called CLASSPATH.

Q.23 What are the ways of importing the java packages ?

OR Write the syntax for importing packages in a Java source file and give an example. AU : May-19

Ans. : The import statement can be written at the beginning of the Java program, using the keyword import. For example -

```
import java.io.File  
or  
import java.io.*
```

Either a class name can be specified explicitly or a * can be used which indicated that the Java compiler should import the entire package.

Q.24 In Java what is the use of interface ?

AU : CSE : Dec.-10

OR What is interface mention its use.

AU : IT : Dec.-11,19

Ans. : In java, the interface is used to specify the behaviour of a group of classes. Using interfaces the concept of multiple inheritance can be achieved.

Q.25 Define Interface and write the syntax of the Interface.

AU : CSE : Dec.-12

Ans. : The interface can be defined using following syntax

```
access_modifier interface name_of_interface  
{  
    return_type method_name1(parameter1,parameter2,...parametern);  
    ...  
    return_type method_name1(parameter1,parameter2,...parametern);  
    type static final variable_name=value;  
    ...  
}
```

The interface is a collection of various methods that can be used by the class that is attached to the interface. The interface name must be preceded by the keyword interface.

Q.26 What modifiers may be used with an interface declaration ?

Ans. : An interface may be declared as public or abstract.

Q.27 Why the variables in interface static and final ?

Ans. : The members of interface are static and final because -

- 1) The reason for being static - The members of interface belong to interface only and not object.
- 2) The reason for being final - Any implementation can change value of fields if they are not defined as final. Then these members would become part of the implementation.
An interface is pure specification without any implementation.

Q.28 What is the purpose of nested interface ?

Ans. : The nested interfaces are used to group related interfaces so that they can be easy to maintain. The nested interface must be referred by the outer interface or class. It can't be accessed directly.

Q.29 What are the properties of nested Interface ?

Ans. :

- 1) Nested interfaces are static by default. You don't have to mark them static explicitly.
 - 2) Nested interfaces declared inside class can take any access modifier.
 - 3) Nested interface declared inside interface is public implicitly.

Q.30 If a class B is derived from class A and extends the interface myinterface, then how will you write this statement for class B definition ?

Ans. ;

class B extends class A implements myinterface

```
...//body of class B
```

UNIT III

3

Exception Handling and Multithreading

Syllabus

Exception Handling basics – Multiple catch Clauses – Nested try Statements – Java's Built-in Exceptions – User defined Exception. Multithreaded Programming : Java Thread Model– Creating a Thread and Multiple Threads – Priorities – Synchronization – Inter Thread Communication- Suspending -Resuming, and Stopping Threads –Multithreading. Wrappers – Auto boxing.

Contents

3.1	Exception Handling Basics.....	Dec.-10, 18, May-11,	Marks 16
3.2	try-catch Block		
3.3	Multiple catch Clauses		
3.4	Nested try Statements		
3.5	Using finally		
3.6	Using throws		
3.7	Using throw		
3.8	Java's Built-in Exceptions	May-19,	Marks 6
3.9	User defined Exception.....	Dec.-13,	Marks 8
3.10	Basic Concepts	Dec.-19,	Marks 13
3.11	Java Thread Model	Dec.-10,11,18, May-12,13, .	Marks 13
3.12	Creating a Thread	Dec.-10,11, 13, May-13,	Marks 16
3.13	Multithreading.....	Dec.-11,12,18,19,May-12,19,	Marks 16
3.14	Priorities		
3.15	Synchronization.....	Dec.-14, 19,	Marks 13
3.16	Inter Thread Communication		
3.17	Suspending - Resuming, and Stopping Threads	May-13,	Marks 16
3.18	Wrappers		
3.19	Autoboxing		
3.20	Two Marks Questions with Answers		

Part I : Exception Handling

3.1 Exception Handling Basics

AU : Dec.-10, 18, May-11, Marks 16

- Exception in Java is an indication of some unusual event. Usually it indicates the error. Let us first understand the concept. In Java, exception is handled using five keywords try, catch, throw, throws and finally.
- The Java code that you may think may produce exception is placed within the try block. Let us see one simple program in which the use of try and catch is done in order to handle the exception *divide by zero*.

Java Program [ExceptionDemo.Java]

```
class ExceptionDemo
{
    public static void main(String args[])
    {
        try
        {
            int a,b;
            a=5;
            b=a/0;
        }
        catch(ArithmeticException e)
        {
            System.out.println("Divide by Zero\n");
        }
        System.out.println("...Executed catch statement");
    }
}
```

A try block which includes the lines for which exception must be raised

Exception is caught by catch block. A try-catch pair must exist

Output

Divide by Zero

...Executed catch statement

Inside a try block as soon as the statement: `b=a/0` gets executed then an arithmetic exception must be raised, this exception is caught by a catch block. Thus there must be a try-catch pair and catch block should be immediate follower of try statement. After execution of catch block the control must come on the next line. These are basically the exceptions thrown by java runtime systems.

3.1.1 Benefits of Exception Handling

Following are the benefits of exception handling -

1. Using exception the main application logic can be separated out from the code which may cause some unusual conditions.
2. When calling method encounters some error then the exception can be thrown. This avoids crashing of the entire application abruptly.
3. The working code and the error handling code can be separated out due to exception handling mechanism. Using exception handling, various types of errors in the source code can be grouped together.
4. Due to exception handling mechanism, the errors can be propagated up the method call stack i.e. problems occurring at the lower level can be handled by the higher up methods.

3.1.2 Exception Hierarchy

The exception hierarchy is derived from the base class **Throwable**. The **Throwable** class is further divided into two classes - **Exceptions** and **Errors**.

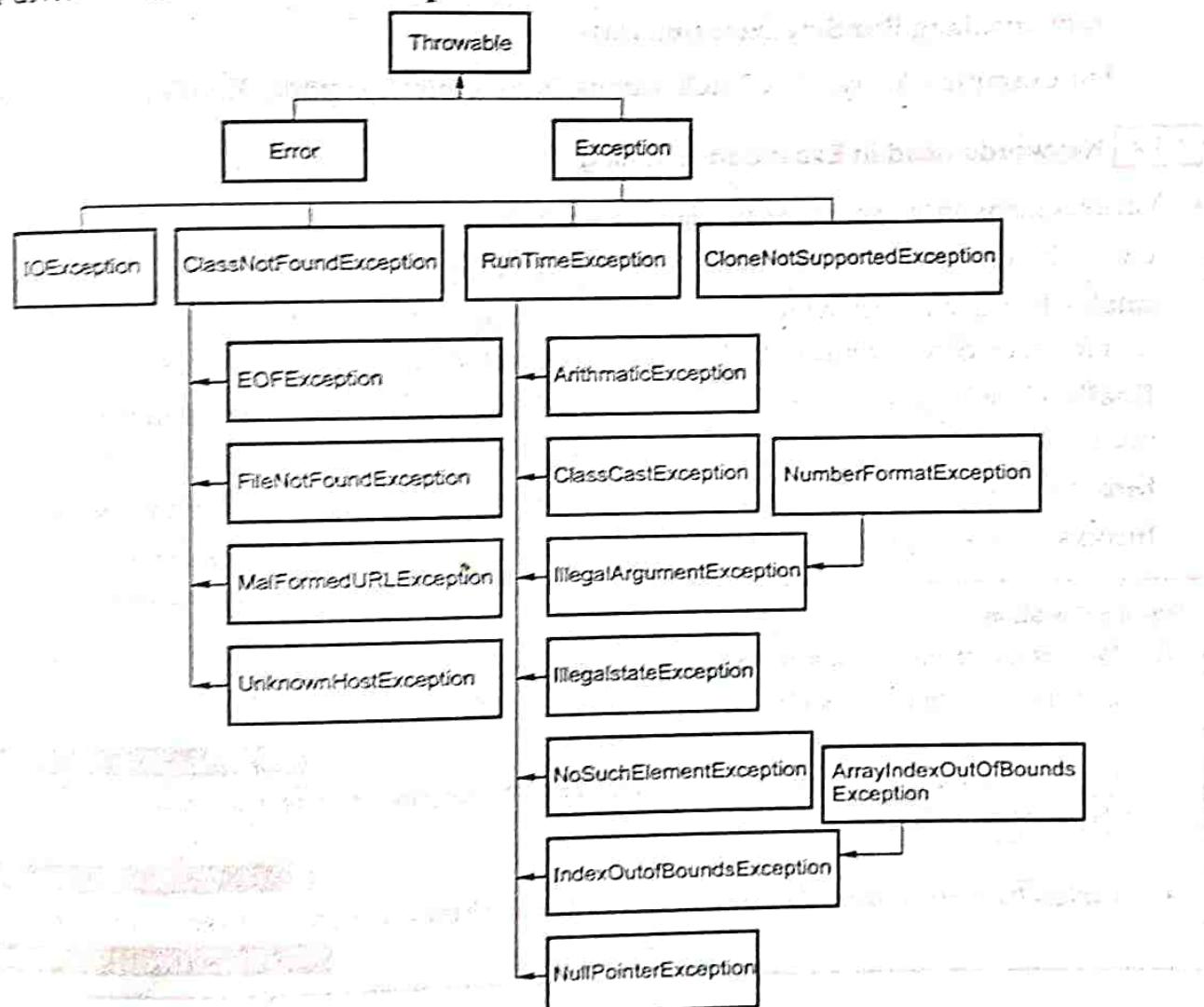


Fig. 3.1.1 Exception hierarchy

Exceptions : Exceptions are thrown if any kind of unusual condition occurs that can be caught. Sometimes it also happens that the exception could not be caught and the program may get terminated.

Errors : When any kind of serious problem occurs which could not be handled easily like **OutOfMemoryError** then an error is thrown.

3.1.3 Types of Exception

- There are two type of exceptions in Java
 - **Checked Exception :** These types of exceptions need to be handled explicitly by the code itself either by using the **try-catch** block or by using **throws**. These exceptions are extended from the **java.lang.Exception** class.

For example : **IOException** which should be handled using the **try-catch** block.

- **UnChecked Exception :** These type of exceptions need not be handled explicitly. The Java Virtual Machine handles these type of exceptions. These exceptions are extended from **java.lang.RuntimeException** class.

For example : **ArrayIndexOutOfBoundsException**, **NullPointerException**, **RunTimeException**.

3.1.4 Keywords used in Exception Handling

- Various keywords used in handling the exception are -
 - try** - A block of source code that is to be monitored for the exception.
 - catch** - The catch block handles the specific type of exception along with the try block. Note that for each corresponding try block there exists the catch block.
 - finally** - It specifies the code that must be executed even though exception may or may not occur.
 - throw** - This keyword is used to throw specific exception from the program code.
 - throws** - It specifies the exceptions that can be thrown by a particular method.

Review Question

1. What is exception ? Explain exception handling in Java.
2. Explain the exception hierarchy.

AU : CSE : Dec.-10, Marks 8

3. Draw the exception hierarchy in java and explain with examples throwing and catching exceptions and the common exception.

AU : IT : May-11, Marks 16

4. Explain the different types of exceptions and the exception hierarchy in java with appropriate examples.

AU : CSE : Dec.-18, Marks 13

3.2 try-catch Block

- The statements that are likely to cause an exception are enclosed within a **try** block. For these statements the exception is thrown.
- There is another block defined by the keyword **catch** which is responsible for handling the exception thrown by the **try** block.
- As soon as exception occurs it is handled by the **catch** block.
- The catch block is added immediately after the try block.
- Following is an example of **try-catch** block

```
try
{
    //exception gets generated here
}
catch(Type_of_Exception e)
{
    //exception is handled here
}
```

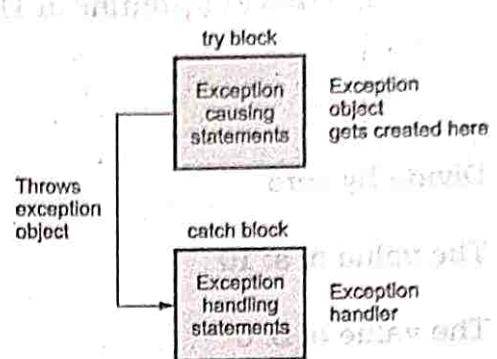
- If any one statement in the **try** block generates exception then remaining statements are skipped and the control is then transferred to the **catch** statement.

Java Program[RunErrDemo.java]

```
class RunErrDemo
{
    public static void main(String[] args)
    {
        int a,b,c;
        a=10;
        b=0;
        try
        {
            c=a/b;
        }
        catch(ArithmeticException e)
        {
            System.out.println("\n Divide by zero");
        }
    }
}
```

Exception occurs because the element is divided by 0.

Exception is handled using catch block



```

System.out.println("\n The value of a: "+a);
System.out.println("\n The value of b: "+b);
}
}

```

Output

Divide by zero

The value of a: 10

The value of b: 0

Note that even if the exception occurs at some point, the program does not stop at that point.

3.3 Multiple catch Clauses

- It is not possible for the try block to throw a single exception always.
- There may be the situations in which different exceptions may get raised by a single try block statements and depending upon the type of exception thrown it must be caught.
- To handle such situation multiple catch blocks may exist for the single try block statements.
- The syntax for single try and multiple catch is -

```

try
{
    ...
    ...//exception occurs
}
catch(Exception_type e)
{
    ...//exception is handled here
}
catch(Exception_type e)
{
    ...//exception is handled here
}
catch(Exception_type e)
{
    ...//exception is handled here
}

```

Example**Java Program[MultipleCatchDemo.java]**

```

class MultipleCatchDemo
{
    public static void main (String args [])
    {
        int a[] = new int [3];
        try
        {
            for (int i = 1; i <=3; i++)
            {
                a[i] = i * i;
            }
            for (int i = 0; i <3; i++)
            {
                a[i] = i/i;
            }
        }
        catch (ArrayIndexOutOfBoundsException e)
        {
            System.out.println ("Array index is out of bounds");
        }
        catch (ArithmaticException e)
        {
            System.out.println ("Divide by zero error");
        }
    }
}

```

Output

Array index is out of bounds

Note : If we comment the first for loop in the try block and then execute the above code we will get following output

Divide by zero error

3.4 Nested try Statements

- When there are chances of occurring multiple exceptions of different types by the same set of statements then such situation can be handled using the nested try statements.
- Following is an example of nested try-catch statements -

Java Program[NestedtryDemo.java]

```
class NestedtryDemo
{
    public static void main (String[] args)
    {
        try
        {
            int a = Integer.parseInt (args [0]);
            int b = Integer.parseInt (args [1]);
            int ans = 0;
            try
            {
                ans = a / b;
                System.out.println("The result is "+ans);
            }
            catch (ArithmeticException e)
            {
                System.out.println("Divide by zero");
            }
            catch (NumberFormatException e)
            {
                System.out.println ("Incorrect type of data");
            }
        }
    }
}
```

Output

D:\>javac NestedtryDemo.java

D:\>java NestedtryDemo 20 10
The result is 2

D:\>java NestedtryDemo 20 a
Incorrect type of data

D:\>java NestedtryDemo 20 0
Divide by zero

Inner
try-catch

Outer
try-catch

Outer catch handles the error

Inner catch handles the error

3.5 Using finally

- Sometimes because of execution of try block the execution gets break off. And due to this some important code (which comes after throwing off an exception) may not get executed. That means, sometimes try block may bring some unwanted things to happen.
- The finally block provides the assurance of execution of some important code that must be executed after the try block.
- Even though there is any exception in the try block the statements assured by finally block are sure to execute. These statements are sometimes called as **clean up code**. The syntax of finally block is

```
finally
{
    //clean up code that has to be executed finally
}
```

- The finally block always executes. The finally block is to free the resources.

Java Program [finallyDemo.java]

```
/*
This is a java program which shows the use of finally block for handling exception
*/
class finallyDemo
{
    public static void main(String args[])
    {
        int a=10,b=-1;
        try
        {
            b=a/0;
        }
        catch(ArithmeticException e)
        {
            System.out.println("In catch block: "+e);
        }
        finally
        {
            if(b!=-1)
                System.out.println("Finally block executes without occurrence of exception");
            else
                System.out.println("Finally block executes on occurrence of exception");
        }
    }
}
```

Output

In catch block: java.lang.ArithmeticException: / by zero

Finally block executes on occurrence of exception

Program Explanation

- In above program, on occurrence of exception in try block the control goes to catch block, the exception of instance *ArithmeticException* gets caught. This is **divide by zero** exception and therefore */ by zero* will be printed as output. Following are the rules for using try, catch and finally block
 1. There should be some preceding try block for catch or finally block. Only catch block or only finally block without preceding try block is not at all possible.
 2. There can be zero or more catch blocks for each try block but there must be single finally block present at the end.

3.6 Using throws

- When a method wants to throw an exception then keyword **throws** is used. The syntax is -


```
method_name(parameter_list) throws exception_list
```
- Let us understand this exception handling mechanism with the help of simple Java program.

Java Program

```
/* This programs shows the exception handling mechanism using throws */
class ExceptionThrows
{
    static void fun(int a,int b) throws ArithmeticException
    {
        int c;
        try
        {
            c=a/b;
        }
        catch(ArithmeticException e)
        {
            System.out.println("Caught exception: "+e);
        }
    }
}
```

```

public static void main(String args[])
{
    int a=5;
    fun(a,0);
}
}

```

Output

Caught exception: java.lang.ArithmaticException: / by zero

- In above program the method *fun* is for handling the exception *divide by zero*. This is an arithmetic exception hence we write

```
static void fun(int a,int b) throws ArithmaticException
```

- This method should be of *static* type. Also note as this method is responsible for handling the exception the *try-catch* block should be within *fun*.

3.7 Using throw

- For explicitly throwing the exception, the keyword *throw* is used. The keyword *throw* is normally used within a method. We can not throw multiple exceptions using *throw*.

Java Programming Example

```

class ExceptionThrow
{
    static void fun(int a,int b)
    {
        int c;
        if(b==0)
            throw new ArithmaticException("Divide By Zero!!!");
        else
            c=a/b;
    }
    public static void main(String args[])
    {
        int a=5;
        fun(a,0);
    }
}

```

Output

Exception in thread "main" java.lang.ArithmaticException: Divide By Zero!!!
at ExceptionThrow.fun(ExceptionThrow.java:7)
at ExceptionThrow.main(ExceptionThrow.java:14)

Difference between throw and throws

Throw	Throws
For explicitly throwing the exception, the keyword throw is used.	For declaring the exception the keyword throws is used.
Throw is followed by instance.	Throws is followed by exception class.
Throw is used within the method.	Throw is used with method signature.
We cannot throw multiple exceptions.	It is possible to declare multiple exceptions using throws.
Example - Refer section 3.7.	Example - Refer section 3.6.

3.8 Java's Built-in Exceptions

AU : May-19, Marks 6

- Various common exception types and causes are enlisted in the following table -

Exception	Description
ArithmaticException	This is caused by error in Math operations. For e.g. Divide by zero.
NullPointerException	Caused when an attempt to access an object with a Null reference is made.
IOException	When an illegal input/output operation is performed then this exception is raised.
IndexOutOfBoundsException	An index when gets out of bound ,this exception will be caused.
ArrayIndexOutOfBoundsException	Array index when gets out of bound, this exception will be caused.
ArrayStoreException	When a wrong object is stored in an array this exception must occur.
EmptyStackException	An attempt to pop the element from empty stack is made then this exception occurs
NumberFormatException	When we try to convert an invalid string to number this exception gets caused.
RuntimeException	To show general run time error this exception must be raised.
Exception	This is the most general type of exception
ClassCastException	This type of exception indicates that one tries to cast an object to a type to which the object can not be casted.
IllegalStateException	This exception shows that a method has been invoked at an illegal or inappropriate time. That means when Java environment or Java application is not in an appropriate state then the method is invoked.

Review Question

1. Write a note on built in exceptions.

AU : May-19, Marks 6

3.9 User defined Exception

AU : Dec.-13, Marks 8

- We can throw our own exceptions using the keyword **throw**.
- The syntax for throwing out own exception is -

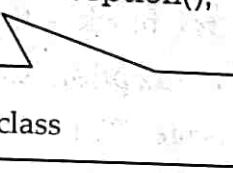
throw new Throwable's subclass

- Here the **Throwable's subclass** is actually a subclass derived from the **Exception class**.

For example -

throw new ArithmeticException();

Throwable's subclass



- Let us see a simple Java program which illustrates this concept.

Java Program[MyExceptDemo.java]

```

import java.lang.Exception;
class MyOwnException extends Exception
{
    MyOwnException(String msg)
    {
        super(msg);
    }
}
class MyExceptDemo
{
    public static void main (String args [])
    {
        int age;
        age=15;
        try
        {
            if(age<21)
                throw new MyOwnException("Your age is very less than the condition");
        }
        catch (MyOwnException e)
        {
            System.out.println ("This is My Exception block");
            System.out.println (e.getMessage());
        }
        finally
    }
}

```

```

    {
        System.out.println ("Finally block:End of the program");
    }
}
}
}

```

Output

This is My Exception block
 Your age is very less than the condition
 Finally block:End of the program

Program Explanation

- In above code, the age value is 15 and in the try block - the if condition throws the exception if the value is less than 21. As soon as the exception is thrown the catch block gets executed. Hence as an output we get the first message "This is My Exception block". Then the control is transferred to the class MyOwnException(defined at the top of the program). The message is set and it is "Your age is very less than the condition". This message can then printed by the catch block using the System.out.println statement by means of e.message.
- At the end the finally block gets executed.

Ex. 3.9.1 : Write an exception class for a time of day that can accept only 24 hour representation of clock hours. Write a java program to input various formats of timings and throw suitable error messages.

AU : IT : Dec.-13, Marks 8

Sol. :

```

import java.lang.Exception;
import java.io.*;
import java.util.*;

class MyException extends Exception
{
    MyException(String msg)
    {
        super(msg);
    }
}

class Clock
{
    private int hour;
    private int minute;
    public void input()throws IOException
    {

```

```

BufferedReader br=new BufferedReader (new InputStreamReader(System.in));
System.out.println("\n Enter the time in hh:mm format");
String str=br.readLine();
StringTokenizer tokn=new StringTokenizer(str,":");
String h=tokn.nextToken();
String m=tokn.nextToken();
hour=Integer.parseInt(h);
minute=Integer.parseInt(m);
try
{
    System.out.println("Hour: "+hour);
    if((hour < 0) || (hour >24))
        throw new MyException("Fatal error: invalid hour");
}
catch(MyException e)
{
    System.out.println(e.getMessage());
}
try
{
    System.out.println("Minute: "+minute);
    if((minute <0) || (minute > 59))
        throw new MyException("Fatal error: invalid minute");
}
catch(MyException e)
{
    System.out.println(e.getMessage());
}
}

```

```

class ClockDemo
{
public static void main(String[] args)throws IOException
{
Clock c=new Clock();
c.input();
}
}.

```

Output(Run1)

Enter the time in hh:mm format

25:80

Hour: 25

Fatal error: invalid hour

Minute: 80

Fatal error: invalid minute

Output(Run2)

Enter the time in hh:mm format

10:70

Hour: 10

Minute: 70

Fatal error: invalid minute

Output(Run3)

Enter the time in hh:mm format

30:40

Hour: 30

Fatal error: invalid hour

Minute: 40

Review Question

- With suitable Java program, explain user defined exception handling.

Part II : Multithreaded Programming

AU : Dec.-19, Marks 13

3.10 Basic Concepts**3.10.1 What is Thread ?**

- One of the exciting features of Windows operating system is that - It allows the user to handle multiple tasks together. This facility in Windows operating system is called multitasking.
- In Java we can write the programs that perform multitasking using the multithreading concept. Thus Java allows to have multiple control flows in a single program by means of multithreading.
- Definition of thread :** Thread is a tiny program running continuously. It is sometimes called as light-weight process. But there lies differences between thread and process.

Sr. No.	Thread	Process
1.	Thread is a light-weight process.	Process is a heavy weight process.
2.	Threads do not require separate address space for its execution. It runs in the address space of the process to which it belongs to.	Each process requires separate address space to execute.

3.10.2 Difference between Multithreading and Multiprocessing

Sr. No.	Multithreading	Multiprocessing
1.	Thread is a fundamental unit of multithreading.	Program or process is a fundamental unit of multiprocessing environment.
2.	Multiple parts of a single program gets executed in multithreading environment.	Multiple programs get executed in multiprocessing environment.
3.	During multithreading the processor switches between multiple threads of the program.	During multiprocessing the processor switches between multiple programs or processes.
4.	It is cost effective because CPU can be shared among multiple threads at a time.	It is expensive because when a particular process uses CPU other processes has to wait.
5.	It is highly efficient.	It is less efficient in comparison with multithreading.
6.	It helps in developing efficient application programs.	It helps in developing efficient operating system programs.

Review Question

1. What is thread ? Explain multithreading and multitasking in detail.

AU : CSE : Dec.-19, Marks 13

3.11 Java Thread Model

AU : Dec.-10, 11, 18, May-12, 13, Marks 13

Thread life cycle specifies how a thread gets processed in the Java program. By executing various methods. Following figure represents how a particular thread can be in one of the state at any time.

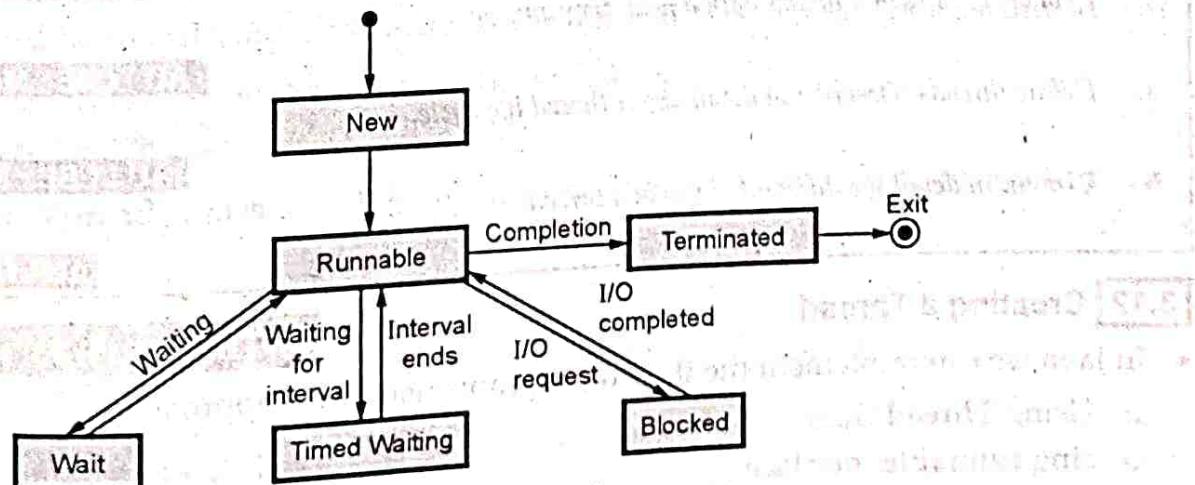


Fig. 3.11.1 Life cycle of thread

New state

- When a thread starts its life cycle it enters in the new state or a create state.

Runnable state

- This is a state in which a thread starts executing.
- Waiting state
- Sometimes one thread has to undergo in waiting state because another thread starts executing.

Timed waiting state

- There is a provision to keep a particular threading waiting for some time interval. This allows to execute high prioritized threads first. After the timing gets over, the thread in waiting state enters in runnable state.

Blocked state

- When a particular thread issues an Input/Output request then operating system sends it in blocked state until the I/O operation gets completed. After the I/O completion the thread is sent back to the runnable state.

Terminated state

- After successful completion of the execution the thread in runnable state enters the terminated state.

Review Questions

1. What is thread ? Explain its state and methods.

AU : CSE : Dec.-10, Marks 8

2. Write about various threads states in Java.

AU : CSE : Dec.-11, Marks 8

3. Explain : States of a thread with a neat diagram.

AU : CSE : May-12, Marks 10

4. Define threads. Describe in detail about thread life cycle.

AU : CSE : May-13, Marks 8

5. Explain in detail the different states of a thread.

AU : Dec.-18, Marks 13

3.12 Creating a Thread

AU : Dec.-10, 11, 13, May-13, Marks 16

- In Java we can implement the thread programs using two approaches -
 - Using Thread class
 - Using runnable interface.

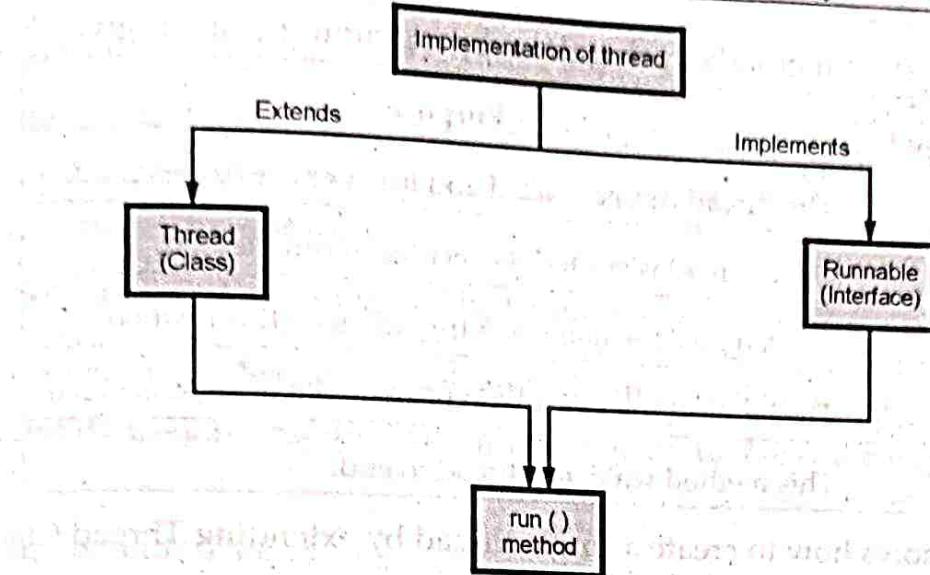


Fig. 3.12.1 : Creation of Java thread

As given in Fig. 3.12.1, there are two methods by which we can write the Java thread programs one is by extending **thread class** and the other is by implementing the **Runnable interface**.

- The **run()** method is the most important method in any threading program. By using this method the thread's behaviour can be implemented. The run method can be written as follows -

```

public void run()
{
    //statements for implementing thread
}
  
```

For invoking the thread's run method the object of a thread is required. This object can be obtained by creating and initiating a thread using the **start()** method.

3.12.1 Extending Thread Class

- The **Thread class** can be used to create a thread.
- Using the **extend keyword** your class extends the **Thread class** for creation of thread. For example if I have a class named **A** then it can be written as

```
class A extends Thread
```

- Constructor of Thread Class :** Following are various syntaxes used for writing the constructor of Thread Class.

```

Thread()
Thread(String s)
Thread(Runnable obj)
Thread(Runnable obj, String s);
  
```

- Various commonly used methods during thread programming are as given below -

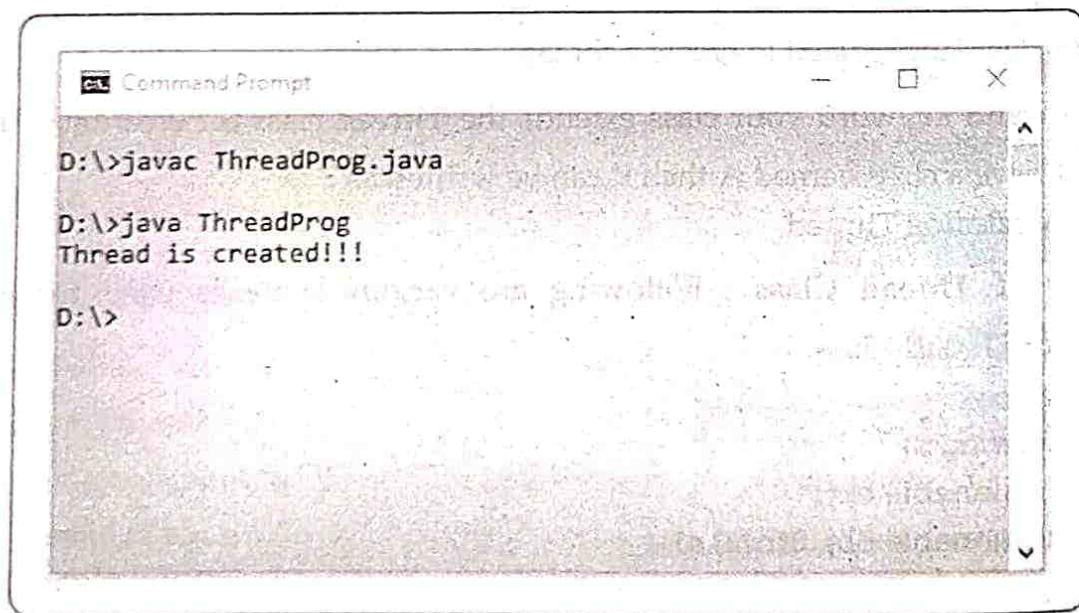
Method	Purpose
start()	The thread can be started and invokes the run method.
run()	Once thread is started it executes in run method.
setName()	We can give the name to a thread using this method.
getName()	The name of the thread can be obtained using this name.
join()	This method waits for thread to end.

Following program shows how to create a single thread by extending Thread Class.

Java Program

```
class MyThread extends Thread
{
    public void run()
    {
        System.out.println("Thread is created!!!");
    }
}
class ThreadProg
{
    public static void main(String args[])
    {
        MyThread t=new MyThread();
        t.start();
    }
}
```

Output



D:\>javac ThreadProg.java
D:\>java ThreadProg
Thread is created!!!
D:\>

Program Explanation :

In above program, we have created two classes. One class named **MyThread** extends the **Thread** class. In this class the **run** method is defined. This **run** method is called by **t.start()** in **main()** method of class **ThreadProg**.

The thread gets created and executes by displaying the message **Thread is created.**

Ex. 3.12.1 : Create a thread by extending the Thread Class. Also make use of constructor and display message "You are Welcome to Thread Programming".

Sol. :

```
class MyThread extends Thread
{
    String str = ""; //data member of class MyThread
    MyThread(String s)//constructor
    {
        this.str=s;
    }
    public void run()
    {
        System.out.println(str);
    }
}
class ThreadProg
{
    public static void main(String args[])
    {
        MyThread t=new MyThread("You are Welcome to Thread Programming");
        t.start();
    }
}
```

Output

You are Welcome to Thread Programming

3.12.2 Implementing Runnable Interface

- The thread can also be created using **Runnable** interface.
- Implementing thread program using **Runnable interface** is preferable than implementing it by extending the **thread** class because of the following two reasons -
 1. If a class extends a **thread** class then it can not extends any other class which may be required to extend.
 2. If a class **thread** is extended then all its functionalities get inherited. This is an expensive operation.

- Following Java program shows how to implement **Runnable** interface for creating a single thread.

Java Program

```
class MyThread implements Runnable
{
    public void run()
    {
        System.out.println("Thread is created!");
    }
}
class ThreadProgRunn
{
    public static void main(String args[])
    {
        MyThread obj=new MyThread();
        Thread t=new Thread(obj);
        t.start();
    }
}
```

Output

```
D:\>javac ThreadProgRunn.java
D:\>java ThreadProgRunn
Thread is created!
```

Program Explanation :

- In above program, we have used interface **Runnable**.
- While using the interface, it is necessary to use **implements** keyword.
- Inside the **main** method
 - Create the instance of class **MyClass**.
 - This instance is passed as a parameter to **Thread** class.
 - Using the instance of class **Thread** invoke the **start** method.
 - The start method in-turn calls the **run** method written in **MyClass**.
- The **run** method executes and display the message for thread creation.

Ex. 3.12.2 : Create a thread by extending the Thread Class. Also make use of constructor and display message "You are Welcome to Thread Programming".

Sol. :

```

class MyThread implements Runnable
{
    String str;
    MyThread(String s)
    {
        this.str=s;
    }
    public void run()
    {
        System.out.println(str);
    }
}

class ThreadProgRunn
{
    public static void main(String args[])
    {
        MyThread obj=new MyThread("You are Welcome to Thread Programming");
        Thread t=new Thread(obj);
        t.start();
    }
}

```

Output

You are Welcome to Thread Programming

Ex. 3.12.3 : Write a Java program that prints the numbers from 1 to 10 line by line after every 10 seconds.

AU : IT : Dec.-10

Sol. :

Java Program [MultiThNum.java]

```

class NumPrint extends Thread
{
    int num;
    NumPrint()
    {
        start(); // directs to the run method
    }
    public void run() // thread execution starts
    {
        try

```

```

{
    for(int i=1;i<=10;i++)
    {
        System.out.println(i);
        Thread.sleep(10000); //10 sec,b'coz 1000millisec=1 sec.
    }
}
catch(InterruptedException e)
{
    System.out.println("Exception in Thread handling");
}
}

public class MultiThNum
{
    public static void main(String args[])
    {
        NumPrint t1;
        t1=new NumPrint();
    }
}

```

Output

C:>javac MultiThNum.java
C:>java MultiThNum

1
2
3
4
5
6
7
8
9
10

Review Questions

- Explain how threads are created in Java.*
- How to extends the thread class ? Give an example.*
- How to implement runnable interface for creating and starting threads ?*
- What is meant by concurrent programming ? Define thread. Discuss the two ways of implementing thread using example.*

AU : CSE : Dec.-11, Marks 8

AU : CSE : May-13, Marks 8

AU : CSE : May-13, Marks 8

AU : CSE : Dec.-13, Marks 16

3.13 Multithreading**AU : Dec.-11,12, 18, 19, May-12, 19, Marks 16**

The multiple threads can be created both by extending Thread class and by implementing the Runnable interface.

1. Java Program for creating multiple threads by extending Thread Class

class A extends Thread

```
{
    public void run()
    {
        for(int i=0;i<=5;i++)//printing 0 to 5
        {
            System.out.println(i);
        }
    }
}
```

class B extends Thread

```
{
    public void run()
    {
        for(int i=10;i>=5;i--)//printing 10 to 5
        {
            System.out.println(i);
        }
    }
}
```

class ThreadProg

```
{
    public static void main(String args[])
    {
        A t1=new A();
        B t2=new B();
        t1.start();
        t2.start();
    }
}
```

Output

```
0
1
2
3
4
5
10
9
```

8
7
6
5

2. Java Program for creating multiple threads by implementing the Runnable interface

```
class A implements Runnable
```

```
{
```

```
    public void run()
```

```
{
```

```
        for(int i=0;i<=5;i++)//printing 0 to 5
```

```
{
```

```
            System.out.println(i);
```

```
}
```

```
}
```

```
class B implements Runnable
```

```
{
```

```
    public void run()
```

```
{
```

```
        for(int i=10;i>=5;i--)//printing 10 to 5
```

```
{
```

```
            System.out.println(i);
```

```
}
```

```
}
```

```
class ThreadProgRunn
```

```
{
```

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

```
{
```

```
        A obj1=new A();
```

```
        B obj2=new B();
```

```
        Thread t1=new Thread(obj1);
```

```
        Thread t2=new Thread(obj2);
```

```
        t1.start();
```

```
        t2.start();
```

```
}
```

```
}
```

Output

0
1
2
3
4

5
10
9
8
7
6
5

Ex. 3.13.1 : Write a Java application program for generating four threads to perform the following operations - i) Getting N numbers as input ii) Printing the numbers divisible by five iii) Printing prime numbers iv) Computing the average.

Sol. :

```

import java.io.*;
import java.util.*;
class FirstThread extends Thread
{
    public void run() //generating N numbers
    {
        int i;
        System.out.println("\nGenerating Numbers: ");
        for (i=1;i<=10;i++)
        {
            System.out.println(i);
        }
    }
}

class SecondThread extends Thread
{
    public void run() //Displaying the numbers divisible by five
    {
        int i;
        System.out.println("\nDivisible by Five: ");
        for (i=1;i<=10;i++) //10 can be replaced by any desired value
        {
            if (i%5==0)
                System.out.println(i);
        }
    }
}

class ThirdThread extends Thread
{
}

```

```

public void run() //generating the prime numbers
{
    int i;
    System.out.println("\nPrime Numbers: ");
    for (i=1;i<=10;i++) //10 can be replaced by any desired value
    {
        int j;
        for (j=2; j<i; j++)
        {
            int n = i%j;
            if (n==0)
                break;
        }
        if(i == j)
            System.out.println(i);
    }
}

class FourthThread extends Thread
{
    public void run() //generating the prime numbers
    {
        int i,sum;
        double avg;
        sum=0;
        System.out.println("\nComputing Average: ");
        for (i=1;i<=10;i++) //10 can be replaced by any desired value
        {
            sum=sum+i;
            avg=sum/(i-1);
            System.out.println(avg);
        }
    }
}

class MainThread
{
    public static void main(String[] args) throws IOException
    {
        FirstThread T1 = new FirstThread(); //creating first thread
        SecondThread T2 = new SecondThread(); //creating second thread
        ThirdThread T3 = new ThirdThread(); //creating Third thread
        FourthThread T4 = new FourthThread(); //creating Fourth thread
    }
}

```

```

    T1.start(); //First Thread starts executing
    T2.start(); //Second Thread starts executing
    T3.start(); //Third Thread starts executing
    T4.start(); //Fourth Thread starts executing
}
}

```

Output**Generating Numbers:**

```

1
2
3
4
5
6
7
8
9
10

```

Divisible by Five:

```

5
10

```

Computing Average:

```

5.0

```

Prime Numbers:

```

2
3
5
7

```

Ex. 3.13.2 : Create a Bank Database application program to illustrate the use of multithreads

AU : Dec.-18, Marks 15

Sol. :

```

public class BankAppl implements Runnable {
    private Account acc = new Account();
    public static void main(String[] args) {
        BankAppl obj = new BankAppl();
        Thread t1 = new Thread(obj);
        Thread t2 = new Thread(obj);
        Thread t3 = new Thread(obj);
        t1.setName("Mr.XYZ");
        t2.setName("Mr.ABC");
        t3.setName("Mr.PQR");
        t1.start();
        t2.start();
    }
}

```

```

        t3.start();
    }

    public void run() {
        for (int x = 0; x < 10; x++) {
            makeWithdrawal(10);
            if (acc.getBalance() < 0) {
                System.out.println("Account Overdrawn!!!");
            }
        }
    }

    void makeWithdrawal(int amt) {
        int bal;
        bal = acc.getBalance();
        if (bal >= amt) {
            System.out.println("\t" + Thread.currentThread().getName() + " withdraws Rs." + amt);
            try {
                Thread.sleep(100);
            } catch (InterruptedException ex) {
            }
            acc.withdraw(amt);
            bal = acc.getBalance();
            System.out.println("\t The Balance: " + bal);
        } else {
            System.out.println("Insufficient Balance in account for " +
                Thread.currentThread().getName() + " to withdraw " +
                acc.getBalance());
        }
    }
}

class Account {
    private int balance = 100;
    public int getBalance() {
        return balance;
    }
    public void withdraw(int amount) {
        balance = balance - amount;
    }
}

```

Review Questions

1. What is multithreading ? What are the methods available in Java for inter-thread communication ? Discuss with example AU : IT : Dec.-11, Marks 10
2. Write a Java program to illustrate multithreaded programming. AU : CSE : Dec.-11,12, Marks 16
3. Write notes on multi-threaded programming. AU : IT : May-12, Marks 16

4. Write a java application that illustrate the use of multithreading. Explain the same with sample input.

AU : CSE : May-12, Marks 16

5. Describe the creation of a single thread and multiple threads using an example.

AU : CSE : May-19, Marks 13

6. Create a simple real life application program in Java to illustrate the use of multithreads.

AU : CSE : Dec.-19, Marks 15

3.14 Priorities

- In Java threads scheduler selects the threads using their priorities.
- The thread priority is a simple integer value that can be assigned to the particular thread. These priorities can range from 1 (lowest priority) to 10 (highest priority).
- There are two commonly used functionalities in thread scheduling -
 - o `setPriority`
 - o `getPriority`
- The function `setPriority` is used to set the priority to each thread
`Thread_Name.setPriority(priority_val);`

Where, `priority_val` is a constant value denoting the priority for the thread. It is defined as follows

1. `MAX_PRIORITY = 10`
 2. `MIN_PRIORITY = 1`
 3. `NORM_PRIORITY = 5`
- The function `getPriority` is used to get the priority of the thread.
`Thread_Name.getPriority();`

What is Preemption ?

- Preemption is a situation in which when the currently executed thread is suspended temporarily by the highest priority thread.
- The highest priority thread always preempts the lowest priority thread.

Let us see the illustration of thread prioritizing with the help of following example -

Java program[Thread_PR_Prog.java]

```
class A extends Thread
```

```
{
```

```
public void run()
```

```
{
```

```
System.out.println("Thread #1");
```

```
for(int i=1;i<=5;i++)
```

```
{
```

```
System.out.println("\tA: "+i);
```

```

        }

        System.out.println("\n-----End of Thread#1-----");

    }

}

class B extends Thread
{
    public void run()
    {
        System.out.println("Thread #2");
        for(int k=1;k<=5;k++)
        {
            System.out.println("tB: "+k);
        }
        System.out.println("\n-----End of Thread#2-----");
    }
}

class Thread_PR_Prog
{
    public static void main(String[] args)
    {
        A obj1=new A();
        B obj2=new B();
        obj1.setPriority(1);
        obj2.setPriority(10);//highest priority
        System.out.println("Strating Thread#1");
        obj1.start();
        System.out.println("Strating Thread#2");
        obj2.start();
    }
}

```

Output

Strating Thread#1

Strating Thread#2

Thread #1

Thread #2

B: 1
B: 2
B: 3
B: 4
B: 5

Thread 1 and Thread 2 starts

Thread 2 preempts the execution of thread 1

End of Thread#2.....

A: 1
A: 2
A: 3
A: 4
A: 5

Then Thread 1 continues its execution

.....End of Thread#1.....

program explanation

In above program,

- 1) We have created two threads - the **Thread#1** and **Thread#2**.
- 2) We assign highest priority to **Thread#2**.
- 3) In the main function both the threads are started but as the **Thread#2** has higher priority than the **Thread#1**, the **Thread#2** preempts the execution of **Thread#1**.
- 4) Thus **Thread#2** completes its execution and then **Thread#1** completes.

3.15 Synchronization**AU : Dec.-14, 19, Marks 13**

- When two or more threads need to access shared memory, then there is some way to ensure that the access to the resource will be by only one thread at a time. The process of ensuring one access at a time by one thread is called synchronization. The synchronization is the concept which is based on monitor. Monitor is used as mutually exclusive lock or mutex. When a thread owns this monitor at a time then the other threads can not access the resources. Other threads have to be there in waiting state.
- In Java every object has implicit monitor associated with it. For entering in object's monitor, the method is associated with a keyword synchronized. When a particular method is in synchronized state then all other threads have to be there in waiting state.
- There are two ways to achieve the synchronization -
 1. Using Synchronized Methods
 2. Using Synchronized Blocks (Statements)

Let us make the method synchronized to achieve the synchronization by using following Java program -

1. Using Synchronized Method

```
class Test
{
    synchronized void display(int num)
    {
        System.out.println("\nTable for "+num);
        for(int i=1;i<=10;i++)
        {
            System.out.print(i*num);
        }
    }
}
```

This is a synchronized method

```

        System.out.print(" "+num*i);
    }
    System.out.print("\nEnd of Table");
    try
    {
        Thread.sleep(1000);
    }catch(Exception e){}
}
}

class A extends Thread
{
    Test th1;
    A(Test t)
    {
        th1=t;
    }
    public void run()
    {
        th1.display(2);
    }
}

class B extends Thread
{
    Test th2;
    B(Test t)
    {
        th2=t;
    }
    public void run()
    {
        th2.display(100);
    }
}

class MySynThread
{
    public static void main(String args[])
    {
        Test obj=new Test();
        A t1=new A(obj);
        B t2=new B(obj);
        t1.start();
        t2.start();
    }
}

```

Output

```
D:\>javac MySynThread.java
D:\>java MySynThread
Table for 2
2 4 6 8 10 12 14 16 18 20
End of Table
Table for 100
100 200 300 400 500 600 700 800 900 1000
End of Table
D:\>
```

Program Explanation :

- In above program we have written one class named Test. Inside this class the synchronized method named display is written. This method displays the table of numbers.
- We have written two more classes named A and B for executing the thread. The constructors for class A and Class B are written and to initialize the instance variables of class Test as th1 (as a variable for class A)and th2(as a variable for class B)
- Inside the run methods of these classes we have passed number 2 and 10 respectively and display method is invoked.
- The display method executes firstly for thread t1 and then for t2 in synchronized manner.

2. Using Synchronized Block

- When we want to achieve synchronization using the synchronized block then create a block of code and mark it as synchronized.
- Synchronized statements must specify the object that provides the intrinsic lock.

Syntax

The syntax for using the synchronized block is -

```
synchronized(object reference)
{
    statement;
    statement; //block of code to be synchronized
}
```

Java Program

```

class Test
{
    void display(int num)
    {
        synchronized(this)
        {
            System.out.println("\nTable for "+num);
            for(int i=1;i<=10;i++)
            {
                System.out.print(" "+num*i);
            }
            System.out.print("\nEnd of Table");
        }
        try
        {
            Thread.sleep(1000);
        }catch(Exception e){}
    }
}

```

This is a synchronized Block

```

}
class A extends Thread
{
}

```

class A extends Thread

```

{
    Test th1;
    A(Test t)
    {
    }
    th1=t;
}

```

```

    public void run()
    {
    }
}

```

```

class B extends Thread
{
}

```

```

    Test th2;
    B(Test t)
    {
    }
    th2=t;
}

```

```

    public void run()
    {
    }
}

```

```

    th2.display(100);
}
}

```

```
class MySynThreadBlock
```

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

```
{ Test obj=new Test();
```

```
A t1=new A(obj);
```

```
B t2=new B(obj);
```

```
t1.start();
```

```
t2.start();
```

```
}
```

Output

D:\>javac MySynThreadBlock.java
D:\>java MySynThreadBlock
Table for 2
2 4 6 8 10 12 14 16 18 20
End of Table
Table for 100
100 200 300 400 500 600 700 800 900 1000
End of Table
D:\>

Points to Remember about Synchronization

1. Only methods can be synchronized but the variables and classes can not be synchronized.
2. Each object has one lock.
3. A class contains several methods and all methods need not be synchronized.
4. If two threads in a class wants to execute synchronized methods and both the methods are using the same instance of a class then only one thread can access the synchronized method at a time.
5. We can not synchronize the constructors.
6. A thread can acquire more than one lock.

Review Questions

1. What is thread synchronization? Discuss with an example.

AU : Dec. 14, Marks 8

2. What is synchronization? Explain the different types of synchronization in java.

AU : Dec. 19, Marks 13

3.16 Inter Thread Communication

- Two or more threads communicate with each other by exchanging the messages. This mechanism is called interthread communication.
- Polling is a mechanism generally implemented in a loop in which certain condition is repeatedly checked.
- To better understand the concept of polling, consider producer-consumer problem in which producer thread produces and the consumer thread consumes whatever is produced.
- Both must work in co-ordination to avoid wastage of CPU cycles.
- But there are situations in which the producer has to wait for the consumer to finish consuming of data. Similarly the consumer may need to wait for the producer to produce the data.
- In Polling system either consumer will waste many CPU cycles when waiting for producer to produce or the producer will waste CPU cycles when waiting for the consumer to consume the data.
- In order to avoid polling there are three in-built methods that take part in inter-thread communication -

notify()	If a particular thread is in the sleep mode then that thread can be resumed using the notify call.
notifyall()	This method resumes all the threads that are in suspended state.
wait()	The calling thread can be send into a sleep mode.

Example Program

Following is a simple Java program in which two threads are created one for producer and another is for consumer.

The producer thread **produces(writes)** the numbers from 0 to 9 and the consumer thread **consumes(reads)** these numbers.

The **wait** and **notify** methods are used to send particular thread to sleep or to resume the thread from sleep mode respectively.

Java Program[InterThread.java]

```
class MyClass
{
    int val;
    boolean flag = false;
    synchronized int get() //by toggling the flag synchronised read and write is performed
    {
        if(!flag)
            return val;
        else
        {
            flag = false;
            return val;
        }
    }
}
```

```

try
{
    wait();
}
catch(InterruptedException e)
{
    System.out.println("InterruptedException!!!");

}
System.out.println("Consumer consuming: " + val);
flag = false;
notify();
return val;
}

synchronized void put(int val)
{
if(flag)
try
{
    wait();
}
catch(InterruptedException e)
{
    System.out.println("InterruptedException!!!");

}
this.val = val;
flag = true;
System.out.println("Producer producing " + val);
notify();
}
}

class Producer extends Thread
{
MyClass th1;
Producer(MyClass t)
{
    th1 = t;
}
public void run()
{
    for(int i = 0;i<10;i++)
    {
        th1.put(i);
    }
}
}

```

Product thread is writing
the numbers from 0 to 9

```

class Consumer extends Thread
{
    MyClass th2;
    Consumer(MyClass t)
    {
        th2 = t;
    }
    public void run()
    {
        for(int i = 0;i<10;i++)
        {
            th2.get();
        }
    }
}
class InterThread
{
    public static void main(String[] arg)
    {
        MyClass TObj = new MyClass();
        Producer pthread=new Producer(TObj);
        Consumer cthread=new Consumer(TObj);
        pthread.start();
        cthread.start();
    }
}

```

Consumer thread is reading
the numbers from 0 to 9

Producer producing 0
 Consumer consuming: 0
 Producer producing 1
 Consumer consuming: 1
 Producer producing 2
 Consumer consuming: 2
 Producer producing 3
 Consumer consuming: 3
 Producer producing 4
 Consumer consuming: 4
 Producer producing 5
 Consumer consuming: 5
 Producer producing 6
 Consumer consuming: 6
 Producer producing 7
 Consumer consuming: 7
 Producer producing 8

Output

Consumer consuming: 8

Producer producing 9

Consumer consuming: 9

Program Explanation

- Inside `get()` -

The `wait()` is called in order to suspend the execution by that time the producer writes the value and when the data gets ready it notifies other thread that the data is now ready.

Similarly when the consumer reads the data execution inside `get()` is suspended. After the data has been obtained, `get()` calls `notify()`. This tells producer that now the producer can write the next data in the queue.

- Inside `put()` -

The `wait()` suspends execution by that time the consumer removes the item from the queue. When execution resumes, the next item of data is put in the queue, and `notify()` is called. When the notify is issued the consumer can now remove the corresponding item for reading.

3.17 Suspending – Resuming, and Stopping Threads

AU : May-13. Marks 16

- Stopping a thread : A thread can be stopped from running further by issuing the following statement -

```
th.stop();
```

By this statement the thread enters in a **dead state**. From stopping state a thread can never return to a **runnable state**.

- Blocking a thread : A thread can be temporarily stopped from running. This is called blocking or suspending of a thread. Following are the ways by which thread can be blocked -

1. `sleep()`

By `sleep` method a thread can be blocked for some specific time. When the specified time gets elapsed then the thread can return to a runnable state.

2. `suspend`

By `suspend` method the thread can be blocked until further request comes. When the `resume()` method is invoked then the thread returns to a runnable state.

3. `wait`

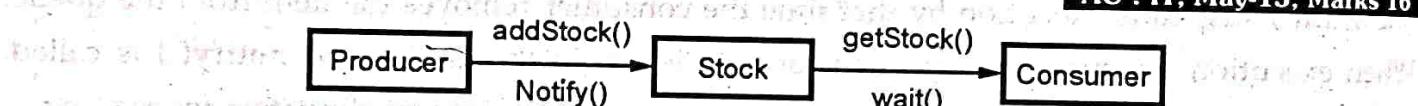
The thread can be made suspended for some specific conditions. When the `notify` method is called then the blocked thread returns to the runnable state.

- The difference between the suspending and stopping thread is that if a thread is suspended then its execution is stopped temporarily and it can return to a runnable state. But in case , if a thread is stopped then it goes to a dead state and can never return to runnable state.
- Resuming a thread : The resume() method is only used with suspend() method. This method is only to resume a thread which was suspended using suspend() method. The syntax is –

public final resume()

Ex. 3.17.1 : Write a java program for inventory problem to illustrates the usage of thread synchronized keyword and inter thread communication process. They have three classes called consumer, producer and stock.

AU : IT, May-13, Marks 16



Sol. :

```

class Consumer implements Runnable
{
    Stock obj1;
    Thread t;
    Consumer (Stock obj1)
    {
        this.obj1 = obj1;
        t = new Thread(this,"Consumer thread");
        t.start();
    }
    public void run()
    {
        while (true)
        {
            try {
                obj1.getStock((int)(Math.random()*100));
            } catch (InterruptedException e) { }
        }
    }
    void stop()
    {
        t.stop();
    }
}
  
```

class Producer implements Runnable

```

{
    Stock obj2;
    Thread t;
    Producer(Stock obj2)
    {
        this.obj2 = obj2;
        t = new Thread(this, "Producer thread");
        t.start();
    }
    public void run()
    {
        while(true)
        {
            try
            {
                t.sleep(900);
            } catch (InterruptedException e) { }
            obj2.addStock((int)(Math.random()*100));
        }
    }
    void stop()
    {
        t.stop();
    }
}

```

class Stock

```

{
    int items = 0;
    public synchronized void addStock(int i)
    {
        items = items + i;
        System.out.println("Stock added :" + i);
        System.out.println("present stock :" + items );
        notify();
    }
    public synchronized int getStock(int j)
    {
        while(true)
        {
            if(items >= j)
            {
                items = items - j;
                System.out.println("Item is removed from stock :" + j);
                System.out.println("Current stock is :" + items);
            }
        }
    }
}
```

```

        break;
    }
    else
    {
        System.out.println("\tStock is not enough!!!");
        System.out.println ("\t Waiting for items to get added..."); do what you want
        try {
            wait();
        }catch(InterruptedException e) { }
    }
}
return items;
}

public static void main(String args[])
{
    Stock j = new Stock();
    Producer p = new Producer(j);
    Consumer c = new Consumer(j);
    try
    {
        Thread.sleep(10000);
        p.stop();
        c.stop();
        p.t.join();
        c.t.join();
        System.out.println("Thread stopped");
    } catch(InterruptedException e) { }
    System.exit(0);
}
}

```

3.18 Wrappers

- Wrapper classes are those classes that allow primitive data types to be accessed as objects.
- The wrapper class is one kind of wrapper around a primitive data type. The wrapper classes represent the primitive data types in its instance of the class.
- Following table shows various primitive data types and the corresponding wrapper classes -

Primitive data type	Wrapper class
boolean	Boolean
byte	Byte
char	Character

double	Double
float	Float
int	Integer
long	Long
short	Short
void	Void

- Methods to handle wrapper classes are enlisted in the following table -

Method used	Description
Integer val=new Integer(int_var);	An object is created for the integer variable int_var.
Float val=new Float(f_var)	An object is created for the Float variable f_var.
Double val=new Double(d_var)	An object is created for the double variable d_var.
Long val=new Long(Long_var)	An object is created for the Long variable Long_var.

- Suppose an object for holding an integer value is created then we can retrieve the integer value from it using **typeValue()** method. For instance the object obj contains an integer value then we can obtain the integer value from obj. It is as follows -
`int num=obj.intValue();`
- Similarly we can use **floatValue()**, **doubleValue()** and **longValue()**.
- Similarly in order to convert the numerical value to string the **toString()** method can be used. For instance
`str=Integer.toString(int_val)`
- The variable **int_val** can be converted to string **str**.
- For converting the string to numerical value the **parseInt** or **parseLong** methods can be used.

Points to remember about wrapper classes

1. The wrapper classes do not contain the constructors.
2. The methods of the wrapper classes are static.
3. After assigning the values to the wrapper class we cannot change them.

Example : Java Program

```

import java.io.*;
import java.lang.*;
class WrapperDemo
{
    public static void main(String[] args)
    {
        System.out.println("Creating an object for value 10");
        Integer i=new Integer(10);
        System.out.println("Obtaining the value back from the object: "+i.intValue());
        String str="100";
        System.out.println("The string is: "+str);
        System.out.println("Obtaining the numeric value from the string: "+ Integer.parseInt(str));
    }
}

```

Output

Creating an object for value 10
 Obtaining the value back from the object: 10
 The string is: 100
 Obtaining the numeric value from the string: 100

Ex. 3.18.1 : Define wrapper class. Give the following wrapper class methods with syntax and use :

- 1) To convert integer number to string.
- 2) To convert numeric string to integer number.
- 3) To convert object numbers to primitive numbers sing type value () method.

Sol. : Wrapper class - Refer section 3.18.

```

class WrapperDemo1
{
    public static void main(String args[])
    {
        System.out.println("\tInteger to String Conversion");
        int i=100;
        String str=Integer.toString(i);
        System.out.println("int Value: "+i);
        System.out.println("Equivalent String: "+str);
        System.out.println("\tString to Integer Conversion");
        str="500";
        int j=Integer.parseInt(str);
        System.out.println("String: "+str);
        System.out.println("Equivalent int: "+j);
        System.out.println("\tobject to Primitive number")
    }
}

```

```

using typeValue Conversion");
Integer a=new Integer(1000);//creating object for float value
int val=a.intValue();
System.out.println("Integer: "+a);
System.out.println("Equivalent int value: "+val);
Float b=new Float(2000);//creating object for float value
float f=b.floatValue();
System.out.println("Float: "+b);
System.out.println("Equivalent float value: "+f);
}
}

```

Output**Integer to String Conversion**

int Value: 100

Equivalent String: 100

String to Integer Conversion

String: 500

Equivalent int: 500

object to Primitive number using typeValue Conversion

Integer: 1000

Equivalent int value: 1000

Float: 2000.0

Equivalent float value: 2000.0

Uses of Wrapper Class

- 1) Wrapper classes are used to convert numeric strings into numeric values.
- 2) Wrapper classes are used to convert numeric value to string using wrapper class.
- 3) Wrapper class is a medium to store primitive data type in an object.
- 4) Using the `typeValue()` method we can retrieve value of the object as its primitive data type.

Review Questions

1. Explain the use of wrapper classes in Java.
2. Explain the concept of wrapper classes with illustrative example.

3.19 Autoboxing

Definition : An automatic conversion of primitive data type into equivalent wrapper type is called as autoboxing.

This is a new feature of Java5

Example Program

```
class AutoboxExample
```

```
{\n    public static void main(String args[]){\n        {\n            //auto-boxing, an int is boxed into Integer object\n            Integer obj = 111;\n\n            //unboxing\n            int val = obj.byteValue();\n            System.out.println("Object value = "+obj);\n            System.out.println("The primitive value = "+val);\n        }\n    }\n}
```

Output

The screenshot shows a Command Prompt window titled "Command Prompt". The command E:\>javac AutoboxExample.java is run, followed by E:\>java AutoboxExample. The output displays "Object value = 111" and "The primitive value = 111".

```
E:\>javac AutoboxExample.java\nE:\>java AutoboxExample\nObject value = 111\nThe primitive value = 111\nE:\>
```

Program Explanation

In above program,

- (1) We are autoboxing an integer type value.
- (2) Then using the `byteValue()` method we are unboxing this value. This method converts the given number into a primitive byte type and returns the value of integer object as byte.
- (3) Finally we are displaying both the object value and primitive value.

3.20 Two Marks Questions with Answers**Q.1 What is an exception ? Give example.****AU : IT, CSE : May, 12**

Ans : Exception is a mechanism which is used for handling unusual situation that may occur in the program. For example -

ArithmeticException : This exception is used to handle arithmetic exceptions such as divide by zero.

IOException : This exception occurs when an illegal input/output operation is performed.

Q.2 What will happen if an exception is not caught ?**AU : CSE : Nov, 10**

Ans. : An uncaught exception results in invoking of the uncaughtException() method. As a result eventually the program will terminate in which it is thrown.

Q.3 What is the benefit of exception handling ?

Ans. : When calling method encounters some error then the exception can be thrown. This avoids crashing of the entire application abruptly.

Q.4 What is the difference between error and exception in java?

Ans. : Errors are mainly caused by the environment in which an application is running. For example, OutOfMemoryError happens there is shortage of memory. Whereas exceptions are mainly caused by the application itself. For example, NullPointerException occurs when an application tries to access null object.

Q.5 What is compile time and run time error ?**Ans. :**

1. The errors that are detected by the Java compiler during the compile time are called compiler time errors.
2. The runtime errors are basically the logically errors that get caused due to wrong logic.

Q.6 What is the use of try, catch keywords ?**Ans. : try -** A block of source code that is to be monitored for the exception.**catch -** The catch block handles the specific type of exception along with the try block.**Q.7 What is the difference between throw and throws ?****Ans. :**

1. The throw keyword is used to explicitly throw an exception.
2. The throws keyword is used to declare an exception.

Q.8 What is ArrayIndexOutOfBoundsException ?

Ans. : When we use an array index which is beyond the range of index then ArrayIndexOutOfBoundsException occurs.

Q.9 What is the need of multiple catch ?**Ans. :**

- There may be the situations in which different exceptions may get raised by a single try block statements and depending upon the type of exception thrown it must be caught.
- To handle such situation multiple catch blocks may exist for the single try block statements.

Q.10 There are three statements in a try block – statement1, statement2 and statement3. After that there is a catch block to catch the exceptions occurred in the try block. Assume that exception has occurred in statement2. Does statement3 get executed or not?

Ans. : No. Once a try block throws an exception, remaining statements will not be executed.

Control comes directly to catch block.

Q.11 Explain the types of exceptions.

Ans. : There are two types of exceptions :

1. **Checked Exception :** These types of exceptions need to be handled explicitly by the code itself either by using the try-catch block or by using throws. These exceptions are extended from the `java.lang.Exception` class.

For example : `IOException` which should be handled using the try-catch block.

2. **Unchecked Exception :** These type of exceptions need not be handled explicitly. The Java Virtual Machine handles these type of exceptions. These exceptions are extended from `java.lang.RuntimeException` class.

For example : `ArrayIndexOutOfBoundsException`, `NullPointerException`, `RunTimeException`.

Q.12 Does finally block get executed If either try or catch blocks are returning the control ?

Ans. : Yes, finally block will be always executed no matter whether try or catch blocks are returning the control or not.

Q.13 Can we have empty catch block ?

Ans. : Yes we can have empty catch block, but it is an indication of poor programming practice. We should never have empty catch block because if the exception is caught by that block, we will have no information about the exception and it will create problem while debugging the code.

Q.14 Which class is the super class for all types of errors and exceptions in java ?

Ans. : The `java.lang.Throwable` is the super class for all types of errors and exceptions in java.

Q.15 What is runtime exceptions ?

AU : Dec.-18

Ans. : `RuntimeException` is the superclass of those exceptions that can be thrown during the normal operation of the Java Virtual Machine. `Runtime Exception` is the parent class in all exceptions of the Java.

Q.16 What is exception handling ?**AU : Dec.-19**

Ans. : Exception handling is a mechanism in which the statements that are likely to cause an exception are enclosed within try block. As soon as exception occurs it is handled using catch block.

Thus exception handling is a mechanism that prevents the program from crashing when some unusual situation occurs.

Q.17 What happens when the statement int value = 25/0 is executed ?**AU : May-19**

Ans. : The exception Arithmetic Exception : Divide by zero will occur.

Q.18 What do you mean by threads in Java ?**AU : CSE : Dec.-11; IT : May-12, Dec.-13**

Ans. : Thread is tiny program running continuously. It is a light weight process in Java.

Q.19 Give the difference between process and thread.**AU : IT : Dec.-11**

Ans. : Refer section 3.10.1.

Q.20 What are different stages in thread ?**AU : CSE : Dec.-10, 19**

Ans. : Various stages in thread are -

- | | | |
|-----------------------|-------------------|---------------------|
| 1. New state | 2. Runnable state | 3. Waiting state |
| 4. Time waiting state | 5. Blocked state | 6. Terminated state |

Q.21 What do you mean by synchronization ?**AU : CSE : Dec.-10, May-12**

Ans. : When two or more threads need to access shared memory, then there is some way to ensure that the access to the resource will be by only one thread at a time. The process of ensuring one access at a time by one thread is called synchronization.

Q.22 What are the three ways by which the thread can enter in waiting stage ?**AU : IT : Dec.-10**

Ans. :

- Waiting state :** Sometimes one thread has to undergo in waiting state because another thread starts executing. This can be achieved by using wait() state.
- Timed waiting state :** There is a provision to keep a particular threading waiting for some time interval. This allows to execute high prioritized threads first. After the timing gets over, the thread in waiting state enters in runnable state. This can be achieved by using the sleep() command.
- Blocked state :** When a particular thread issues an input/output request then operating system sends it in blocked state until the I/O operation gets completed. After

the I/O completion the thread is sent back to the runnable state. This can be achieved by using suspend() method.

Q.23 What is multithreading ?

AU : CSE : Dec.-11, May-12; IT : Dec.-12, 18

Ans. : Multithreading is an environment in which multiple threads are created and they can execute simultaneously. The multiple threads can be created either by extending the thread class or by implementing the runnable interface.

Q.24 Mention two mechanisms for protecting a code block from concurrent access.

Ans. : There are two mechanisms for protecting a code block from concurrent access -

1. Use of reentrant code
2. Use synchronized block

Q.25 What is meant by notify methods in multithreading ?

Ans. : The notify() method is used for inter thread communication. If a particular thread is in the sleep mode then that thread can be resumed by using the notify call.

Q.26 What are the two ways of creating a thread ?

AU : IT : Dec-12

Ans. : Thread can be created using

1. Thread class
2. runnable interface.

Q.27 Why do we need run() and start() method both ? Can we achieve it with only run() method ?

AU : IT : May-13

Ans. : We use start() method to start a thread instead of calling run() method because the run() method is not a regular class method. It should only be called by the JVM. If we call the run() method directly then it will simply invoke the caller's thread instead of its own thread. Hence the start() method is called which schedules the thread with the JVM.

Q.28 What is the need for thread ?

AU : CSE : May-13

Ans. : In Java threads are used to handle multiple tasks together. This kind of programming is called concurrent programming.

Q.29 Name any four thread constructor.

AU : CSE : May-13

Ans. :

1. Thread()
2. Thread(String name)
3. Thread(Runnable target)
4. Thread(Runnable target, String name)

Q.30 When thread is initiated and created, what is its initial stage ?

AU : May 15

Ans. : A thread is in "Ready" state after it has been created and started. This state signifies that the thread is ready for execution. From here, it can be in the running state.

Q.31 "Thread is a lightweight process" – Comment

AU : Dec 19

Ans. : Threads do not require separate address for its execution. It runs in the address space of the process to which it belongs to. Hence thread is a lightweight process.

Q.32 Sketch the life cycle of thread

AU : May 19

Ans. : Refer Fig. 3.11.1.

UNIT IV

4

I/O, Generics, String Handling

Syllabus

I/O Basics - Reading and Writing Console I/O - Reading and Writing Files. Generics : Generic Programming - Generic classes - Generic Methods - Bounded Types - Restrictions and Limitations. Strings : Basic String class, methods and String Buffer Class.

Contents

Part I : Input and Output

- | | | | |
|-----|--|-------------------------------|-----------------|
| 4.1 | <i>I/O Basics.....</i> | <i>May-13, 15,</i> | <i>Marks 8</i> |
| 4.2 | <i>Reading and Writing Console I/O</i> | | |
| 4.3 | <i>Reading and Writing Files</i> | <i>May-19, Dec.-19,</i> | <i>Marks 13</i> |

Part II : Generics

- | | | | |
|-----|-------------------------------------|----------------------------|-----------------|
| 4.4 | <i>Generic Programming.....</i> | <i>Dec.-13,.....</i> | <i>Marks 4</i> |
| 4.5 | <i>Generic Methods.....</i> | <i>May-15, 19,</i> | <i>Marks 8</i> |
| 4.6 | <i>Generic Classes.....</i> | <i>May-12,13,14,</i> | |
| | | <i>Dec.-10,12,14</i> | <i>Marks 16</i> |
| 4.7 | <i>Bounded Types</i> | | |
| 4.8 | <i>Restrictions and Limitations</i> | | |

Part III : Strings

- | | | | |
|------|---|----------------------|----------------|
| 4.9 | <i>Basic String Class</i> | | |
| 4.10 | <i>String Methods.....</i> | <i>Dec.-13,.....</i> | <i>Marks 8</i> |
| 4.11 | <i>StringBuffer Class</i> | | |
| 4.12 | <i>Two Marks Questions with Answers</i> | | |

Part I : Input and Output

4.1 I/O Basics

AU : May-13, 15, Marks 8

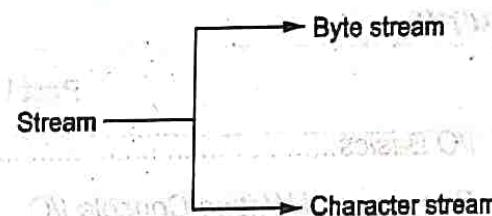
4.1.1 Stream

Definition :

- Stream is basically a channel on which the data flow from sender to receiver.
- An input object that reads the stream of data from a file is called **input stream**.
- The output object that writes the stream of data to a file is called **output stream**.

4.1.2 Byte Stream and Character Stream

- In Java input and output operations are performed using streams. And Java implements streams within the classes that are defined in `java.io`.
- There are basically two types of streams used in Java.



4.1.3 Byte Stream

- The byte stream is used for inputting or outputting the bytes.
- There are two super classes in byte stream and those are **InputStream** and **OutputStream** from which most of the other classes are derived.
- These classes define several important methods such as **read()** and **write()**.
- The input and output stream classes are as shown in Fig. 4.1.1 and Fig. 4.1.2.

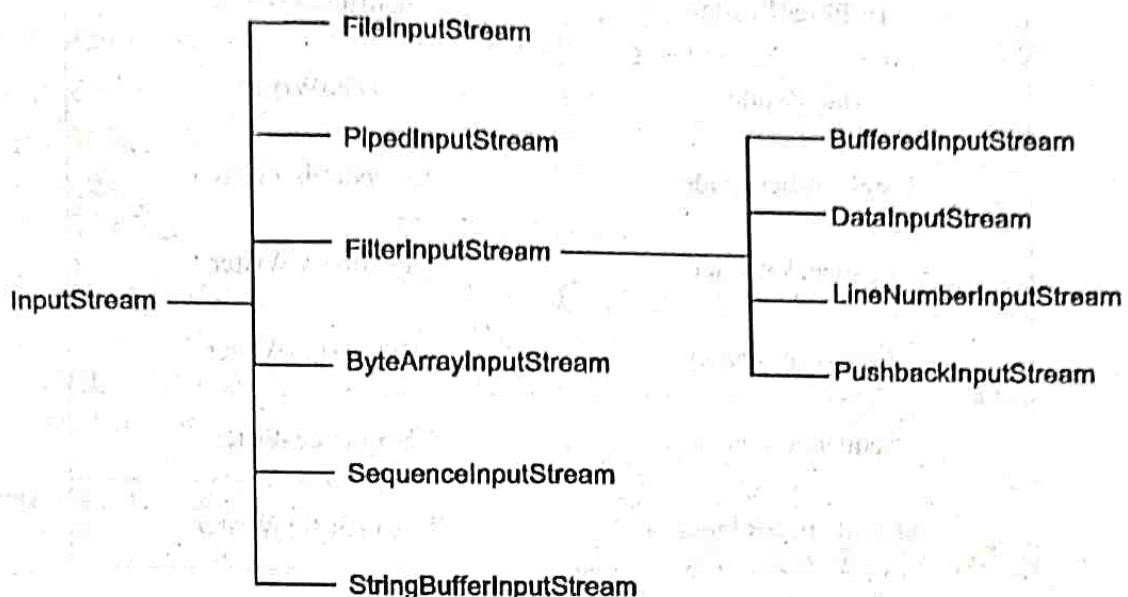


Fig. 4.1.1 InputStream classes

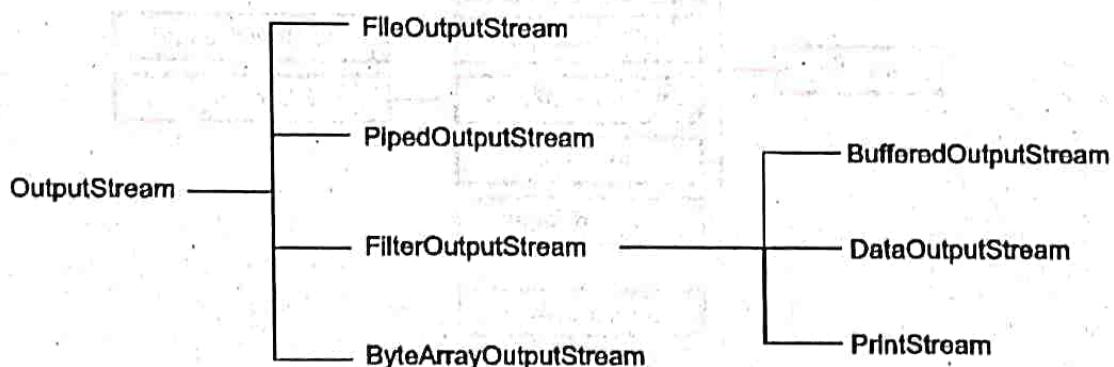


Fig. 4.1.2 OutputStream classes

4.1.4 Character Stream

- The character stream is used for inputting or outputting the characters.
- There are two super classes in character stream and those are **Reader** and **Writer**.
- These classes handle the Unicode character streams.
- The **two important methods** defined by the character stream classes are **read()** and **Write()** for reading and writing the characters of data.
- Various Reader and Writer classes are –

FileReader	FileWriter
PipeReader	PipeWriter
FilterReader	FilterWriter

BufferedReader	BufferedWriter
DataReader	DataWriter
LineNumberReader	LineNumberWriter
PushbackReader	PushbackWriter
ByteArrayReader	ByteArrayWriter
SequenceReader	SequenceWriter
StringBufferReader	StringBufferWriter

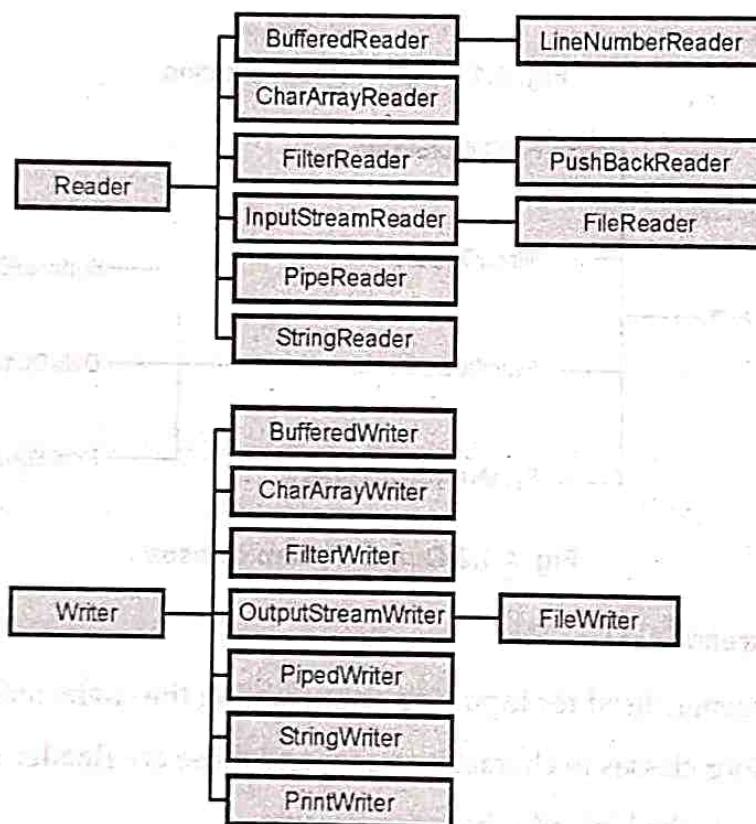


Fig. 4.1.3

Reader and **InputStream** define similar APIs but for different data types. For example, Reader contains these methods for reading characters and arrays of characters :

```

int read()
int read(char buf[])
int read(char buf[], int offset, int length)
  
```

InputStream defines the same methods but for reading bytes and arrays of bytes :

```

int read()
int read(byte buf[])
  
```

```
int read(byte buf[], int offset, int length)
```

Writer and OutputStream are similarly parallel. Writer defines these methods for writing characters and arrays of characters :

```
int write(int c)
int write(char buf[])
int write(char buf[], int offset, int length)
```

And OutputStream defines the same methods but for bytes :

```
int write(int c)
int write(byte buf[])
int write(byte buf[], int offset, int length)
```

4.1.5 Comparison between Byte Stream and Character Stream

Sr.No.	Byte Stream	Character Stream
1.	The byte stream is used for inputting and outputting the bytes.	The character stream is used for inputting and outputting the characters.
2.	There are two super classes used in byte stream and those are – InputStream and OutputStream	There are two super classes used in byte stream and those are – Reader and Writer
3.	A byte is a 8-bit number type that can represent values from – 127 to 127. Ascii values can be represented with exactly 7 bits.	A character is a 16 bit number type that represents Unicode.
4.	It never supports Unicode characters	It supports Unicode characters.

Review Questions

1. What is meant by stream ? What are the types of streams and classes ? Explain in detail.

AU : IT : May-13, Marks 8

2. Explain I/O streams with suitable example

AU : May-15, Marks 8

4.2 Reading and Writing Console I / O

4.2.1 Reading Console Input

- In this section, we will understand how to *read the input from console?*
- In Java, System is a class defined in java.lang and in, out and err are the variables of the class System.

- Hence `System.out` is an object used for standard output stream and `System.in` and `System.err` are the objects for standard input stream and error.

Method 1 : Reading input using Buffered Reader, InputStreamReader, system. in

- When we want to read some character from the console we should make use of `System.in`.
- The character stream class `BufferedReader` is used for this purpose. In fact we should pass the variable `System.in` to `BufferedReader` object. Along with it we should also mention the abstract class of `BufferedReader` class which is `InputStreamReader`.
- Hence the syntax is,

```
BufferedReader object_name =new  
BufferedReader(new InputStreamReader(System.in));
```

Enables us to read
console input

For example the object named `obj` can be created as

```
BufferedReader obj =new BufferedReader(new  
InputStreamReader(System.in));
```

- Then, using this object we invoke `read()` method. For e.g. `obj.read()`.
- But this is not the only thing needed for reading the input from console; we have to mention the exception `IOException` for this purpose. Hence if we are going to use `read()` method in our program then function `main` can be written like this –
`public static void main(String args[]) throws IOException`
- Let us understand all these complicated issues with the help of some simple Java program

Java program [ReadChar.java]

```
/*  
This is a java program which is for reading the input from console  
*/  
import java.io.*;  
class ReadChar  
{  
    public static void main(String args[])  
    throws IOException  
    {  
        //declaring obj for read() method  
        ↓  
        BufferedReader obj=new BufferedReader(new InputStreamReader(System.in));  
        int count=0;  
        char ch;
```

Step 1 : Creating instance
for reading console input

```

System.out.println("\n Enter five characters");
while(count<5)
{
    ch=(char)obj.read(); //reading single character
    System.out.println(ch); //outputting it
    count++; //count to keep track of 5 characters
}
}
}

```

Step 2 : Actually reading character

Output

```

Enter five characters
hello
h
e
l
l
o

```

Program Explanation

- This program allows you take the input from console.
- As given in above program, **read()** method is used for reading the input from the console.
- The method **read()** returns the integer value, hence it is typecast to **char** because we are reading characters from the console.
- And last but not least we should write,

import java.io.*;
in our java program, because these I/O operations are supported by the java package **java.io**.
Let us discuss one more example of reading input from console.

Ex. 4.2.1 : Write a Java program to read entire line through keyboard.

Sol. : Java Program [ReadString.java]

```

/*
This is a java program which is for reading the input from console
*/
import java.io.*;
class ReadString
{
    public static void main(String args[])
        throws IOException
    {
        BufferedReader obj=new BufferedReader(new InputStreamReader(System.in));
        String s;
        s=obj.readLine(); //for reading the string
        while(!s.equals("end"))

```

Reading input using BufferedReader
InputStreamReader and System.in

```
{\n    System.out.println(s);\n    s=obj.readLine();\n}\n}\n}
```

Output

hello how are you
hello how are you
end

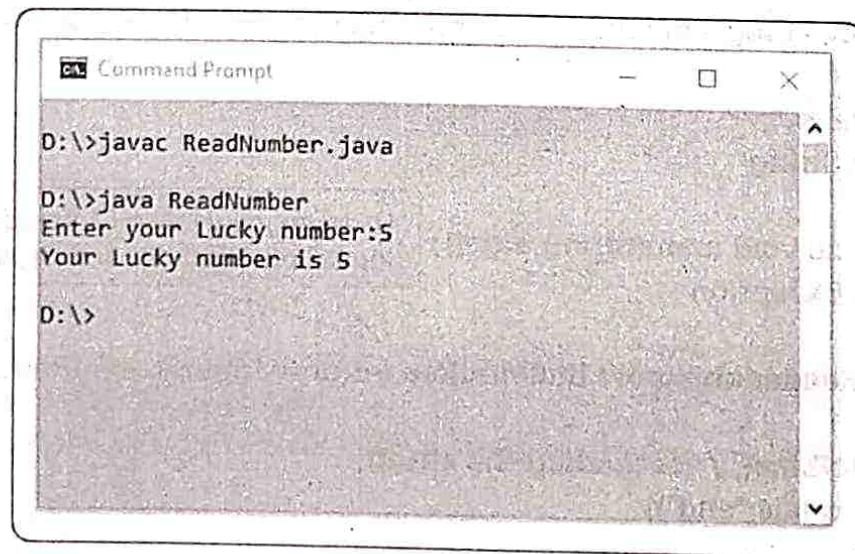
Note that to read the string input we have used `readLine()` function. Otherwise this program is almost same as previous one.

Ex. 4.2.2 : Write a Java Program by which user can enter his/her lucky number using keyboard. This number should also be displayed on the console.

Sol. :-

```
import java.io.*;
public class ReadNumber
{
    public static void main(String[] args) throws IOException
    {
        String s;
        int num;
        BufferedReader reader; //create BufferedReader object
        reader = new BufferedReader(new InputStreamReader(System.in));
        System.out.print("Enter your Lucky number:");
        s = reader.readLine();
        num = Integer.parseInt(s); //string value is converted to integer value
        System.out.println("Your Lucky number is " + num);
    }
}
```

Output



Method 2 : Reading Console input using Scanner and System.in

- The Scanner is a class defined in `java.util` package. With the help of `System.in` we can read the console input by creating the instance for the class Scanner.
- The steps to read the console input are as follows
 - Step 1 :** Import `java.util` package
 - Step 2 :** Create instance for Scanner class as follows

```
Scanner scanner = new Scanner(System.in);
```

- Step 3 :** Invoke the `nextXX()` method to read the input of specific data type

- Example 1 : Reading String**

```
import java.util.*; //Step 1
public class Example
{
    public static void main(String[] args)
    {
        Scanner scanner = new Scanner(System.in); //Step 2
        System.out.print("What is your Name? ");
        String name = scanner.next(); //Step 3
        System.out.println("My Name is: " + name);
    }
}
```

Output

What is your Name? Ankita
My Name is: Ankita

- Example 2 : Reading integer**

```
import java.util.*;
public class Example
{
    public static void main(String[] args)
    {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter your Lucky number");
        int num = scanner.nextInt();
        System.out.println("My Lucky number is: " + num);
    }
}
```

Output

Enter your Lucky number 5
My Lucky number is: 5

- Example 3 : Reading float**

```
import java.util.*;
```

```
public class Example
```

```
{
    public static void main(String[] args)
    {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter your marks ");
        float num = scanner.nextFloat();
        System.out.println("My marks are: " + num);
    }
}
```

- **Example 4 : Reading double**

```
import java.util.*;
public class Example
{
    public static void main(String[] args)
    {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter double number ");
        double num = scanner.nextDouble();
        System.out.println("You have entered: " + num);
    }
}
```

Output

Enter double number 212.444
 You have entered : 212.444

- **Example 5 : Reading boolean**

```
import java.util.*;
public class Example
{
    public static void main(String[] args)
    {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter boolean value as true or false ");
        boolean b = scanner.nextBoolean();
        System.out.println("You have entered: " + b);
    }
}
```

Output

Enter boolean value as true or false false
 You have entered: false

Ex. 4.2.3 : Write a Java program for calculating the bill amount. User enters the price of the item and its quantity.

Sol:

```

import java.util.*;
public class Example
{
    public static void main(String[] args)
    {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the price: ");
        float price = scanner.nextFloat();
        System.out.print("Enter the quantity: ");
        int quantity = scanner.nextInt();
        float amount;
        amount = price * quantity;
        System.out.println("\nPrice: " + price);
        System.out.println("Quantity: " + quantity);
        System.out.println("-----");
        System.out.println("Total Bill in Rs: " + amount);
        System.out.println("-----");
    }
}

```

Output

Enter the price: 20.50

Enter the quantity: 10

Price: 20.5

Quantity: 10

Total Bill in Rs: 205.0

4.2.2 Writing Console Output

The simple method used for writing the output on the console is `write()`. Here is the syntax of using `write()` method -

```
System.out.write(int b)
```

The bytes specified by `b` has to be written on the console. But typically `print()` or `println()` is used to write the output on the console. And these methods belong to `PrintWriter` class.

4.3 Reading and Writing Files

AU : May-19, Dec-19, Marks 13

`InputStream` is an abstract class for streaming the byte input. Various methods defined by this class are as follows.

Methods	Purpose
int available()	It returns total number of byte input that are available currently for reading.
void close()	The input source gets closed by this method. If we try to read further after closing the stream then IOException gets generated.
void mark(int n).	This method places a mark in the input stream at the current point. This mark remains valid until the n bytes are read.
boolean markSupported()	It returns true if mark() or reset() are supported by the invoking stream.
void reset()	This method resets the input pointer to the previously set mark.
long skip(long n)	This method ignores the n bytes of input. It then returns the actually ignored number of bytes.
int read()	It returns the next available number of bytes to read from the input stream. If this method returns the value -1 then that means the end of file is encountered.
int read(byte buffer[])	This method reads the number of bytes equal to length of <i>buffer</i> . Thus it returns the actual number of bytes that were successfully read.
int read(byte buffer[], int offset, int n)	This method returns the n bytes into the <i>buffer</i> which is starting at offset. It returns the number of bytes that are read successfully.

OutputStream is an abstract class for streaming the byte output. All these methods under this class are of void type. Various methods defined by this class are as follows

Methods	Purpose
void close()	This method closes the output stream and if we try to write further after using this method then it will generate IOException
void flush()	This method clears output buffers.
void write(int val)	This method allows to write a single byte to an output stream.

<code>void write(byte buffer[])</code>	This method allows to write complete array of bytes to an output stream.
--	--

<code>void write(byte buffer[], int offset, int n)</code>	This method writes n bytes to the output stream from an array $buffer$ which is beginning at $buffer[offset]$
---	---

4.3.1 FileInputStream / FileOutputStream

The `FileInputStream` class creates an `InputStream` using which we can read bytes from a file. The two common constructors of `FileInputStream` are -

`FileInputStream(String filename);`
`FileInputStream(File fileobject);`

In the following Java program, various methods of `FileInputStream` are illustrated -

Java Program[FileStreamProg.java]

```
import java.io.*;
class FileStreamProg
{
    public static void main(String[] args) throws Exception
    {
        int n;
        InputStream fobj = new FileInputStream("f:/I_O_programs/FileStreamProg.java");
        System.out.println("Total available bytes: " + (n = fobj.available()));
        int m = n - 400;
        System.out.println("\n Reading first " + m + " bytes at a time");
        for (int i = 0; i < m; i++)
            System.out.print((char) fobj.read());
        System.out.println("\n Skipping some text");
        fobj.skip(n / 2);
        System.out.println("\n Still Available: " + fobj.available());
        fobj.close();
    }
}
```

Output

```

Command Prompt
F:\I_O_programs>javac FileStreamProg.java
F:\I_O_programs>java FileStreamProg
Total available bytes: 569
Reading first 169 bytes at a time
import java.io.*;
class FileStreamProg
{
    public static void main(String[] args) throws Exception
    {
        int n;
        InputStream fobj=new FileInputStream("f:/I_O_program
Skipping some text
Still Available: 116
F:\I_O_programs>

```

Reading this much text from the file

The **FileOutputStream** can be used to write the data to the file using the **OutputStream**. The constructors are -

```

FileOutputStream(String filename)
FileOutputStream(Object fileobject)
FileInputStream(String filename,boolean app);
FileInputStream(String fileobject,boolean app);

```

The *app* denotes that the append mode can be true or false. If the append mode is true then you can append the data in the file otherwise the data can not be appended in the file.

In the following Java program, we are writing some data to the file.

Java Program[FileOutStreamProg.java]

```

import java.io.*;
class FileOutStreamProg
{
    public static void main(String[] args) throws Exception
    {
        String my_text="India is my Country\n"+"and I love my country very much.";
        byte b[]=my_text.getBytes();
        OutputStream fobj=new FileOutputStream("f:/I_O_programs/output.txt");
        for(int i=0;i<b.length;i++)
            fobj.write(b[i]);
        System.out.println("\n The data is written to the file");
        fobj.close();
    }
}

```

Output

```

Command Prompt
F:\I_O_Programs>javac FileOutputStreamProg.java
F:\I_O_Programs>java FileOutputStreamProg
The data is written to the file
F:\I_O_Programs>type output.txt
India is my Country
and I love my country very much.
F:\I_O_Programs>_

```

Ex. 4.3.1 : Write a program that copies the content of one file to another by removing unnecessary spaces between words

Sol. :

```

import java.io.*;
public class CopyFile
{
    private static void CopyDemo(String src, String dst)
    {
        try
        {
            File f1 = new File(src);
            File f2 = new File(dst);
            InputStream in = new FileInputStream(f1);
            OutputStream out = new FileOutputStream(f2);
            byte[] buff = new byte[1024];
            int len;
            len=in.read(buff);
            while (len > 0)
            {
                String text=new String(buff);//converting bytes to text
                text = text.replaceAll("\\s+", " ");//removing unnecessary spaces
                buff=text.getBytes();//converting that text back to bytes
                out.write(buff,0,len);//writing bytes to destination file
                len=in.read(buff);//reading the remaining content of the file
            }
            in.close();
            out.close();
            System.out.println("File copied.");
        }
        catch(FileNotFoundException ex)
    }
}

```

```

{
    System.out.println(ex.getMessage() + " in the specified directory.");
    System.exit(0);
}
catch(IOException e)
{
    System.out.println(e.getMessage());
}
}
public static void main(String[] args)
{
    CopyDemo(args[0],args[1]);
}
}
}

```

Output

D:\test>javac CopyFile.java

D:\test>javac CopyFile.java

D:\test>java CopyFile in.txt out.txt

File copied.

D:\test>type in.txt

This is my India.

I love my country.

D:\test>type out.txt

This is my India. I love my country.

4.3.2 FilterInputStream / FilterOutputStream

FilterStream class are those classes which wrap the input stream with the byte. That means using input stream you can read the bytes but if you want to read integers, doubles or strings you need to a filter class to wrap the byte input stream.

The syntax for FilterInputStream and FilterOutputStream are as follows –

FilterOutputStream(OutputStream o)

FilterInputStream(InputStream i)

The methods in the Filter stream are similar to the methods in **InputStream** and **OutputStream**

4.3.2.1 DataInputStream / DataOutputStream

DataInputStream reads bytes from the stream and converts them into appropriate primitive data types whereas the **DataOutputStream** converts the primitive data types into the bytes and

then writes these bytes to the stream. The superclass of **DataInputStream** class is **FilterInputStream** and superclass of **DataOutputStream** class is **FilterOutputStream** class. Various methods under these classes are -

Methods	Purpose
boolean readBoolean()	Reads Boolean value from the inputstream.
byte readByte()	Reads byte value from the inputstream.
char readChar()	Reads char value from the inputstream.
float readFloat()	Reads float value from the inputstream.
double readDouble()	Reads double value from the inputstream.
int readInt()	Reads integer value from the inputstream.
long readLong()	Reads long value from the inputstream.
String readLine()	Reads string value from the inputstream.
String readUTF()	Reads string in UTF form.
void writeBoolean(boolean val)	Writes the Boolean value to output stream.
void writeBytes(String str)	Writes the bytes of characters in a string to output stream.
void writeFloat(float val)	Writes the float value to output stream.
void writeDouble(float val)	Writes the double value to output stream.
void writeInt(int val)	writes the integer value to output stream.
void writeLong(long val)	writes the long value to output stream.
void writeUTF(String str)	writes the string value in UTF form to output stream.

Let us see how these methods can be used in a Java Program -

Java Program[DataStreamProg.java]

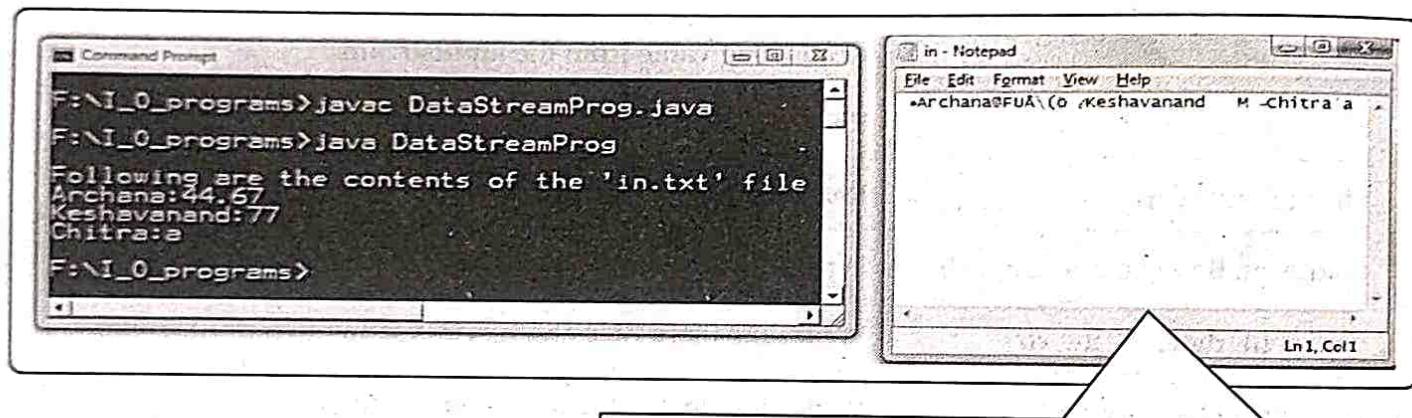
```
import java.io.*;
class DataStreamProg
{
    public static void main(String[] args) throws Exception
    {
        DataOutputStream obj=new DataOutputStream
            (new FileOutputStream("in.txt"));
        obj.writeUTF("Archana");
        obj.writeDouble(44.67);
```

```

        obj.writeUTF("Keshavanand");
        obj.writeInt(77);
        obj.writeUTF("Chitra");
        obj.writeChar('a');
        obj.close();
    DataInputStream in=new DataInputStream(new FileInputStream("in.txt"));
    System.out.println("\nFollowing are the contents of the 'in.txt' file");
    System.out.println(in.readUTF()+":"+in.readDouble());
    System.out.println(in.readUTF()+":"+in.readInt());
    System.out.println(in.readUTF()+":"+in.readChar());
}
}

```

Output



Open the **in.txt** file in the Notepad and you can see the contents that gets saved through your Java program

Program explanation

Note that in above program, along with other simple read and write methods we have used **readUTF** and **writeUTF** methods. Basically UTF is an encoding scheme that allows systems to operate with both ASCII and Unicode. Normally all the operating systems use ASCII and Java uses Unicode. The **writeUTF** method converts the string into a series of bytes in the UTF-8 format and writes them into a binary stream. The **readUTF** method reads a **string** which is written using the **writeUTF** method.

4.3.2.2 BufferedInputStream / BufferedOutputStream

The **BufferedInputStream** and **BufferedOutputStream** is an efficient class used for speedy read and write operations. All the methods of **BufferedInputStream** and **BufferedOutputStream** class are inherited from **InputStream** and **OutputStream** classes. But these methods add the buffers in the stream for efficient read and write operations. We can specify the buffer size otherwise the default buffer size is 512 bytes.

In the following Java program the **BufferedInputStream** and **BufferedOutputStream** classes are used.

Java Program[BUFFEREDSTREAMPROG.java]

```

import java.io.*;

class BufferedStreamProg
{
    public static void main(String[] args) throws Exception
    {
        DataOutputStream obj = new DataOutputStream(new BufferedOutputStream(new
FileOutputStream("in.txt")));
        obj.writeUTF("Archana");
        obj.writeDouble(44.67);
        obj.writeUTF("Keshavanand");
        obj.writeInt(77);
        obj.writeUTF("Chitra");
        obj.writeChar('a');
        obj.close();
        DataInputStream in = new DataInputStream(new BufferedInputStream(new
FileInputStream("in.txt")));
        System.out.println("\nFollowing are the contents of the 'in.txt' file");
        System.out.println(in.readUTF() + ":" + in.readDouble());
        System.out.println(in.readUTF() + ":" + in.readInt());
        System.out.println(in.readUTF() + ":" + in.readChar());
    }
}

```

Note that the **Buffered stream class is added here.**

Program Explanation

This program is similar to the Java program we have discussed in section 5.4.2.1. The only change is we have added buffers. The creation of buffers is shown by the bold faced statements in above program.

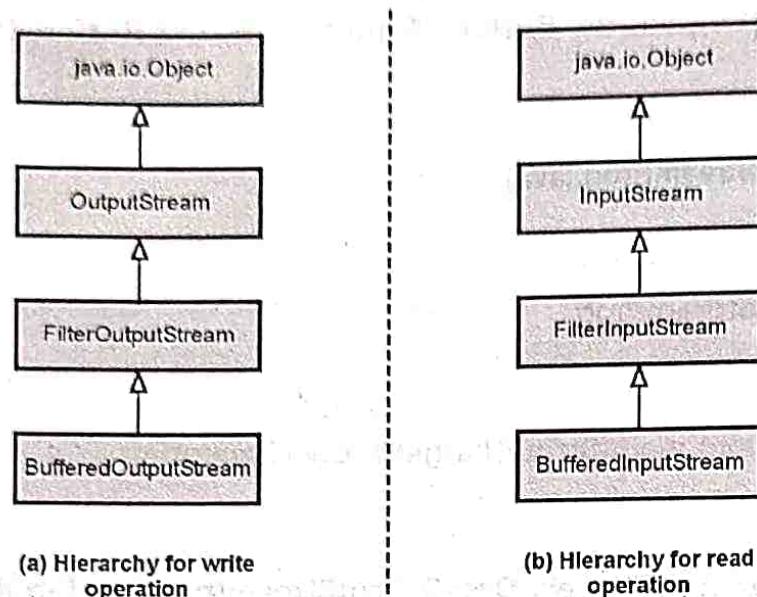


Fig. 4.3.1 Inheritance hierarchy

4.3.3 Programming Example on Reading and Writing Files

Ex. 4.3.2 : Write a program to replace all "word1" by "word2" from a file 1, and output is written to file2 file and display the number of replacement.

Sol. :

```

import java.io.*;
import java.util.*;
class FileDemo
{
    public static void main(String[] args)throws Exception
    {
        int count=0;
        File fin=new File("C:/lab/word.txt");
        BufferedReader br=new BufferedReader(new FileReader(fin));

        String FullStr="";
        String TempStr;
        while((TempStr=br.readLine())!=null)//if multi-line input is present in the file
        {
            //then it is collected in one string separated by spaces
            FullStr+=TempStr+" ";
        }
        Scanner input=new Scanner(System.in);
        System.out.print("Enter the word to be searched from the file: ");
        String word1=input.next();

        String Word_List[]=FullStr.split(" ")//Extract words from string for counting
                                            //no.of occurrences

        //Searching for the word1
    }
}

```

```

for(int i=0;i<Word_List.length;i++)
{
    if(Word_List[i].equals(word1))
    {
        count++; //counting no. of occurrences
    }
}
System.out.print("\nEnter new Word for replacement:");
String word2=input.next();

String New_Str=FullStr.replace(word1,word2);

//File is opened for writing purpose
File fout=new File("C:/lab/Newword.txt");
BufferedWriter bw=new BufferedWriter(new FileWriter(fout));
//Modified contents are written to the file
bw.write(New_Str);

System.out.println("\n The replacement is done and file is rewritten");
System.out.println("\nThe number of replacements are "+count);
bw.close();

br=new BufferedReader(new FileReader(fout));
System.out.println("\n Now, the Contents of the file are...\n");
while((TempStr=br.readLine())!=null)
{
    System.out.println(TempStr);
    br.close();
}

}
*****
<word.txt>
*****
```

PHP is fun.

Programming in PHP is interesting.

People love PHP

Output

Enter the word to be searched from the file: PHP

Enter new Word for replacement: Java

The replacement is done and file is rewritten

The number of replacements are 3

Now, the Contents of the file are...

Java is fun. Programming in Java is interesting. People love Java

Ex. 4.3.3 : Write a program to replace all "word1" by "word2" to a file without using temporary file and display the number of replacement.

Sol. :

```

import java.io.*;
import java.util.*;
class FileDemo
{
    public static void main(String[] args) throws Exception
    {
        int count=0;
        File fname=new File("C:/lab/word.txt");
        BufferedReader br=new BufferedReader(new FileReader(fname));
        String FullStr="";
        String TempStr;
        while((TempStr=br.readLine())!=null)//if multi-line input is present in the file
        {
            FullStr+=TempStr+" "; //then it is collected in one string separated by spaces
        }
        Scanner input=new Scanner(System.in);
        System.out.print("Enter the word to be searched from the file: ");
        String word1=input.next();
        String Word_List[]=FullStr.split(" "); //Extract words from string for counting
                                                //no.of occurrences
        //Searching for the word1
        for(int i=0;i<Word_List.length;i++)
        {
            if(Word_List[i].equals(word1))
            {
                count++; //counting no. of occurrences
            }
        }
        System.out.print("\nEnter new Word for replacement: ");
        String word2=input.next();
        String New_Str=FullStr.replace(word1,word2);
        //File is opened for writing purpose
        BufferedWriter bw=new BufferedWriter(new FileWriter(fname));
        //Modified contents are written to the file
        bw.write(New_Str);
        System.out.println("\n The replacement is done and file is rewritten");
        System.out.println("\nThe number of replacements are "+count);
        bw.close();
    }
}

```

```

        br = new BufferedReader(new FileReader(fname));
        System.out.println("\n Now, the Contents of the file are...\n");
        while((TempStr = br.readLine())!=null)
            System.out.println(TempStr);
        br.close();
    }
}

```

Ex. 4.3.4 : Write a program that takes input for filename and search word from command-line arguments and checks whether that file exists or not. If exists, the program will display those lines from a file that contains given search word.

Sol.:

```

import java.io.*;
class WordSearchDemo extends java.lang.Object
{
    public static void main(String args[]) throws IOException
    {
        boolean status;
        boolean WordOccur=false;
        File fobj = new File(args[0]);
        if(fobj.exists())
            status = true;
        else
        {
            status=false;
            System.out.println("\n This file does not exist!!!");
            return;
        }
        if(status)
        {
            BufferedReader in = new BufferedReader(new FileReader(fobj));
            String Line;
            String key = args[1];
            while ((Line = in.readLine()) != null)
            {
                String[] Data = Line.split(" ");
                for(String Element:Data)
                {
                    if(Element.equalsIgnoreCase(key))
                }
            }
        }
    }
}

```

```

        System.out.println(key + " is present in the line " + Line);
        WordOccur = true;
    }
}
if(!WordOccur)
    System.out.println("This Word is not present in the Input File");
}
}
}

```

Output

E:\test\src>javac WordSearchDemo.java
E:\test\src>java WordSearchDemo input.dat java
'java' is present in the line 'learn Java Programming.'

Ex. 4.3.5 : Write a program that counts the no. of words in a text file. The file name is passed as a command line argument. The program should check whether the file exists or not. The words in the file are separated by white space characters.

Sol. :

```

import java.io.*;
import java.util.*;

public class WordCountDemo
{
    public static void main(String[] args) throws NullPointerException, IOException
    {

        try
        {
            BufferedReader in = new BufferedReader(new FileReader(args[0]));
            String New_Line = "", Word = "";
            int Word_Count = 0;
            while ((New_Line = in.readLine()) != null)

                Word += New_Line + " ";
            StringTokenizer Token = new StringTokenizer(Word);
            while (Token.hasMoreTokens())
            {
                String s = Token.nextToken();
                Word_Count++;
            }
            System.out.println("Text file contains " + Word_Count + " words.");
        }
    }
}

```

```

        catch (NullPointerException e)
        {System.out.println("NULL PointerError: "+e.getMessage());}
        catch (IOException e)
        {System.out.println("IO Error: " + e.getMessage());}
    }
}

<input.txt>
Hello Friends
It is very interesting to
learn Java Programming.

```

Output

E:\test\src>javac WordCountDemo.java

E:\test\src>java WordCountDemo input.txt

Text file contains 10 words.

Ex. 4.3.6 : Write a program to count the total no. of chars, words, lines, alphabets, digits, white spaces in a given file.

Sol. :

```

import java.lang.*;
import java.io.*;
import java.util.*;
class WordCount
{
    public static void main(String arg[]) throws Exception
    {
        int char_count=0;
        int word_count=0;
        int line_count=0;
        int wspace_count=0;
        int digit_count=0;
        int alphabet_count=0;

        String fname;
        String temp_str;
        StringTokenizer token;

        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        System.out.print("Enter filename : ");
        fname=br.readLine();

        br=new BufferedReader(new FileReader(fname));
        while((temp_str=br.readLine())!=null)
        {

```

```

line_count++;
    for(int i = 0; i < temp_str.length(); i++)
    {
        if(Character.isWhitespace(temp_str.charAt(i)))
            wspace_count++;
        if(Character.isDigit(temp_str.charAt(i)))
            digit_count++;
        if(Character.isLetter(temp_str.charAt(i)))
            alphabet_count++;
    }

token=new StringTokenizer(temp_str, " ");
while(token.hasMoreTokens())
{
    word_count++;
    String s=token.nextToken();
    char_count+=s.length();
}

System.out.println("Character Count :" +char_count);
System.out.println("Word Count :" +word_count);
System.out.println("Line Count :" +line_count);
System.out.println("Alphabet Count :" +alphabet_count);
System.out.println("Digit Count :" +digit_count);
System.out.println("White Space Count :" +wspace_count);

br.close();
}
}

```

Sample input file: inputFile.txt
Hello123
How are u?
I am fine.
Enjoying my job

Output
Enter filename : inputFile.txt
Character Count : 36
Word Count : 10
Line Count: 4
Alphabet Count : 32
Digit Count : 3
White Space Count : 6

Ex. 4.3.7 Create an IN file in Java to store the details of 100 students using Student class. Read the details from IN file, Convert all the letters of IN file to lowercase letters and write it to OUT file.

Sol. :

AU : May-19, Marks 13

Step 1 : Create a Student class in a separate Java file

```

import java.io.Serializable;
public class Student implements Serializable {

```

```

//default serialVersion id
private static final long serialVersionUID = 1L;

private String first_name;
private String last_name;
private int age;

public Student(String fname, String lname, int age){
    this.first_name = fname;
    this.last_name = lname;
    this.age = age;
}

public void setFirstName(String fname) {
    this.first_name = fname;
}
public String getFirstName() {
    return this.first_name;
}
public void setLastName(String lname) {
    this.last_name = lname;
}
public String getLastName() {
    return this.last_name;
}
public void setAge(int age) {
    this.age = age;
}
public int getAge() {
    return this.age;
}
@Override
public String toString() {
    return new StringBuffer("\t").append(this.first_name)
        .append("\t").append(this.last_name).append("\t").append(this.age).toString();
}

```

Step 2 : Create another java file for performing file handling operations.

StudentDemo.java

```

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.Scanner;
public class StudentDemo
{
    private static final String filepath="D:\\IN.txt";
    public static void main(String args[])
    {
        StudentDemo objectIO = new StudentDemo();
        try
        {
            FileOutputStream fileOut = new FileOutputStream("D:\\IN.txt");
            ObjectOutputStream objectOut = new ObjectOutputStream(fileOut);
            for(int i=0;i<100;i++)
            {
                Scanner sc = new Scanner(System.in);
                System.out.println("Enter first name of the student: ");
                String fname= sc.next();
                System.out.println("Enter last name of the student: ");
                String lname= sc.next();
                System.out.println("Enter Age of the student: ");
                int age= sc.nextInt();
                Student student = new Student(fname,lname,age);
                objectOut.writeObject((Object)student);
            }
            System.out.println(" The File is created Successfully ");
            objectOut.close();
        }
        catch (Exception ex)
        {
            ex.printStackTrace();
        }
    }
    //Read object from IN.txt and write to OUT.txt
    Student st;
    try
    {
        FileInputStream fileIn = new FileInputStream("D:\\IN.txt");
        ObjectInputStream objectIn = new ObjectInputStream(fileIn);
        FileOutputStream fileOut = new FileOutputStream("D:\\OUT.txt");
        ObjectOutputStream objectOut = new ObjectOutputStream(fileOut);
    }
}

```

```

Object obj;
for(int i=0;i<100;i++)
{
    obj = objectIn.readObject(); //reading object
    st = (Student)obj; //converting it to Student type
    String fn = st.getFirstName().toLowerCase();
    String ln = st.getLastName().toLowerCase();
    Student student = new Student(fn,ln,st.getAge());
    objectOut.writeObject((Object)student); //writing the modified object
}
System.out.println("The File is copied Successfully");
objectIn.close();
objectOut.close();
}
catch (Exception ex)
{
    ex.printStackTrace();
}
//Read object from OUT.txt
try
{
    FileInputStream fileIn = new FileInputStream("D:\\OUT.txt");
    ObjectInputStream objectIn = new ObjectInputStream(fileIn);
    Object obj;
    System.out.println(" FirstName LastName Age");
    for(int i=0;i<100;i++)
    {
        obj = objectIn.readObject();
        System.out.println((Student)obj);
    }
    objectIn.close();
}
catch (Exception ex)
{
    ex.printStackTrace();
}
}
}

```

Review Question

1. Explain in detail about the following with sample program :
 - i) Reading from a file ii) Writing in a file.

AU : Dec.-19, Marks 13

Part II : Generics

4.4 Generic Programming

AU : Dec.-13, Marks 4

Generic is a mechanism for creating a general model in which generic methods and generic classes enable programmers to specify a single method (or a set of related methods) and single class (or a set of related classes) for performing the desired task.

For example -

Suppose we want to create a stack and if we create a stack of integers then it will store only the integer elements, if we try to push any string or any double type element then a compile time error will occur. If we want to push any string then we need to create a separate stack, separate class and separate methods for handling the String elements. Same is true for the elements of any other data type. This results in complex code building. To avoid such complexity, a concept of generic programming is introduced.

What is the need for Generic?

Following features show the importance of generics in Java.

1. It saves the programmers burden of creating separate methods for handling data belonging to different data types.
2. It allows the code reusability.
3. Compact code can be created.

Review Question

1. Mention the motivations of generic programming

AU : Dec.-13, Marks 4

4.5 Generic Methods

AU : May-15, 19, Marks 8

Generic method allows a programmer to write a generalised method for the methods of different data types.

Suppose we want to print an array of integer, float and character type elements then normally we write three different methods performing the corresponding task. The object oriented feature of Java allows us to make use of same function name. This idea is illustrated in following Java program -

Java Program[OverloadProg1.java]

```
import java.io.*;
import java.util.*;
public class OverloadProg1
```

```

{
    public static void display(float[] a)
    {
        for(int i=0;i<5;i++)
            System.out.printf(" %.2f",a[i]);
    }

    public static void display(int[] a)
    {
        for(int i=0;i<5;i++)
            System.out.printf(" %d",a[i]);
    }

    public static void display(char[] a)
    {
        for(int i=0;i<5;i++)
            System.out.printf(" %c",a[i]);
    }

    public static void main(String[] args)
    {
        float[] dbl_a={11,22,33,44,55};
        int[] int_a={1,2,3,4,5};
        char[] char_a={'A','B','C','D','E'};

        System.out.println("\n The Float elements are ...");
        display(dbl_a);
        System.out.println("\n The Integer elements are ...");
        display(int_a);
        System.out.println("\n The character elements are ...");
        display(char_a);
    }
}

```

Note that the name
of these three
functions is same
i.e. display

Output

The Float elements are ...

11.00 22.00 33.00 44.00 55.00

The Integer elements are ...

1 2 3 4 5

The character elements are ...

A B C D E

But if we override the generic methods then the code becomes more simplistic –

Java Program[OverloadProg2.java]

```

import java.io.*;
import java.util.*;
public class OverloadProg2
{
    public static < T > void display(T[] a)          //Generic Method is created
    {
        for(int i=0;i<5;i++)
    }
}

```

```

        System.out.printf(" %s",a[i]);
    }
    public static void main(String[] args)
    {
        float[] dbl_a={11,22,33,44,55};
        int[] int_a={1,2,3,4,5};
        char[] char_a={'A','B','C','D','E'};
        System.out.println("\n The Float elements are ...");
        display( dbl_a );
        System.out.println("\n The Integer elements are ...");
        display( int_a );
        System.out.println("\n The character elements are ...");
        display( char_a );
    }
}

```

Output

The Float elements are ...
11.00 22.00 33.00 44.00 55.00
The Integer elements are ...
1 2 3 4 5
The character elements are ...
A B C D E

Ex. 4.5.1 : Write generic method for sorting an array of integer objects.**AU : May-19, Marks 6****Sol. :**

```

private <E extends Comparable<E>>void bubbleSortG(E[] Arr) {
    E temp;
    for(int j = 1; j < Arr.length; j++) {
        for(int i = 0; i < Arr.length - j; i++) {
            if(Arr[i].compareTo(Arr[i+1]) > 0) {
                temp = Arr[i];
                Arr[i] = Arr[i+1];
                Arr[i+1] = temp;
            }
        }
    }
    for(E i: Arr) {
        System.out.print(" " + i);
    }
}

```

Review Question

1. Explain about generic method with suitable example.

AU : May-15, Marks 8

4.6 Generic Classes

AU : May-12, 13, 14, Dec.-10, 12, 14, Marks 16

A generic class contains one or more variables of generic data type. Following is a simple example which shows how to define the generic class .

```
public class Test<T>
{
    public Test(){val=null;}
    public Test(T val)
    {
        this.val=val;
    }
    public getVal()
    {
        return val;
    }
    public setVal()
    {
        val=newValue;
    }
    private T val; //variable defined as of generic type
}
```

The concept of generic class supports the idea of type-independency. Typical example of data structure is stack. We can create a generic class for stack which allows us to insert integer, character or any other data type elements using the same push and pop methods.

Ex. 4.6.1 : Create a generic class for the stack data structure. Your class must handle integer and character type elements. Show clearly how will you handle the stack empty condition

Sol. : We will create a generic class for stack data structure using following steps -

Step 1 : Create a Java file named Stack.java as follows -

Java Program[Stack.java]

```
import java.util.*; //Supports the ArrayList
public class Stack<T> //T denotes any data type
{
    public ArrayList<T> obj;
    public Stack(int size) //Constructor will be invoked from main
    {
        obj=new ArrayList<T>(size);
    }
    public void push(T item) //Generic method for PUSH operation
    {
        obj.add(item);
    }
    public T pop()//Generic method for POP operation
```

```

{
    if(obj.isEmpty())
    {
        System.out.println("\n Stack is Empty");
        return null;
    }
    return obj.remove(obj.size()-1);
}
}

```

Step 2 : Create another Java program in a separate file named **StackGeneric.java**. It is as given below -

Java Program[StackGeneric.java]

```

import java.io.*;
import java.util.*;
public class StackGeneric
{
    public static void main(String[] args)
    {
        int[] iArray={1,2,3,4,5};
        char[] cArray={'A','B','C','D','E'};
    }
}

```

Declaring integer and character values to be pushed onto the stack.

```

Stack<Integer> ist=new Stack<Integer>(5);
Stack<Character> cst=new Stack<Character>(5);

```

ist is an instance for integer stack and cst is an instance for character stack.
The Constructor Stack(size) will be invoked.

```

System.out.println("\n Pushing the elements in integer stack");
for(int i=0;i<5;i++)
    ist.push(iArray[i]);

```

Pushing onto the integer stack

```

System.out.println("\n Pushing the elements in character stack");
for(int i=0;i<5;i++)
    cst.push(cArray[i]);

```

Pushing onto the character stack

```

System.out.println("\n Popping two elements from character stack");
for(int i=0;i<2;i++)
    System.out.printf("\n%c",cst.pop());

```

Popping from the character stack

```

System.out.println("\n Popping all the elements from integer stack");
for(int i=0;i<5;i++)
    System.out.printf("\n%d",ist.pop());

```

Popping from the integer stack

```

        System.out.println("\n Performing one more pop for integer stack");
        System.out.printf("\n%d",ist.pop());
    }
}

```

Output

F:\>javac StackGeneric.java
F:\>java StackGeneric

This pop operation
is to handle the
stack Empty
condition

Pushing the elements in integer stack

Pushing the elements in character stack

Popping two elements from character stack

E
D
Popping all the elements from integer stack
5
4
3
2
1
Performing one more pop for integer stack

Stack is Empty

null

Program Explanation

In the step 1 we have created a separate Java file, in which a class is written for defining the generic methods push and pop. Note that it is necessary to define the constructor for this class because it will then allow to initialize the class of appropriate data type.

In step 2 we are first creating the separate instances for each of these classes say cst and ist. Then using these instances the generic push and pop methods will be invoked.

The output given in the step 2 is self explanatory.

Ex. 4.6.2 : Using generic classes, write a program to perform the following operations on an array i) Add an element in the beginning/middle/end ii) Delete an element from a given position

AU : May-14, Marks 16

Sol.:

```

import java.io.*;
import java.util.*; //Supports the ArrayList
class Arr<T> //T denotes any data type
{

```

```

public ArrayList<T> obj;
public Arr(int size) //Constructor will be invoked from main()
{
    obj=new ArrayList<T>(size);
}
public void insert(int index,T item) //Generic method for insert Operation
{
    obj.add(index,item);
}
public void display()
{
    System.out.print(" "+obj);
}
public T del(int index)//Generic method for delete Operation
{
    return obj.remove(index);
}
}
public class ArrayGeneric
{
    public static void main(String[] args)
    {
        int[] iArray={1,2,3,4,5};
        Arr<Integer> iobj=new Arr<Integer>(10);
        int i,index;
        System.out.println("\n Array of integers is ...");
        for(i=0;i<5;i++)
            iobj.insert(i,iArray[i]);
        iobj.display();
        System.out.println("\n Inserting the elements in integer Array");
        System.out.println("Enter the element to be inserted: ");
        Scanner sc=new Scanner(System.in);
        int item = sc.nextInt();
        System.out.println("Enter the index at which the element is to be inserted: ");
        index = sc.nextInt();
        iobj.insert(index,item);
        iobj.display();
        System.out.println("\n Enter the index of the element to be deleted: ");
        index = sc.nextInt();
        iobj.del(index);
        iobj.display();
        double[] dArray={11.11,22.22,33.33,44.44,55.55};
        Arr<Double> dobj=new Arr<Double>(10);
        System.out.println("\n Array of doubles is ...");
    }
}

```

```

for(i=0;i<5;i++)
    dobj.insert(i,dArray[i]);
dobj.display();
System.out.println("\n Inserting the elements in double Array");
System.out.println("Enter the element to be inserted: ");
sc=new Scanner(System.in);
double ditem = sc.nextDouble();
System.out.println("Enter the index at which the element is to be inserted: ");
index = sc.nextInt();
dobj.insert(index,ditem);
dobj.display();
System.out.println("\n Enter the index of the element to be deleted: ");
index = sc.nextInt();
dobj.del(index);
dobj.display();
}
}

```

Output

Array of integers is ...

[1, 2, 3, 4, 5]

Inserting the elements in integer Array

Enter the element to be inserted:

100

Enter the index at which the element is to be inserted:

2

[1, 2, 100, 3, 4, 5]

Enter the index of the element to be deleted:

4

[1, 2, 100, 3, 5]

Array of doubles is ...

[11.11, 22.22, 33.33, 44.44, 55.55]

Inserting the elements in double Array

Enter the element to be inserted:

111.222

Enter the index at which the element is to be inserted:

3

[11.11, 22.22, 33.33, 111.222, 44.44, 55.55]

Enter the index of the element to be deleted:

2

[11.11, 22.22, 111.222, 44.44, 55.55]

Review Questions

1. Explain the generic classes and generic methods with example.

AU : CSE : Dec.-10, 12, May-12, IT : May-13, Marks 16

2. Explain in detail about generic classes and methods in java with suitable example.

AU : IT : May-13, Marks 16

3. Develop a Java program that will illustrate the use of Generic classes. Give self explanatory comments in your program

AU : Dec.-14, Marks 8

4.7 Bounded Types

- While creating objects to generic classes we can pass any derived type as type parameters.
- Many times it will be useful to limit the types that can be passed to type parameters. For that purpose, bounded types are introduced in generics.
- Using bounded types, we can make the objects of generic class to have data of specific derived types.
- For example, If we want a generic class that works only with numbers (like int, double, float, long) then declare type parameter of that class as a bounded type to Number class. Then while creating objects to that class you have to pass only Number types or it's subclass types as type parameters.
- The syntax for declaring Bounded type parameters is

<T extends SuperClass>

- This specifies that 'T' can only be replaced by 'SuperClass' or it's sub classes.

- For example

```
class Test<T extends Number> //Declaring Number class as upper bound of T
```

```
{
```

```
    T t;
```

```
    public Test(T t)
```

```
{
```

```
    this.t = t;
```

```
}
```

```
    public T getT()
```

```
{
```

```
    return t;
```

```
}
```

```
}
```

```
public class BoundedTypeDemo
```

```
{
```

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

```
{
```

```
//Creating object by passing Number as a type parameter
```

```

Test<Number> obj1 = new Test<Number>(123);
System.out.println("The integer is: "+obj1.getT());
//While Creating object by passing String as a type parameter, it gives compile time
//error
Test<String> obj2 = new Test<String>"("I am string"); //Compile time error
System.out.println("The string is: "+obj2.getT());
}
}

```

Review Question

1. Explain the use of bounded types in Generics with illustrative program.

4.8 Restrictions and Limitations

1. In Java, generic types are compile time entities. The runtime execution is possible only if it is used along with raw type.
2. Primitive type parameters is not allowed for generic programming.
For example: Stack<int> is not allowed.
3. For the instances of generic class throw and catch instances are not allowed.

For example :

```

public class Test<T> extends Exception
{
    //code // Error:can't extend the Exception class
}

```

4. Instantiation of generic parameter T is not allowed.

For example :

```

new T() //Error
new T[10]

```

5. Arrays of parameterized types are not allowed.

For example :

```

new Stack<String>[10];//Error

```

6. Static fields and static methods with type parameters are not allowed.

Part III : Strings
4.9 Basic String Class

Definition : String is a collection of characters. In Java String defines the object.

The string is normally used to represent the collection of characters

Syntax of String Constructor

<code>String()</code>	It is used for initialization of string object. It represents empty character sequence.
<code>String(byte[] byte)</code>	Constructs a new String by decoding the specified array of bytes using the platform's default charset.
<code>String(byte[] bytes, int offset, int length, String charsetName)</code>	Constructs a new String by decoding the specified subarray of bytes using the specified charset.
<code>String (String strObj)</code>	It constructs a String object which is initialized to same character sequence as that of the string referenced by strObj.

Example Program

Let us look at a sample program in which the string object is used in order to handle some text.

Java Program [StringDemo.Java]

```
/*This is a Java program which makes use of strings
*/
```

```
class StringDemo
{
    public static void main(String args[])
    {
        String s="Hello, How are You?";
        System.out.println(s);
    }
}
```

Output

Hello, How are You?

String Literal

In Java String can be represented as a sequence of characters enclosed within the double quotes.

For example

`String s="Hello,How are you?"`

We can initialize the string using the empty string literal. For example

`String mystr=""`

Similarly the escape sequence can be used to denote a particular character in the string literal. For example we can use back slash followed by double quote to print the double quote character.

"\ "

Alternatively, String can be denoted as the array of characters. For example :

```
char str[] = {'P','R','O','G','R','A','M'};
```

Now we can have a variable **s** which can then be initialized with the string "PROGRAM" as follows -

```
String s = new String(str);
```

Operations on String

Following are some commonly defined methods by a string class -

Method	Description
s1.charAt(position)	Returns the character present at the index position.
s1.compareTo(s2)	If $s1 < s2$ then it returns positive, if $s1 > s2$ then it returns negative and if $s1 = s2$ then it returns zero.
s1.concat(s2)	It returns the concatenated string of s1 and s2.
s1.equals(s2)	If s1 and s2 are both equal then it returns true.
s1.equalsIgnoreCase(s2)	By ignoring case, if s1 and s2 are equal then it returns true.
s1.indexOf('c')	It returns the first occurrence of character 'c' in the string s1.
s1.indexOf('c',n)	It returns the position of 'c' that occur at after nth position in string s1.
s1.length()	It gives the length of string s1.
String.valueOf(var)	Converts the value of the variable passed to it into String type.

4.10 String Methods

AU : Dec.-13, Marks 8

We can find the length of a given string using the method length(). Following program shows the use of length() method for strings

Java Program [Str_lengthDemo.Java]

```
class Str_lengthDemo
{
    public static void main(String[] args)
    {
        char str[] = {'P','R','O','G','R','A','M'};
        String S = new String(str);
```

```

System.out.println("The string S is "+S);
System.out.println("The length of string is "+S.length());
System.out.print("The string in character array is ");
for(int i=0;i<S.length();i++)
    System.out.print(str[i]);
}
}

```

Output

```

D:\>javac Str_lengthDemo.java
D:\>java Str_lengthDemo
The string S is PROGRAM
The length of string is 7
The string in character array is PROGRAM

```

String in reverse direction

There is no direct method for reversing the string but we can display it in reverse direction. Following is the Java program for the same

Java Program[Str_reverse.java]

```

*****
Printing the String in reverse order
*****
class Str_reverse
{
    public static void main(String[] args)
    {
        char str[]={'S','T','R','T','N','G'};
        String S=new String(str);
        System.out.println("The string S is "+S);
        System.out.print("The string written in Reverse order ");
        for(int i=S.length()-1;i>=0;i--)
            System.out.print(str[i]);
    }
}

```

Output

```

The string S is STRING
The string written in Reverse order GNIRTS

```

4.10.1 Character Extraction

- As we know that the string is a collection of characters. **String** class provides the facility to extract the character from the **String** object. There is a method called **charAt(index)** with the help of which we can extract the character denoted by some index in the array.

For example :

```
String fruit=new String("mango");
```

```
char ch;
ch=fruit.charAt(2);
System.out.println(ch);
```

- The output of above code fragment will be n because at the index position 2 of string object fruit the character n is present.

4.10.2 String Comparison

Sometimes we need to know whether two strings are equal or not. We use method **equals()** for that purpose. This method is of Boolean type. That is, if two strings are equal then it returns true otherwise it returns false.

The syntax is -

```
Boolean equals(String Str);
```

Let us see one illustrative program -

Java Program [StringCompareDemo.java]

```
class StringCompareDemo
{
    public static void main(String[] args)
    {
        String str1=new String("INDIA");
        String str2=new String("india");
        if(str1.equals(str2)==true)
            System.out.println("\n The two strings are equal");
        else
            System.out.println("\n The two strings are not equal");
    }
}
```

Output

```
D:\>javac StringCompareDemo.java
```

```
D:\>java StringCompareDemo
The two strings are not equal
```

The above program will generate the message "The two strings are not equal" if str1 and str2 are not equal but if we write same str1 and str2 then naturally the output will be "The two strings are equal". This string comparison is case sensitive. But if we want to compare the two strings without caring for their case differences then we can use the method **equalsIgnoreCase()**.

4.10.3 Searching for the Substring

We can look for the desired substring from the given string using a method **substring()**. The syntax of method **substring()** is as follows -

String substring(int start_index,int end_index)

Java Program [SubstringDemo.java]

```
class SubstringDemo
{
    public static void main(String[] args)
    {
        String str1=new String("I love my country very much");
        System.out.println("\n One substring from the given sentence
                           is:" + str1.substring(2,6));
        System.out.println("\n And another substring
                           is:" + str1.substring(18));
    }
}
```

Output

D:\>javac SubstringDemo.java

D:\>java SubstringDemo

One substring from the given sentence is:love
And another substring is:very much

4.10.4 Replacing the Character from String

We can replace the character by some desired character. For example

Java Program [replaceDemo.java]

```
class replaceDemo
{
    public static void main(String[] args)
    {
        String str=new String("Nisha is Indian ");
        String s=str.replace('i','a');
        System.out.println(s);
    }
}
```

Output

D:\>javac replaceDemo.java

D:\>java replaceDemo

Nasha as Indaan

4.10.5 Upper and Lower Case

We can convert the given string to either upper case or lower case using the methods `toUpperCase()` and `toLowerCase()`. Following program demonstrates the handling of the case of the string -

Java Program [CaseDemo.Java]

```
class caseDemo
{
    public static void main(String[] args)
    {
        String str=new String("Nisha is Indian ");
        System.out.println("\n The original string is: "+str);
        String str_upper= str.toUpperCase();
        System.out.println("\n The Upper case string is: "+str_upper);
        String str_lower= str.toLowerCase();
        System.out.println("\n The Lower case string is: "+str_lower);
    }
}
```

Output

```
D:\>javac caseDemo.java
D:\>java caseDemo
```

The original string is: Nisha is Indian

The Upper case string is: NISHA IS INDIAN

The Lower case string is: nisha is Indian

4.10.6 Concatenating Strings

We can concatenate two strings using + operator which is illustrated by following code -

Java Program [Test.Java]

```
class Test
{
    public static void main(String[] args)
    {
        String fruit=new String("mango");
        System.out.println("I like "+fruit+" very much");
    }
}
```

Here we have used + to concatenate
two strings

Output

I like mango very much

We can make use of a method `concat()` for concatenating two strings. The above program is slightly modified for the use of `concat()` -

Java Program

```

class Test
{
public static void main(String[] args)
{
String str=new String("I like ");
String fruit=new String("mango very much");
System.out.println(str.concat(fruit));
}
}

```

Output

I like mango very much

Ex. 4.10.1 : Write a Java Program that collects the input as a decimal number of integer type and converts it into a String of equivalent hexadecimal number.

Sol. :

AU : (IT) Dec.-13, Marks 8

```

import java.io.*;
class DecToHex
{
    public static void main(String[] args) throws IOException
    {
        //Following 3 statements is required to read the input through keyboard
        BufferedReader br=new BufferedReader (new InputStreamReader(System.in));

        System.out.print("Enter a decimal number : ");
        int num=Integer.parseInt(br.readLine());

        int rem;
        String str="";
        //array storing the digits (as characters) in a hexadecimal number system
        char digits[]={'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'};
        while(num>0)//divide until num is 0

        {
            rem=num%16; //finding remainder by dividing the number by 16
            str=digits[rem]+str; //adding the remainder to the result
            num=num/16;
        }
        System.out.println("Output = "+str);
    }
}

```

Output

Enter a decimal number : 10

Output = A

Ex. 4.10.2 : Write a Java Program that arranges the given set of strings in alphabetical order. Supply the strings through the command line.

AU : IT: Dec.-13, Marks 8

Sol. :

```

class StringDemo
{
    public static void main(String[] args)
    {
        String temp;
        int n = args.length;
        int i,j,c;
        for (i=0; i<n-1; i++)
        {
            for (j=i+1; j<n; j++)
            {
                c = args[i].compareTo(args[j]);
                if (c >0)
                {
                    temp = args[i];
                    args[i] = args[j];
                    args[j] = temp;
                }
            }
        }
        for (i=0; i<n ;i++)
        {
            System.out.println(args[i]);
        }
    }
}

```

Output

```

D:\>javac StringDemo.java
D:\>java StringDemo Swapna Radha Archana Shilpa
Archana
Radha
Shilpa
Swapna

```

4.11 StringBuffer Class

- The **StringBuffer** is a class which is alternative to the **String** class. But **StringBuffer** class is more flexible to use than the **String** class. That means, using **StringBuffer** we can insert some components to the existing string or modify the existing string but in case of **String** class once the string is defined then it remains fixed. The **StringBuffer** and the **StringBuilder** are almost one and the same. The **StringBuilder** or **StringBuffer** have three constructors and 30 methods. Following are some simple methods used for **StringBuffer** -

Name of method	Description
append(String str)	Appends the string to the buffer
capacity()	It returns the capacity of the string buffer
charAt(int index)	It returns a specific character from the sequence which is specified by the index.

delete(int start,int end)	It deletes the characters from the string specified by the starting and ending index.
insert(int offset,char ch)	It inserts the character at the position specified by the offset.
length()	It returns the length of the string buffer.
setCharAt(int index,char ch)	The character specified by the index from the stringbuffer is set to ch.
setLength(int new_len)	It sets the length of the string buffer.
toString()	It converts the string representing data in this string buffer.
replace(int start,int end,String str)	It replaces the characters specified by the new string
reverse()	The character sequence is reversed.

These methods can be illustrated with the help of following Java Programs -

Ex. 4.11.1 : Write a Java Program illustrating the difference between the capacity and length function of StringBuffer.

Sol. :

Java Program [StringBuffDemo1.java]

```
public class StringBuffDemo1{
    public static void main(String args[])
    {
        StringBuffer str=new StringBuffer("Java");
        System.out.println("The String Buffer is "+str);
        System.out.println("The length is "+str.length());
        System.out.println("The capacity is"+ str.capacity());
    }
}
```

Creating string buffer in str

Finding length of str

Finding capacity of str

Output

F:\test>javac StringBuffDemo1.java

F:\test>java StringBuffDemo1

The String Buffer is Java

The length is 4

The capacity is 20

Program Explanation

Using the StringBuffer/StringBuilder class, we can create a buffer of characters. The length function returns the number of characters in the string and the capacity function returns the number of characters in the string +16 additional characters.

Ex. 4.11.2 : Write a Java program, to convert the string 'Great' to a new string 'God'.

Sol. :

```
public class StringBufferDemo2{
    public static void main(String args[])
    {
        StringBuffer str=new StringBuffer("Great");
        System.out.println("The String is: "+str);
        str.setCharAt(1,'o');
        str.setCharAt(2,'d');
        str.setLength(3);
        System.out.println("Now the String is: "+str);
    }
}
```

Output

F:\test>javac StringBufferDemo2.java

F:\test>java StringBufferDemo2

The String is: Great

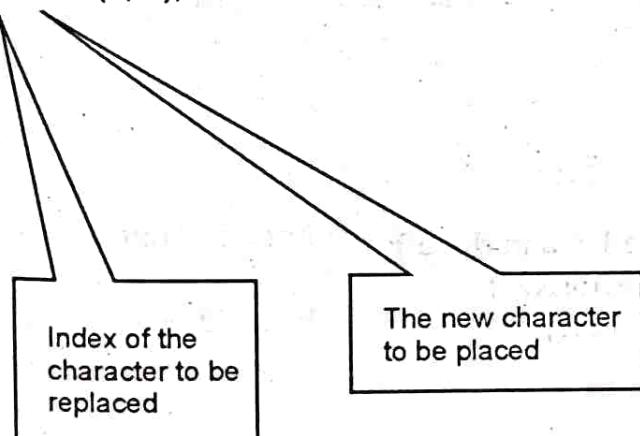
Now the String is: God

Program Explanation

In above program, we have to replace certain characters by new characters and the length of the string needs to be changed. For that purpose, there are two methods defined in **StringBuffer** and those are - **setCharAt** and **setLength**.

The **setCharAt** method, replaces the desired character by a new character.

`setChar(1,'o');`



Similarly, there is a **setLength** function which sets the length of the string. The string 'Great' has a length 5. To convert this string to a new string say 'God', the length is set to 3. The above given output simplifies the program.

Ex. 4.4.3 : Write a simple Java program to append the string "God" by the string "is Great" and then append an exclamation mark to the complete string.

Sol. :**Java Program [StringBuffDemo3.java]**

```
public class StringBufferDemo3{
    public static void main(String args[])
    {
        StringBuffer str=new StringBuffer("God");
        System.out.println("Initially the String is: "+str);
        str.append(" is Great");
        str.append("!");
        System.out.println("Now the String is: "+str);
    }
}
```

Output

F:\test>javac StringBufferDemo3.java

F:\test>java StringBufferDemo3

Initially the String is: God

Now the String is: God is Great!

Program Explanation

The append method is used to insert the string at the end of a given string. In case of String class the + operator is used to append the string to the existing string. Using append method we can append character string, numeric value or any object.

Ex. 4.11.4 : Write a Java program to insert a string "FRIEND" in the given string "----in need is a ---- in deed".

Sol. :**Java Program[StringBuffDemo4.java]**

```
public class StringBufferDemo4{
    public static void main(String args[])
    {
        StringBuffer str=new StringBuffer(" in need is a in deed");
        StringBuffer new_str=new StringBuffer(" FRIEND");
        System.out.println("Initially the String is: "+str);
        str.insert(13,new_str);
        str.insert(0,new_str);

        System.out.println("Now the String is: "+str);
    }
}
```

Output

F:\test>javac StringBufferDemo4.java

F:\test>java StringBufferDemo4

Initially the String is: in need is a in deed

Now the String is: FRIEND in need is a FRIEND in deed

F:\test>

Program Explanation

Using the insert method, the string or numeric value or any character can be inserted at the specified location in the given string. The program is self explanatory.

Ex. 4.11.5 : Write a simple Java program to check the palindrome of the given string.

Sol. :

Java Program [StringBuffDemo5.java]

```
public class StringBufferDemo5{
    public static void main(String args[])
    {
        String str1=new String("Java");
        StringBuffer str2=new StringBuffer(str1);
        str2.reverse();
        System.out.println("Initially the String is: "+str1);
        System.out.println("Reversed String is: "+str2);
        if(str1.equals(str2.toString())) //checks if str1=str2
            System.out.println("This string is palindrome ");
        else
            System.out.println("This string is not palindrome ");
    }
}
```

Output

F:\test>javac StringBufferDemo5.java

F:\test>java StringBufferDemo5

Initially the String is: Java

Reversed String is: avaJ

This string is not palindrome

Ex. 4.11.6 : Write a simple Java program to change the string "impossible" to "possible".

Sol. :**Java Program [StringBuffDemo6.java]**

```
public class StringBufferDemo6{
    public static void main(String args[])
    {
        StringBuffer str1=new StringBuffer("Impossible");
        System.out.print("The word "+str1);
        str1.delete(0,2);
        System.out.println(" is changed to "+str1);
    }
}
```

Output

```
F:\test>javac StringBufferDemo6.java
F:\test>java StringBufferDemo6
The String Impossible is changed to possible
```

Program Explanation

In above program, we have used the method **delete** for deleting the some characters. The **first argument** denotes the starting character and the **second character** denotes the ending character.

Sr. No.	String	StringBuffer
1.	Strings are fixed length constants.	StringBuffers are variable length constants.
2.	Objects of String class are immutable. That means the objects of String class can not be changed.	The objects of StringBuffer class can be changed.

4.12 Two Marks Questions with Answers**Q.1 What is stream ?**

Ans. : Stream is basically a channel on which the data flow from sender to receiver.

Q.2 What is input stream and output stream ?

Ans. : An input object that reads the stream of data from a file is called input stream and the output object that writes the stream of data to a file is called output stream.

Q.3 What is byte stream ? Enlist its super classes.

Ans. : The byte stream is used for inputting or outputting the bytes. The **InputStream** and **OutputStream** are the superclass of byte stream.

Q.4 What is character stream ? Enlist its super classes.

Ans. : The character stream is used for inputting or outputting the characters. The **Reader** and **Writer** are the superclass of character stream.

Q.5 List the byte stream classes.

Ans. : The Byte Stream classes are –

1. FileInputStream
2. PipedInputStream
3. FilterInputStream
4. ByteArrayInputStream
5. FileOutputStream
6. PipedOutputStream
7. FilterOutputStream
8. ByteArrayOutputStream

Q.6 What is the purpose of BufferedInputStream and BufferedOutputStream classes ?

Ans. :

- These are efficient classes used for speedy read and write operations.
- We can specify Buffer size while using these classes.

Q.7 Give an example on stream.

Ans. :

- There are two types of streams - Byte stream and Character stream.
- Stream classes for byte stream : FileInputStream, PipedInputStream, BufferedInputStream, FileOutputStream, PipedOutputStream and so on.
- Stream classes for character stream : FileReader, PipeReader, BufferedReader, FileWriter, PipeWriter, BufferedWriter and so on.

Q.8 What is absolute file name ?

Ans. : The filename that can be specified with the complete path name and drive letter is called absolute file name.

Q.9 What is relative file name ?

Ans. : The file name which is specified as a path relative to the current directory is called relative file name. For example new File("myprogram.html");

Q.10 What is the use of seek method ?

Ans. : The seek() allows you to specify the index from where the read and write operations will start. The syntax is

```
void seek(long position);
```

Q.11 Write a Java code to check if the command line argument is file or not.

Ans. :

```
class Test
{
    public static void main(String [] args)
    {
        File obj=new File(args[0]);
        If(obj.isFile())
    }
}
```

```
        {
            System.out.println("This is a file name"+obj.getPath());
        }
    }
```

O.12 Enlist the two common constructors of FileInputStream.

Ans : The common constructors of FileInputStream are

```
FileInputStream(String filename);  
FileInputStream(File fileobject);
```

Q.13 What is the use of Input Stream Reader and Output Stream Writer ?

Ans. : The InputStreamReader converts byte into character and OutputStreamWriter converts the characters written into byte.

Q.14 Give an example for reading data from files using FileInputStream

AU : May 19

Ans. :

```
import java.io.FileInputStream;
public class test
{
    public static void main(String args[])
    {
        try
        {
            FileInputStream fin=new FileInputStream("D:\\input.txt");
            int i=0;
            while((i=fin.read())!=-1)
            {
                System.out.print((char)i);
            }
            fin.close();
        }catch(Exception e){System.out.println(e);}
    }
}
```

Q.15 Why generic programming is required ?

OR What is the need for generic programming ?

AU : CSE : May-13, Dec.-18

Ans. : Following some reasons for the need of generic programming -

1. It saves the programmers burden of creating separate methods for handling data belonging to different data types.
 2. It allows the code reusability.
 3. Compact code can be created.

Q.16 List out motivation needed in generic programming.**Ans. :**

1. Suppose we want to create a stack and if we create a stack of integers then it will store only the integer elements, if we try to push any string or any double type element then a compile time error will occur. If we want to push any string then we need to create a separate stack, separate class and separate methods for handling the String elements. Same is true for the elements of any other data type. This results in complex code building. To avoid such complexity, a concept of generic programming is introduced.
2. It saves the programmers burden of creating separate methods for handling data belonging to different data types.
3. It allows the code reusability.
4. Compact code can be created.

Q.17 With an example define a generic class. Or Describe generic classes.**AU : CSE : Dec.-12, 13, May-19**

Ans. : A generic class contains one or more variables of generic data type. Following is a simple example which shows how to define the generic class .

```
public class Test<T>
{
    public Test(){val=null;}
    public Test(T val)
    {
        this.val=val;
    }
    public getVal()
    {
        return val;
    }
    public setVal()
    {
        val=newValue;
    }
    private T val; //variable defined as of generic type
}
```

Q.18 Can generic be used with inheritance in several ways ? What are they ?

Ans. : Following are the ways by which generic can be used with inheritance -

- Consider a class and a subclass such as Employee and Trainee . There are two pair classes Pair<Employee> and Pair<Trainee> which do not possess an inheritance relation. That is, Pair<Trainee> is not a subclass of Pair<Employee> even if these two classes are related to each other.

- The parameterised raw type can be converted to a raw type.
- The generic classes can extend to implement other generic classes.

Q.19 List any two advantages of type parameters.

AU : May-11

Ans. :

1. Due to the use of type parameter it saves the programmers burden of creating separate methods for handling data belonging to different data types.
2. Due to type parameter the early error detection at compile time occurs. This avoids crashing of the code(due to type incompatibility) at run time.

Q.20 State any two challenges of generic programming in virtual machine.

AU : IT : May-13

Ans. :

- i) The virtual machine does not work with generic classes or generic methods. Instead it makes uses raw types in which the raw types are replaced with ordinary java types. Each type variable is replaced with its bound or with object, if it is not bounded. This technique is called type erasure.
- ii) In order to handle the type erasure methods the compiler has to generate bridge methods in corresponding class.



UNIT V

5

JavaFX Event Handling, Controls and Components

Syllabus

JAVAFX Events and Controls : Event Basics - Handling Key and Mouse Events. Controls : Checkbox, ToggleButton - RadioButtons - ListView - ComboBox - ChoiceBox - Text Controls - ScrollPane. Layouts - FlowPane - HBox and VBox - BorderPane - StackPane - GridPane. Menus - Basics - Menu - Menu bars - MenuItem.

Contents

- 5.1 Basics of JAVAFX
- 5.2 Event Basics
- 5.3 Handling Key and Mouse Events
- 5.4 Controls
- 5.5 Layouts
- 5.6 Menus
- 5.7 Two Marks Questions with Answers

5.1 Basics of JAVAFX

- JavaFX was originally developed by Chris Oliver, when he was working for a company named See Beyond Technology Corporation, which was later acquired by Sun Microsystems in the year 2005.
- JavaFX is a set of graphics and media packages that enables developers to design, create, test, debug, and deploy rich client applications that operate consistently across different platforms.
- JavaFX makes it easier to create desktop applications and games in Java.
- JavaFX has replaced Swing as the recommended GUI toolkit for Java. Furthermore, JavaFX is more consistent in its design than Swing, and has more features.

Features of JAVAFX

- 1) JavaFX library is written in java. Hence developer find it easy to learn and write the applications in JAVAFX.
- 2) JAVAFX library creates UI controls using which any GUI based application can be developed.
- 3) It provides the classes for 2D and 3D graphics.
- 4) JavaFX contains a set of ready-to-use chart components, so you don't have to code that from scratch every time you need a basic chart.
- 5) JavaFX components can be styled using CSS.
- 6) It provides the support for integrating audio ,videos, and web applications.

We need to import `javafx.application.Application` class in every JavaFX application.

Every JAVAFX program is divided into three main components - Stage, Scene and Node and Scene graph.

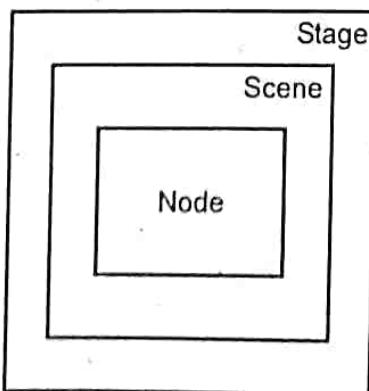


Fig. 5.1.1 Major Components of JavaFX Application

- (1) **Stage** : Stage is like main frame of the program. It acts as a container for all the objects of JavaFX application. The most commonly used stage is a **PrimaryStage**. It is created internally by the platform. The object of PrimaryStage is passed to the **start()** method. There is a show method that is used to **show** the stage.
- (2) **Scene** : The **JavaFx.scene.Scene** class provides the methods to deal with the scene object. It contains the nodes or all the contents of Scene graph. The scene is created by instantiating the Scene class.
- (3) **Scene Graph and Node** : The scene graph is a collection of various nodes. The node is an element that can be visualized on the screen. The node can be button, textbox, radio button and so on.

The nodes are implemented in tree like manner. There is one root node. This will be the parent node of all the other nodes.

Important Methods used in JavaFX Application

- (1) **start()** : This is the method in which we write the code for JavaFX application. The construct for this method is

```
public abstract void start(Stage primaryStage)
```

- (2) **launch()** : In the main method, you have to launch the application using the **launch()** method. This method internally calls the **start()** method of the Application class. Since the **launch()** method is static, you need to call it from a static method such as **main()**

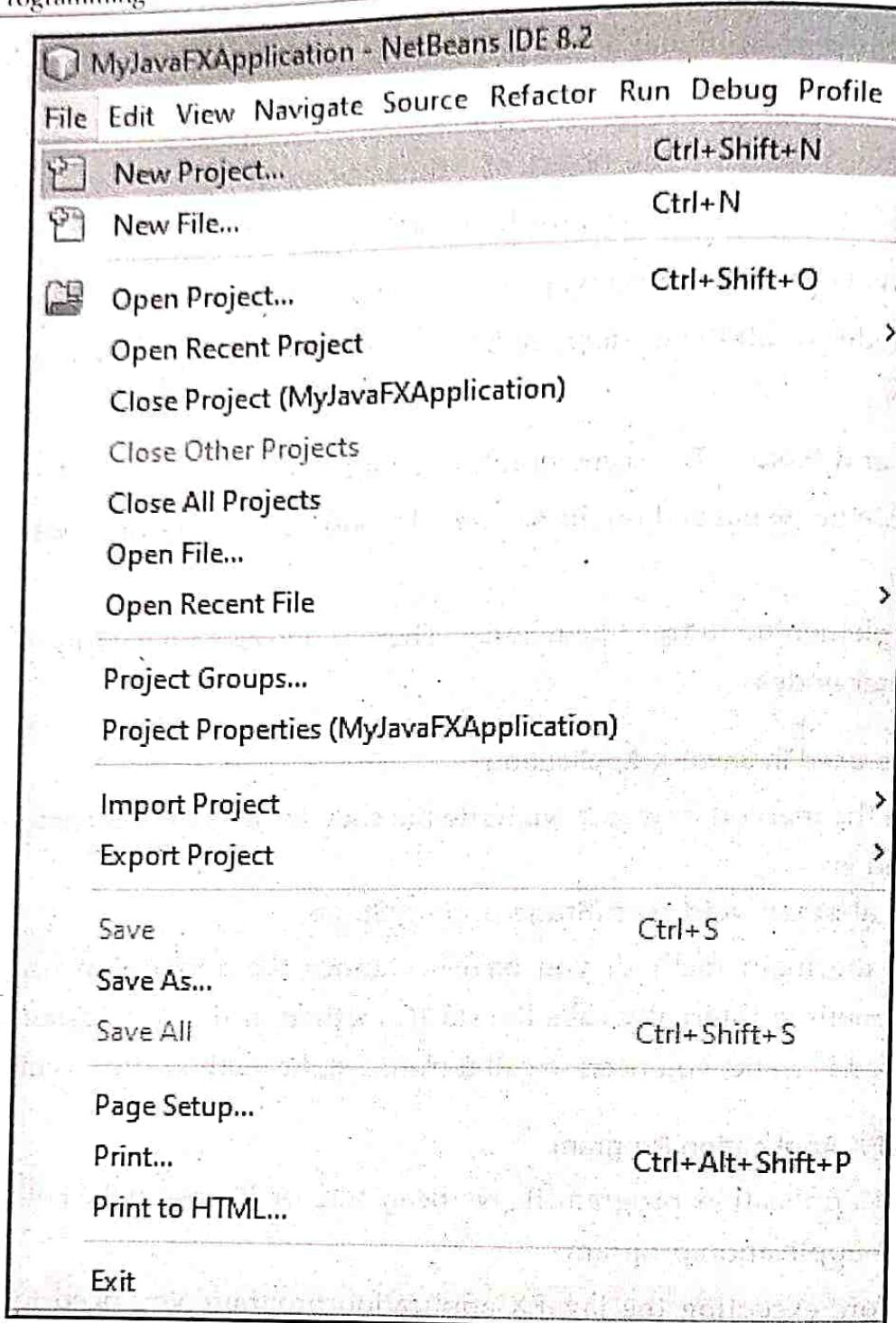
Writing First JavaFX Application Program

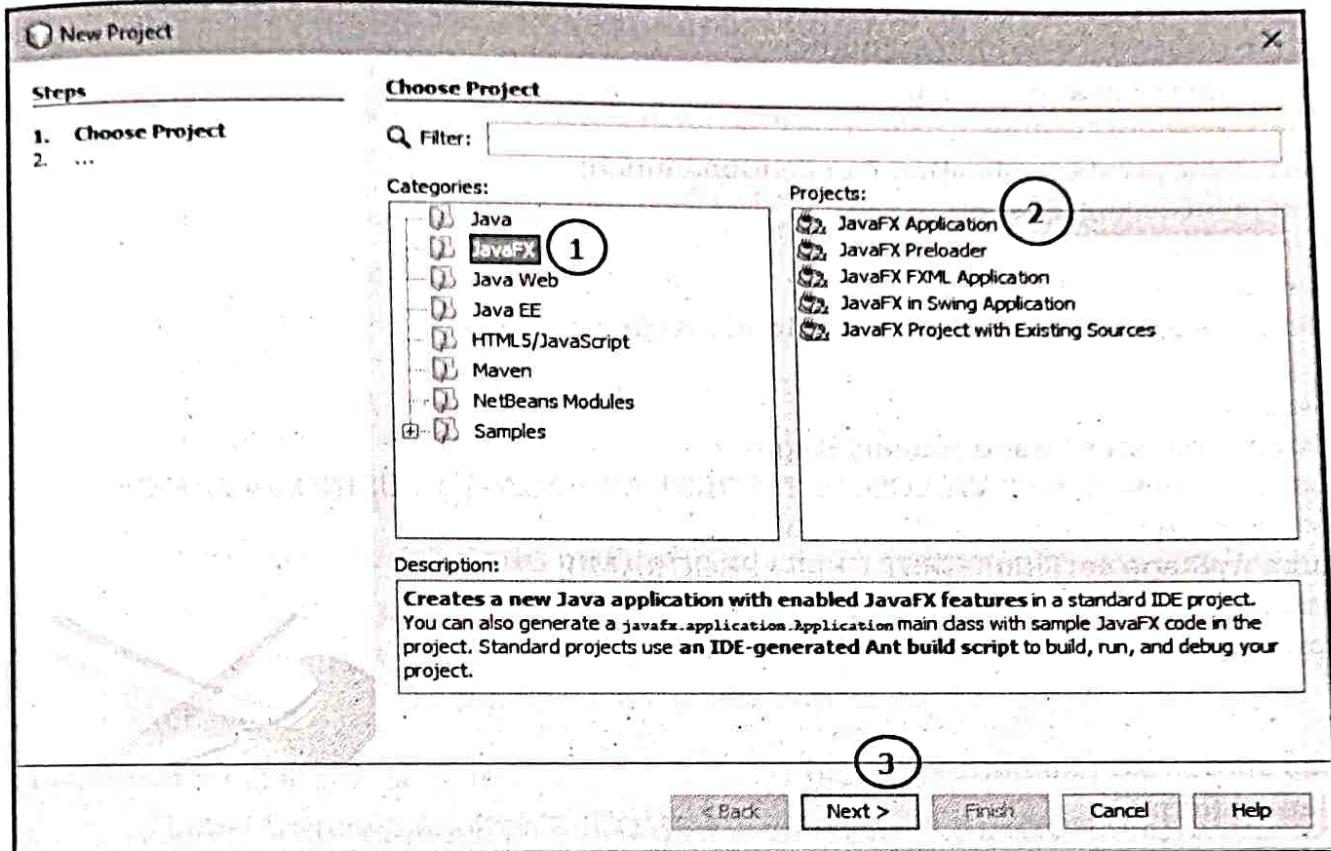
We can create an application program in NetBean IDE or Eclipse IDE. Following steps are used to create this application program -

Prerequisite : Before executing the JavaFX application program, you need to have following things installed in your PC

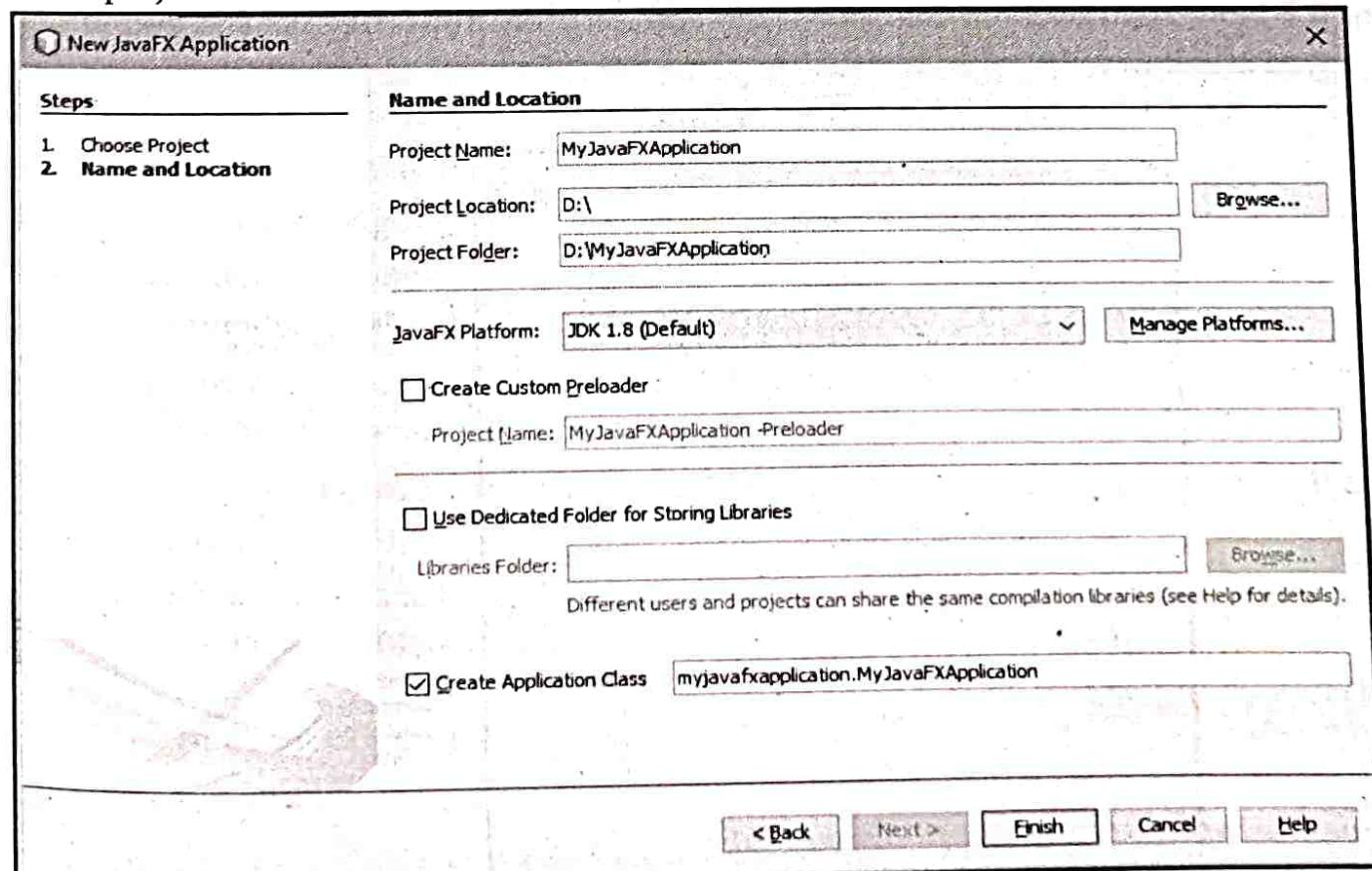
- i) Java 8
- ii) NetBeans 8

Step 1 : From the File menu, choose **New Project**. Then select JavaFX application category, choose JavaFX Application. Click **Next**.





Name the project and click Finish.



Then the code can be written as follows

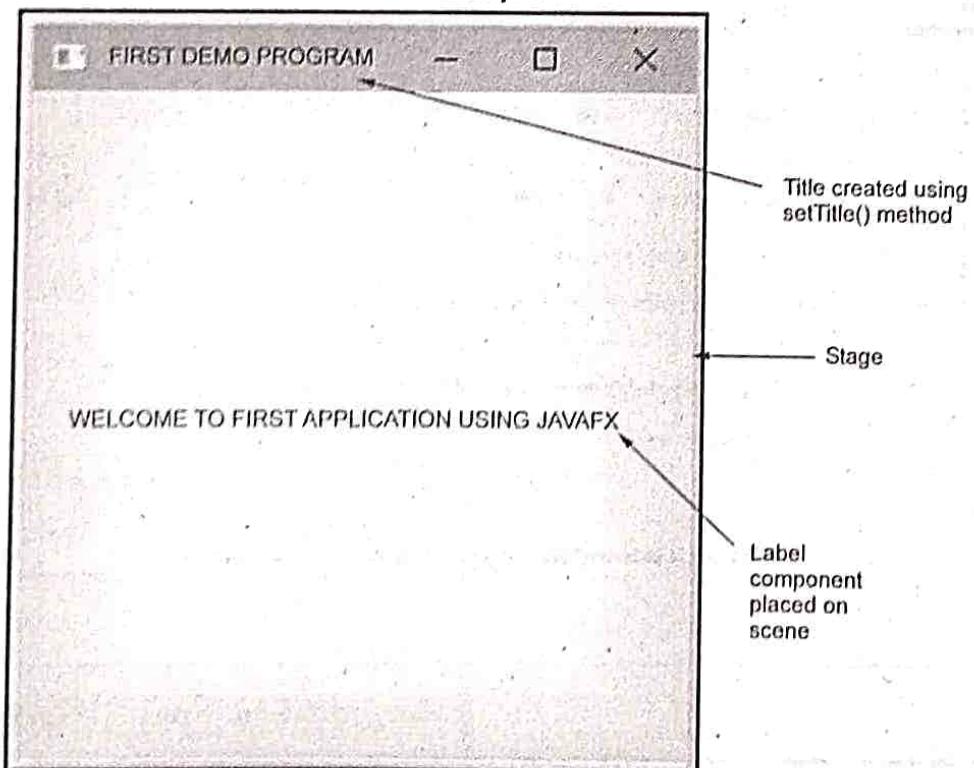
```
package myjavafxapplication;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.stage.Stage;
public class MyJavaFXApplication extends Application {

    @Override
    public void start(Stage primaryStage) {
        Label L = new Label(" WELCOME TO FIRST APPLICATION USING JAVAFX");
        Scene scene = new Scene(L, 300, 300);
        primaryStage.setTitle("FIRST DEMO PROGRAM");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

Step 2 : Now right click on the source file and click on **Run File As**. The output will be displayed as follows

Output



Program Explanation : In above program,

1. We have to import `javafx.application.Application` package to support the `Application` class.

2. Then we need to import the classes for supporting - Stage Scene, Label component and launch method. Hence we add following lines in our code.

```
import static javafx.application.Application.launch;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.stage.Stage;
```

3. Our class must extend `Application` class.

4. Then we write overridden `start()` method in which the entire application code is written. The `primaryStage` is passed as a framework to this start method as a parameter. Note that these are all standard steps for any JavaFX application program.

5. Then we have -

Step 1 : Created a Label component.

```
Label L=new Label(" WELCOME TO FIRST APPLICATION USING JAVAFX");
```

Step 2 : Created a Scene using new operator and placed the Label component on Scene.

```
Scene scene = new Scene(L, 300, 300);
```

Step 3 : Using `setScene` method we have set this scene for Stage

```
primaryStage.setScene(scene);
```

Step 4 : If you want to display the title for your application page, then it is possible by using `setTitle` method for the `Stage` component. Keep it in mind that here `stage` is something like a frame on which scene, UI components are placed.

```
primaryStage.setTitle("FIRST DEMO PROGRAM");
```

Step 5 : Finally any component can be made visible by using `show()` method.

```
primaryStage.show();
```

6. Then write the `main()` method which in turn will call `launch()` method to call your `start()` method.

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

That's it. Now run your application program and a Label component with some message will be displayed as an output.

5.1.1 Panes and UI Controls

- Pane is a container class using which the UI components can be placed at any desired location with any desired size.
- Generally you place a **node** inside a pane and then place pane into a **Scene**. Actually node is any visual component such as UI controls, shapes, or a image view.
- **UI Controls** refer to label, button, Checkbox, radio buttons and so on.
- **Shapes** refer to lines, rectangle, circle and so on.
- A **Scene** can be displayed in a **Stage**. The relationship between node, Pane, Scene, Stage is shown by following figure.

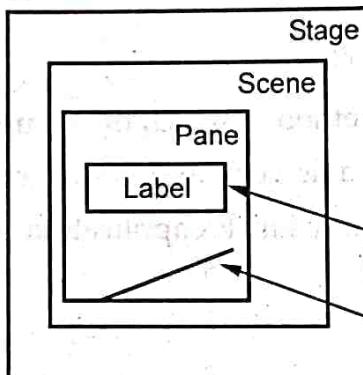
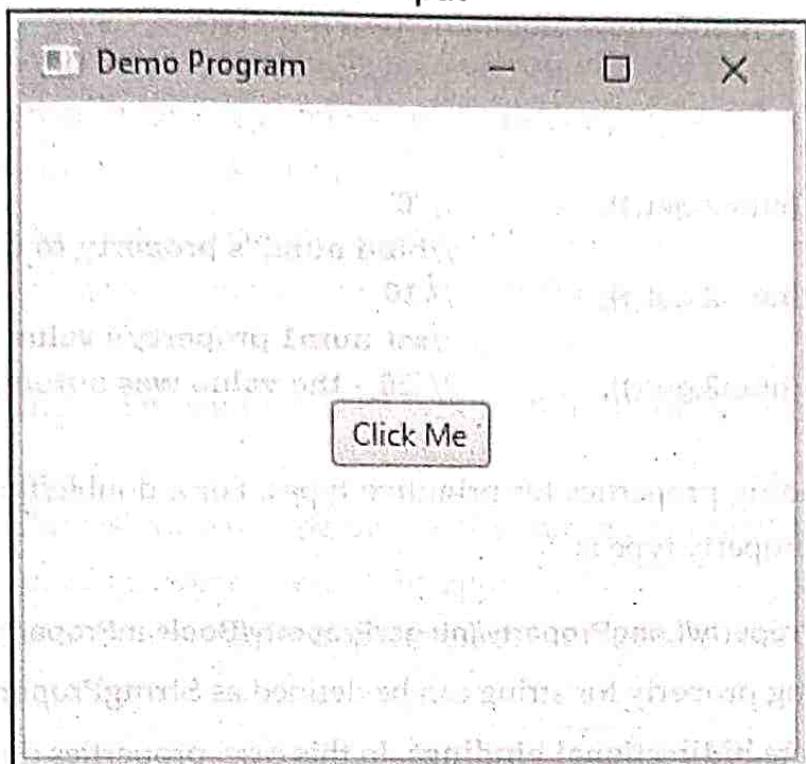


Fig. 5.1.2 Relationship between Node(UI Control and Shape), Pane, Scene, and Stage

• Programming Example

```
package myjavafxapplication;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;
public class MyJavaFXApplication extends Application {
    @Override
    public void start(Stage primaryStage) {
        Button btn = new Button("Click Me");
        StackPane root = new StackPane();
        root.getChildren().add(btn);
        Scene scene = new Scene(root, 300, 250);
        primaryStage.setTitle("Demo Program");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

Output

Program Explanation : In above program,

- 1) We have created a **Pane** named **StackPane**, and added a **button** component as a child of the pane. For that purpose we use **getChildren()** method.
- 2) The **StackPane** places the nodes in the center of the pane on top of each other. Here, there is only one node in the pane. The use of **Pane** allows the button component to be displayed with its preferred size.

5.1.2 Property Binding

- JavaFX introduces a new concept called **binding property** that enables a **target object** to be **bound to a source object**.
- If the value in the source object changes, the target property is also changed **automatically**.
- The target object is simply called a binding object or a binding property.
- A target listens to the changes in the source and automatically updates itself once a change is made in the source. A target binds with a source using the bind method as follows :

```
target.bind(source);
```

- The bind method is defined in the **javafx.beans.property.Property** interface.
- A binding property (target) is an instance of **javafx.beans.property.Property**.
- A source object is an instance of the **javafx.beans.value.ObservableValue** interface.
- For example -

```
//create a property with value=10
SimpleIntegerProperty num1 =new SimpleIntegerProperty(10);
```

```

//create another property
SimpleIntegerProperty num2=new SimpleIntegerProperty();

public void test()
{
    System.out.println(num2.get());           // '0'
    num2.bind(num1);                         //bind num2's property to num1
    System.out.println(num2.get());           // 10
    num1.set(20);                           //set num1 property's value
    System.out.println(num2.get());           // '20' - the value was automatically updated
}

```

- JavaFX defines binding properties for primitive types. For a double/float/ long/int/Boolean value, its binding property type is

DoubleProperty/FloatingPointProperty/LongProperty/IntegerProperty/BooleanProperty

- Similarly the binding property for string can be defined as **StringProperty**.
- It is possible to create **bidirectional bindings**. In this case, properties depend on each other.
- A unidirectional binding binds a target with a source. A bidirectional binding binds two objects together. Changes in one object affects the other. The above given example is an example of unidirectional binding.
- Example of Bidirectional Binding

```

import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;
public class test {
    public static void main(String[] args) {
        DoubleProperty num1 = new SimpleDoubleProperty(10);
        DoubleProperty num2 = new SimpleDoubleProperty(20);
        num1.bind(num2);
        System.out.println("Num1 = " + num1.getValue()); //Outputs 20
        System.out.println("Num2 = " + num2.getValue()); //Outputs 20
        num2.setValue(555);
        System.out.println("Num1 = " + num1.getValue()); //Outputs 555
        System.out.println("Num2 = " + num2.getValue()); //Outputs 555
    }
}

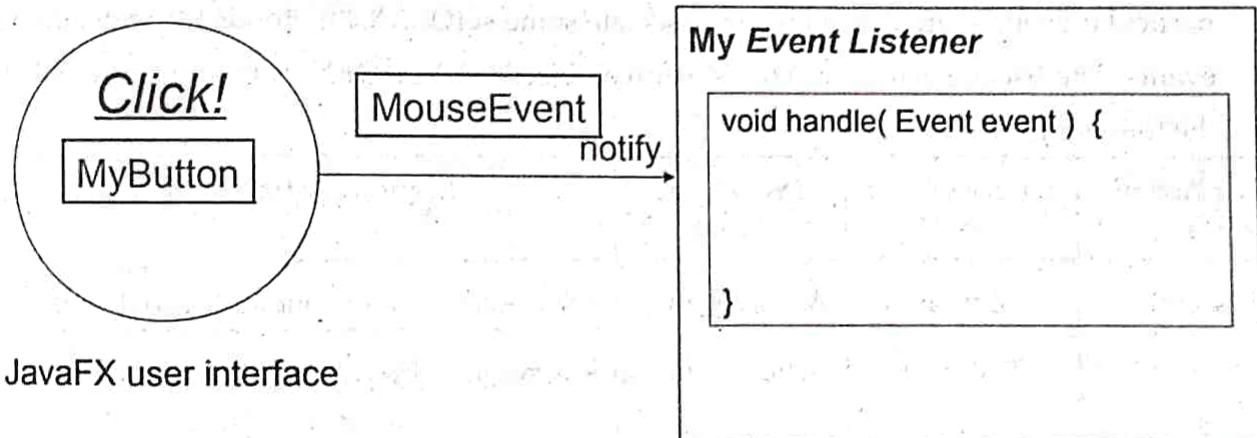
```

Review Questions

- What is property binding concept ?
- Explain the basic structure of JavaFX program with suitable example.
- What are the important methods used in JavaFX application ?
- Explain the concept of Panes, UI control and shapes.

5.2 Event Basics

- Event means any activity that interrupts the current ongoing activity. For example: when user clicks button then it generates an event. To respond to button click we need to write the code to process the button clicking action.
- **Event Source Object :** The object that generates the event is called event source object. For example- if the event gets generated on clicking the button, then button is the event source object.
- **Event Handler :** The event handling code written to process the generated event is called event handler.
- **Event Listener :** The task of handling an event is carried out by event listener. When an event occurs, first of all an event object of the appropriate type is created. This object is then passed to a Listener. A listener must implement the interface that has the method for event handling.



5.2.1 Registering Handlers and Handling Events

- JavaFX has just one interface for all kinds of Event Handlers.
- It is an instance of the `EventHandler<T extends Event>` interface. This interface defines the common behavior for all handlers. `<T extends Event>` denotes that T is a generic type that is a subtype of Event.
- The `EventHandler` object handler must be registered with the event source object using the method `source.setOnAction(handler)`.
- The `EventHandler<ActionEvent>` interface contains the `handle(ActionEvent)` method for processing the action event. We have to write the handler class that must override this method to respond to the event. Hence we must import `javafx.event.EventHandler` class in our application program.

The event Class hierarchy is as shown below -

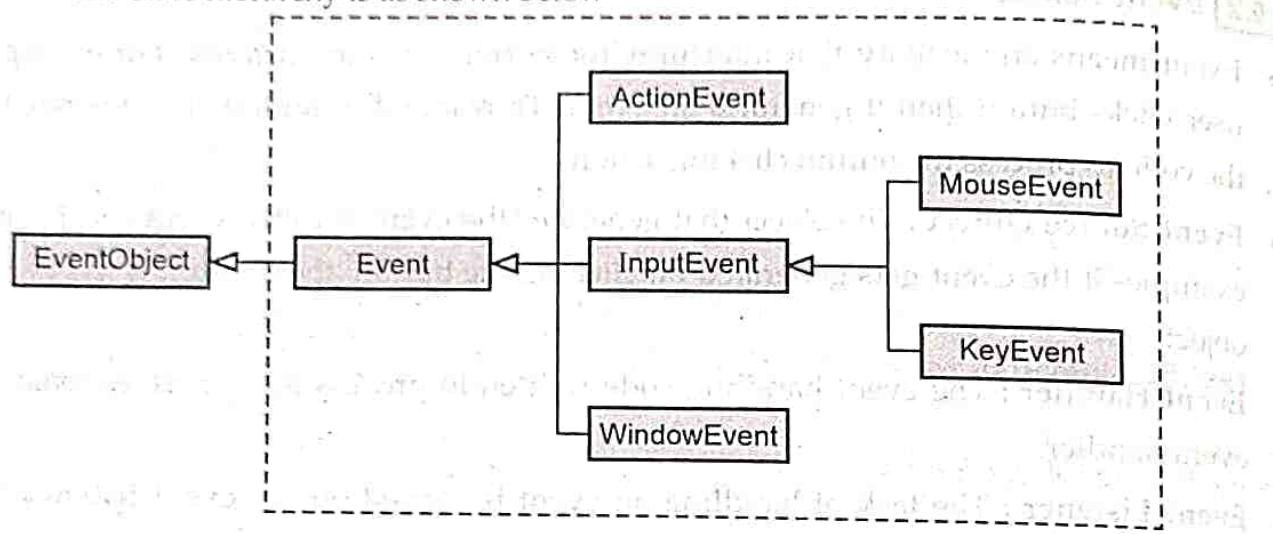
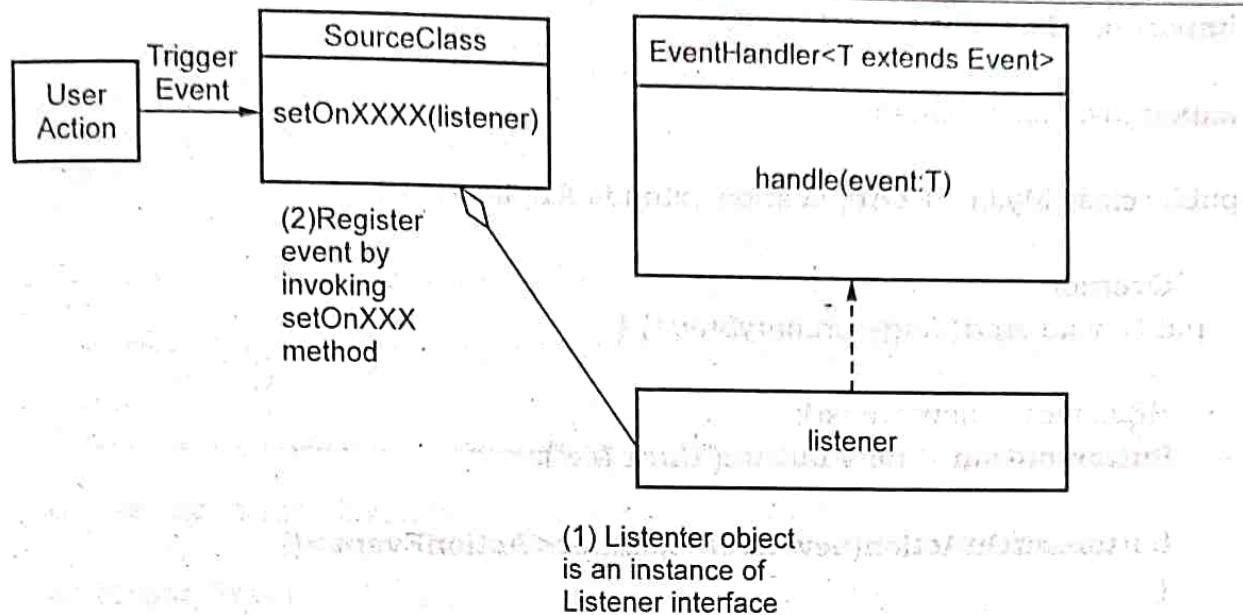


Fig. 5.2.1 Event class hierarchy

- There are user actions that are associated with some **source objects**. For these actions particular event is fired. The JavaFX associate some **setOnXXX** methods for registering these events. The list of such user actions, source objects and **setOnXXX** methods are enlisted in the following table.

User Action	Source Object	Event Type Fired	Event Registration Method
Click button	Button	ActionEvent	setOnAction(EventHandler<ActionEvent>)
Enter text in textfield	TextField	ActionEvent	setOnAction(EventHandler<ActionEvent>)
Click radio button for check or uncheck	RadioButton	ActionEvent	setOnAction(EventHandler<ActionEvent>)
Check or uncheck checkbox	Checkbox	ActionEvent	setOnAction(EventHandler<ActionEvent>)
Select an item	ComboBox	ActionEvent	setOnAction(EventHandler<ActionEvent>)

- Java makes use of delegation based model for event handling and registration. This model is as shown below.



The event handling process is a two step process. It performs following functionalities -

- (1) The handler object is an instance of appropriate **EventHandler** interface. The **EventHandler** interface is defined as **EventHandler<T extends Event>**. It contains **handle()** function for processing the event.
- (2) The handler object is registered by the source object. The **registration methods** depend on the event type. For **ActionEvent**, the method is **setOnAction**. For a mouse pressed event, the method is **setOnMousePressed**. For a key pressed event, the method is **setOnKeyPressed**.

For example

```

Button button = new Button("Click Me");
//Event object is button
button.setOnAction(new EventHandler<ActionEvent>() //Event registration
{
    public void handle(ActionEvent event)
    ...
    ...
})
  
```

Following is a complete JavaFX program, that displays the message on clicking the button. Thus button click event is handled in the following program.

Programming Example

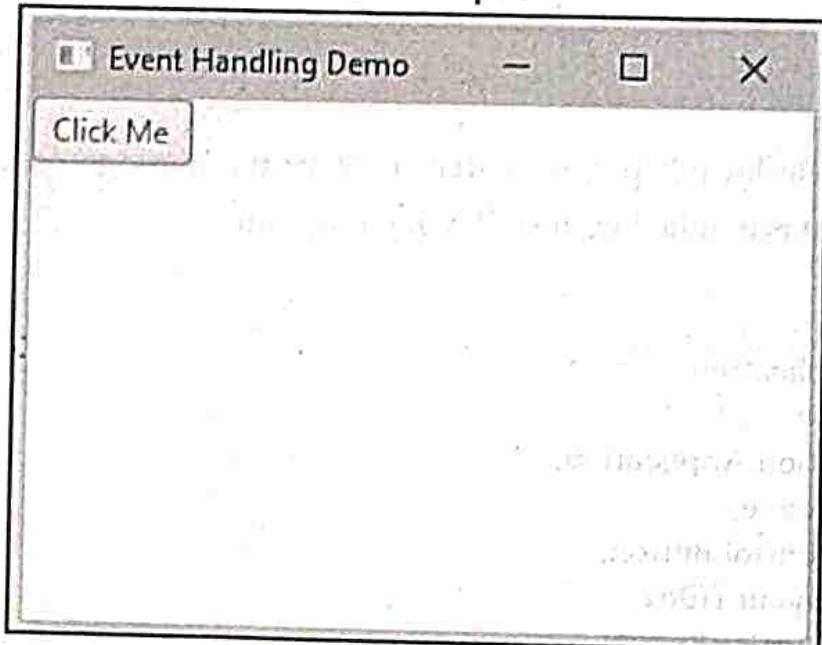
```

package myjavafxapplication;

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.HBox;
import javafx.event.ActionEvent;
  
```

```
import javafx.event.EventHandler;
import javafx.stage.Stage;
public class MyJavaFXApplication extends Application {
    @Override
    public void start(Stage primaryStage) {
        HBox root = new HBox();
        Button button = new Button("Click Me");

        button.setOnAction(new EventHandler<ActionEvent>() {
            public void handle(ActionEvent event) {
                System.out.println("Button is clicked!!");
            }
        });
        root.getChildren().add(button);
        Scene scene = new Scene(root, 300, 200);
        primaryStage.setScene(scene);
        primaryStage.setTitle("Event Handling Demo");
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

Output

On clicking the above button, we get following message in the output window,

```
No base JDK. Package will use system JRE.
jfx-deployment-script:
jfx-deployment:
jar:
Copying 12 files to D:\MyJavaFXApplication\dist\runtimes\run1721531213
jfx-project-run:
Executing D:\MyJavaFXApplication\dist\runtimes\run1721531213\MyJavaFXApplication.jar using platform
Button is clicked!!
```

5.3 Handling Key and Mouse Events

5.3.1 Handling Mouse Event

Mouse event is fired when the mouse button is pressed, released, clicked, moved or dragged on the node.

Following table shows the user actions and associated event handling activities -

User Action	Source Object	Event Type Fired	Event Registration Method
Mouse Pressed			setOnMousePressed(EventHandler<MouseEvent>)
Mouse Released			setOnMouseReleased(EventHandler<MouseEvent>)
Mouse Clicked			setOnMouseClicked(EventHandler<MouseEvent>)
Mouse Moved			setOnMouseMoved(EventHandler<MouseEvent>)
Mouse Dragged			setOnMouseDragged(EventHandler<MouseEvent>)
Mouse entered			setOnMouseEntered(EventHandler<MouseEvent>)
Mouse exited			setOnMouseExited(EventHandler<MouseEvent>)

Various methods for **MouseEvent** class are enlisted in the following table

Method	Description
MouseButton getButton()	It represents which mouse button is pressed.
int getClickCount()	It returns the number of mouse clicks.
double getX()	It returns the x-coordinate of the mouse point in the event source node.
double getY()	It returns the y-coordinate of the mouse point in the event source node.
double getSceneX()	It returns the x-coordinate of the mouse point in the scene.
double getSceneY()	It returns the y-coordinate of the mouse point in the scene.
double getScreenX()	It returns the x-coordinate of the mouse point in the screen.
double getScreenY()	It returns the y-coordinate of the mouse point in the screen.
boolean isAltDown()	It checks if the ALT key is pressed or not.
boolean isControlDown()	It checks if the Control key is pressed or not.
boolean isShiftDown()	It checks if the Shidt key is pressed or not.

There are Four constants—PRIMARY, SECONDARY, MIDDLE, and NONE - are defined in **MouseButton** to indicate the left, right, middle, and none mouse buttons.

Programming Example

```
package myjavafxapplication;
```

```
import javafx.application.Application;
import javafx.event.EventHandler;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.input.MouseEvent;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.scene.text.Text;

import javafx.stage.Stage;

public class MyJavaFXApplication extends Application {

    @Override
    public void start(Stage primaryStage) {
```

```

Rectangle rect = new Rectangle(50,50,100,100);
rect.setArcWidth(20);
rect.setArcHeight(20);
rect.setFill(Color.BLUE);
Text text = new Text(20,20,"Click on the rectangle to change its color");

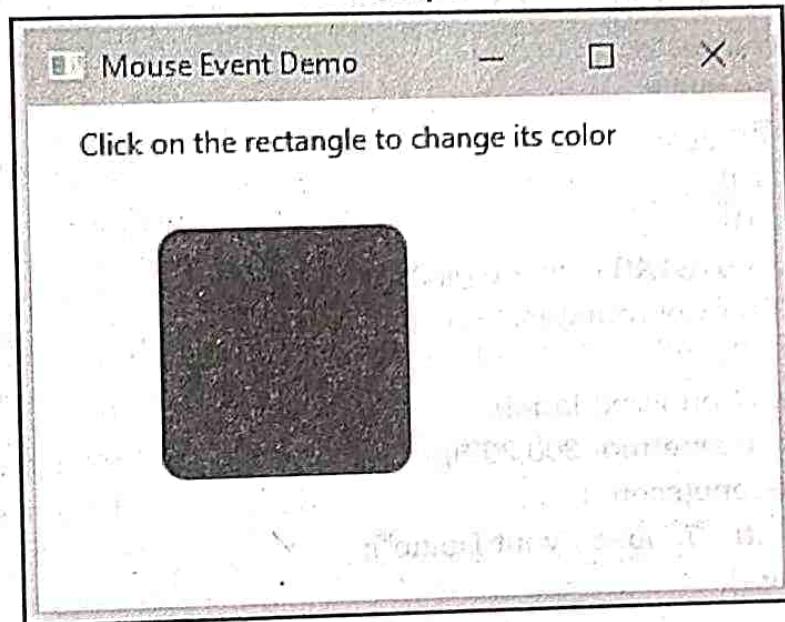
//Handling the mouse event
rect.setOnMouseClicked(e->{
    rect.setFill(Color.RED);
});

Group root = new Group(rect,text);
Scene scene=new Scene(root,300,200);
primaryStage.setScene(scene);
primaryStage.setTitle("Mouse Event Demo");
primaryStage.show();

}

public static void main(String[] args) {
    launch(args);
}
}

```

Output

Program Explanation : In above program,

- (1) We have created one rectangular object and it is colored using blue color.
- (2) The mouse click event is used. When user clicks over the rectangle, the color of the rectangle gets changed to red.
- (3) Mouse Event is created and handled using following code

```

rect.setOnMouseClicked(e->{
    rect.setFill(Color.RED);
});

```

Example 5.3.1 : Write a JavaFX program to display the mouse position in the form of (x,y) co-ordinates when the text present on the screen is dragged somewhere else.

Sol. :

```
package myjavafxapplication;

import javafx.application.Application;
import javafx.event.EventHandler;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.input.MouseEvent;
import javafx.scene.text.Text;

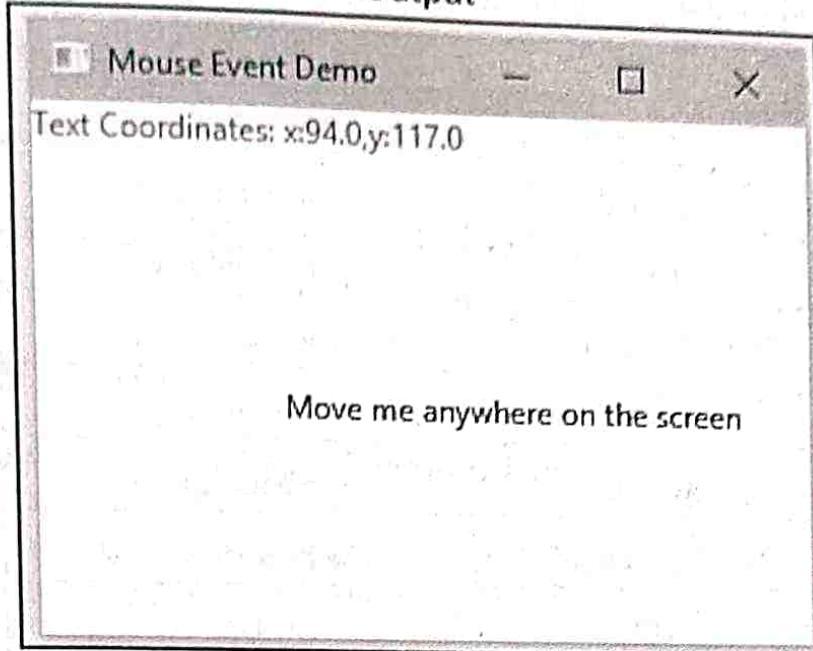
import javafx.stage.Stage;

public class MyJavaFXApplication extends Application {

    @Override
    public void start(Stage primaryStage) {

        Text text = new Text(20,20,"Move me anywhere on the screen");
        Label label = new Label();
        //Handling the mouse event
        text.setOnMouseDragged(e->{
            text.setX(e.getX());
            text.setY(e.getY());
            String msg="x:"+e.getX()+"y:"+e.getY();
            label.setText("Text Coordinates: "+msg);
        });
        Group root = new Group(text,label);
        Scene scene=new Scene(root,300,200);
        primaryStage.setScene(scene);
        primaryStage.setTitle("Mouse Event Demo");
        primaryStage.show();
    }

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

Output

5.3.2 Keyboard Events

When any key on the keyboard is pressed, released, or typed on a node then the keyboard event occurs.

The associated keyboard event triggering actions made by user, type of event and event registration methods are enlisted in the following table.

User Action	Source Object	Event Type Fired	Event Registration Method
Key Pressed	Node, Scene	KeyEvent	setOnKeyPressed(EventHandler<KeyEvent>)
Key Released			setOnKeyReleased(EventHandler<KeyEvent>)
Key typed			setOnKeyTyped(EventHandler<KeyEvent>)

For recognizing the keyboard events we use **KeyEvent** class. Various methods of **KeyEvent** class are as described in the following table.

Method	Description
String getCharacter()	Returns the character associated with the key pressed.
KeyCode getCode()	Returns the key code associated with the key in the event.
String getText()	Returns the string describing the key code.
boolean isAltDown()	It returns true if the ALT key is down.
boolean isControlDown()	It returns true if the Control key is down.
boolean isMetaDown()	It returns true if the Meta button is pressed.
boolean isShiftDown()	It returns true if the shift key is pressed.

The key codes returned by the `getCode()` method are constants. These constants are enlisted in the following table -

Constant	Associated Key
HOME	The Home key
END	The End key
CONTROL	The control key
SHIFT	The shift key
PAGE_UP	The page up key
PAGE_DOWN	The page down key
UP	The Up key
DOWN	The Down key
LEFT	The left arrow key
RIGHT	The right arrow key
BACK_SPACE	The backspace tree
CAPS	The caps lock key
ESCAPE	The escape key
ENTER	The enter key
TAB	The tab key
NUM_LOCK	The num lock key
0 to 9	The keys from 0 to 9
A to Z	The letter keys from A to Z

Following is a simple keyboard event handling program that displays the character typed on the output window.

Programming Example

```
package myjavafxapplication;
```

```
import javafx.application.Application;
import javafx.event.EventHandler;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.input.MouseEvent;
```

```
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;
import javafx.stage.Stage;

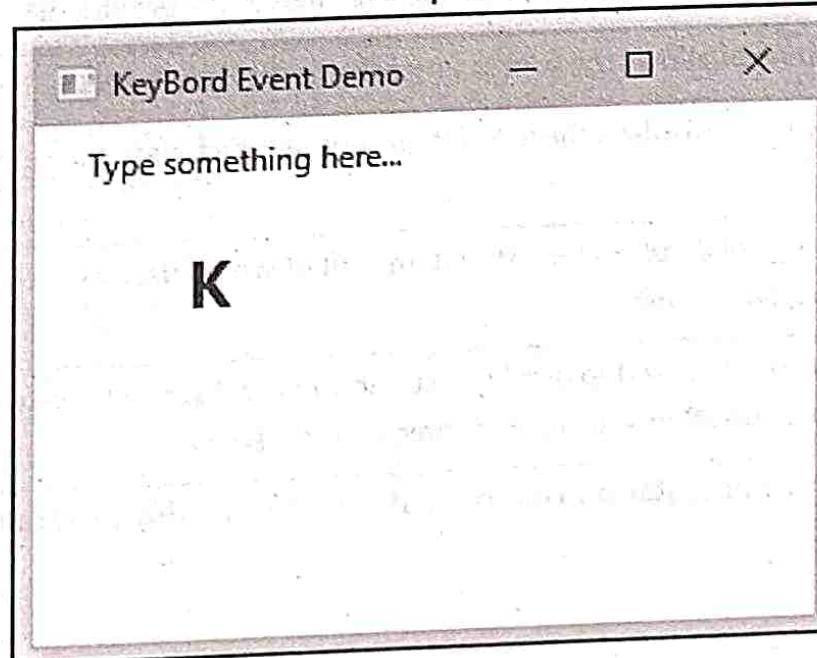
public class MyJavaFXApplication extends Application {

    @Override
    public void start(Stage primaryStage) {

        Text msg = new Text(20,20,"Type something here...");
        Text text = new Text(50,50,"");
        text.setFont(Font.font("Arial",FontWeight.BOLD,30));
        //Creating the keyboard event handler
        text.setOnKeyPressed(e-> {
            text.setText(e.getText());
        });
        Group root = new Group(msg,text);
        Scene scene=new Scene(root,300,200);
        primaryStage.setScene(scene);
        primaryStage.setTitle("KeyBord Event Demo");
        primaryStage.show();
        text.requestFocus();
    }

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

Output



Program Explanation : In above code,

- (1) For displaying the character typed we have set the **Text** node. As we want bold,arial and a font size of 30 px, we use **setFont** method.
- (2) Then the keybpard event is set using the lambda function

```
text.setOnKeyPressed(e->{
    text.setText(e.getText());
});
```

Due to this event, the key being pressed is displayed.

5.4 Controls

- The graphical User Interface (GUI) is an essential part of any window based application.
- The interaction between user and the application program is possible with the help of User Interface (UI) controls that are placed on the screen.
- JavaFX allows its developer a rich set of UI controls. Some of the most commonly used UI Controls are enlisted in the following table.

Control	Description
Label	Label is a control that displays simple text on the screen. It is generally used to describe the other user controls.
Button	Button component is used to control specific function of the application.
RadioButton	The radio button is used to provide the choices to the user. The radio button can be selected or deselected.
CheckBox	The checkbox is used to provide various choices to the user. The check box can be checked(True) or unchecked(False).
TextField	For getting the input from the user, the textfield is provided.
Textarea	This control allows the user to enter the multiple line text.
ComboBox	This control displays the list of items out of which user can select at the most one item.
ListView	This control displays the list of items out of which user can select one or multiple items from the list.
Slider	This control is used to display a continuous or discrete range of valid numeric choices and allows the user to interact with the control.

Let us now learn and understand how to create simple JavaFX programs that use these UI controls.

5.4.1 Label

- The label control displays simple text on the screen.
- Its main purpose is to describe other components such as textfield, textarea, radio button and so on.
- For using this control the package `javafx.scene.control.Label` need to be imported.
- The **constructors** used for using Label control is

`Label()` : This creates an empty label.

`Label(String txt)` : This creates a label with supplied text.

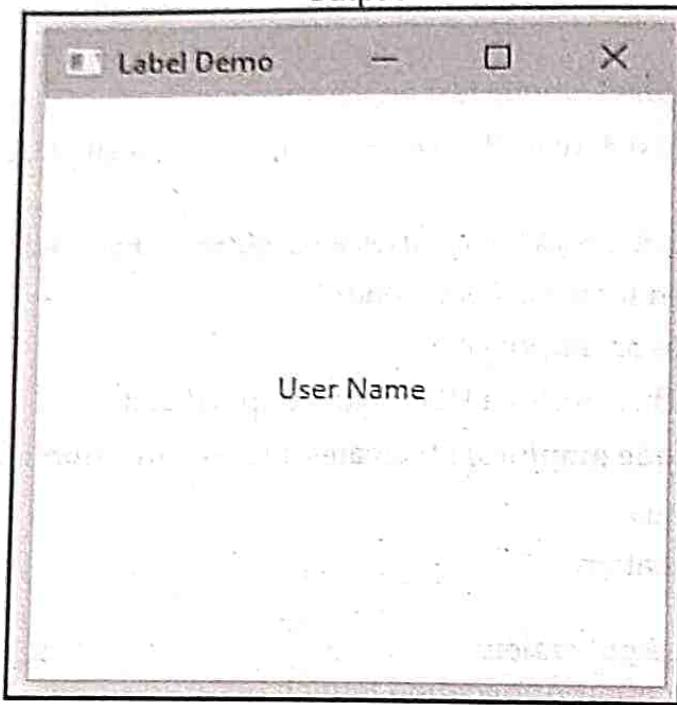
`Label(String txt, Node graphics)` : It creates a label with supplied text and graphics.

- Programming Example

```
package MyJavaFXApplication;
```

```
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.layout.StackPane;
import javafx.scene.control.Label;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class MyJavaFXApplication extends Application {
    @Override
    public void start(Stage primaryStage) {
        StackPane root = new StackPane();
        Label L = new Label("User Name");
        Scene scene = new Scene(root, 250, 250);
        root.getChildren().add(L);
        primaryStage.setTitle("Label Demo");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

Output

Program Explanation : In above program,

- (1) We have created StackPane so that the Label control can be arranged on top just like a stack. The stackPane is used to decide the layout of the screen. This is named as root node.
- (2) Then we have created one label "User Name" using the Label constructor.
- (3) These controls must be contained in a scene. Hence a scene is prepared by adding a root node using Scene constructor. This scene is then attached to a primaryStage using setScene method.
- (4) Then using show() method we display the primaryStage. This ultimately displays the label control.

5.4.2 Button

- The button control is the most commonly used control in any GUI application. This UI component controls the behavior of the application. Some event gets generated when a button is clicked.
- The button control is created using following code

```
Button btn=new Button("Click Here");
```

- For using the button control we need to import the package `javafx.scene.control.Button`. Following program shows the creation of button control in JavaFX
- **Programming Example**

```
package MyJavaFXApplication;
```

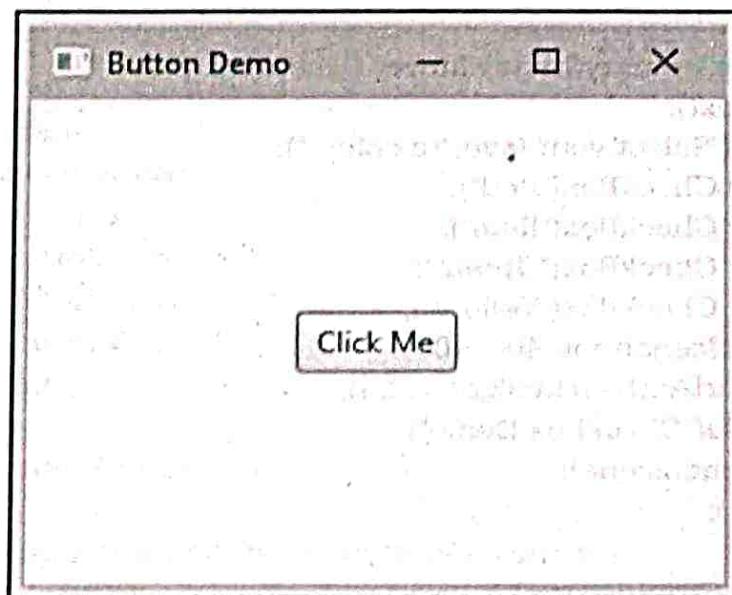
```

import java.io.FileInputStream;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.layout.StackPane;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class MyJavaFXApplication extends Application {
    @Override
    public void start(Stage primaryStage) throws Exception {
        StackPane root = new StackPane();
        Button B = new Button("Click Me");
        Scene scene = new Scene(root, 200, 250);
        root.getChildren().add(B);
        primaryStage.setTitle("Button Demo");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}

```

Output



5.4.3 Checkbox

Checkbox is used to provide **more than one choices** at a time.

For using checkbox in our application program, we must insert following line in the program at the beginning -

```
import javafx.scene.control.Checkbox
```

The checkbox can be created using following statement

```
CheckBox ch=new CheckBox("Label Name");
```

The checkbox is selected **true** by using the method, **setSelected("true")**

Following program shows how to create checkbox.

Programming Example

```
package MyJavaFXApplication;
```

```
import java.io.InputStream;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.CheckBox;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;
```

```
public class MyJavaFXApplication extends Application {
```

```
    @Override
```

```
    public void start(Stage primaryStage) throws Exception {
```

```
        HBox root=new HBox();
```

```
        Label L=new Label("Select your favorite color: ");
```

```
        CheckBox ch1=new CheckBox("Red");
```

```
        CheckBox ch2=new CheckBox("Blue");
```

```
        CheckBox ch3=new CheckBox("Green");
```

```
        CheckBox ch4=new CheckBox("Yellow");
```

```
        Scene scene = new Scene(root, 400, 100);
```

```
        root.getChildren().addAll(L,ch1,ch2,ch3,ch4);
```

```
        primaryStage.setTitle("Checkbox Demo");
```

```
        primaryStage.setScene(scene);
```

```
        primaryStage.show();
```

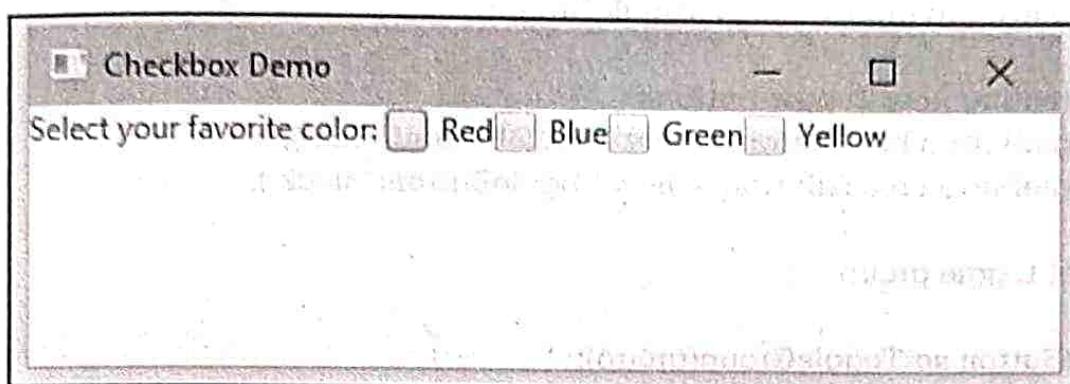
```
}
```

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

```
        launch(args);
```

```
}
```

```
}
```

Output**5.4.4 ToggleButton**

- ToggleButton is a button control that has two states selected or deselected.
- The ToggleButtons are placed in a Toggle Group. From the same group at the most only one button is selected at a time. If one button is selected from that group then other buttons of that group remain deselected.
- Constructors
 1. ToggleButton(): Creates a toggle button with an empty string for its label.
 2. ToggleButton(String txt): Creates a toggle button with the specified text as its label.
- Methods
 1. setToggleGroup(ToggleGroup val): Sets the value of the property toggleGroup.
 2. setSelected(boolean val): Sets the value of the property selected.
 3. isSelected(): Gets the value of the property selected.

Example Program

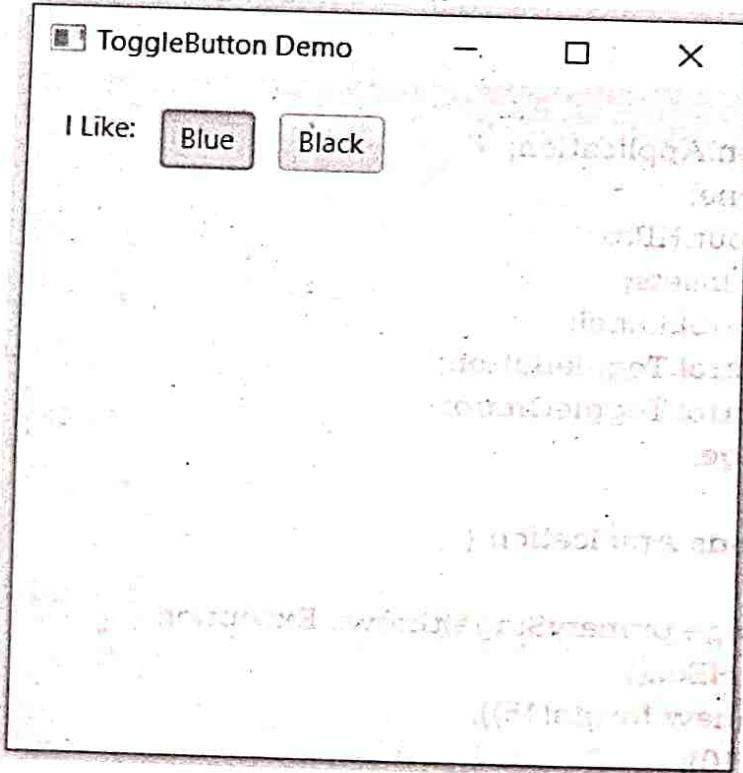
```
package application;

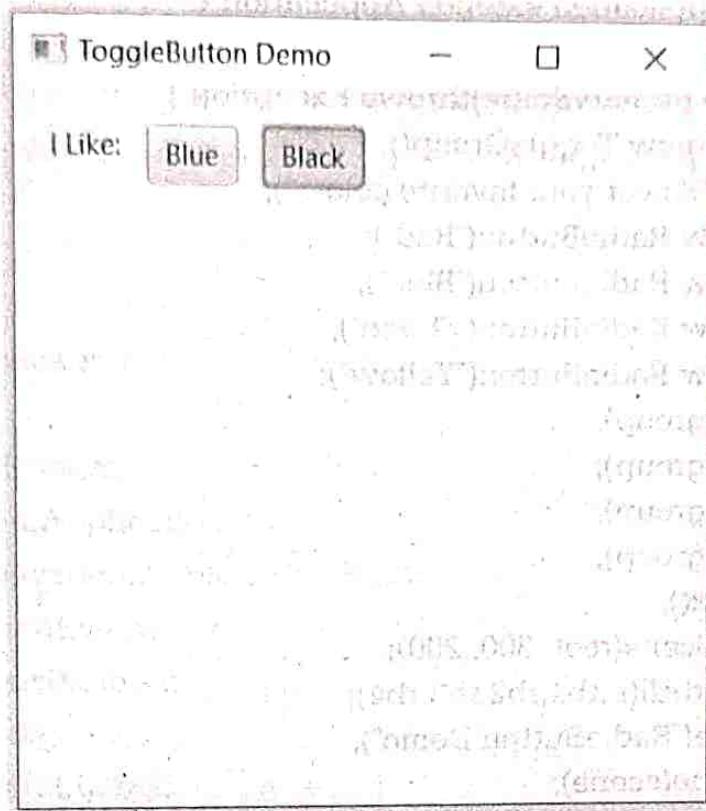
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.HBox;
import javafx.geometry.Insets;
import javafx.scene.control.Label;
import javafx.scene.control.ToggleButton;
import javafx.scene.control.ToggleGroup;
import javafx.stage.Stage;

public class Main extends Application {
    @Override
    public void start(Stage primaryStage) throws Exception {
        HBox root = new HBox();
        root.setPadding(new Insets(15));
        root.setSpacing(10);
        root.getChildren().add(new Label("I Like:"));
    }
}
```

```
// Creating a ToggleGroup  
ToggleGroup group = new ToggleGroup();  
  
// Creating new Toggle buttons  
ToggleButton blueButton = new ToggleButton("Blue");  
ToggleButton blackButton = new ToggleButton("Black");  
  
// Set toggle group  
  
blueButton.setToggleGroup(group);  
blackButton.setToggleGroup(group);  
  
blueButton.setSelected(true);  
  
root.getChildren().addAll(blueButton, blackButton);  
  
Scene scene = new Scene(root, 300, 300);  
primaryStage.setTitle("ToggleButton Demo");  
primaryStage.setScene(scene);  
primaryStage.show();  
}  
public static void main(String[] args) {  
    launch(args);  
}
```

Output





5.4.5 RadioButtons

The Radio button control is used to make a choice.

The difference between checkbox and radio button is that with checkbox we can have more than one selection but with radio button we select only one option at a time.

We can create a radiobutton as follows –

```
RadioButton radioButton1 = new RadioButton("Label1");
```

We can group JavaFX RadioButton instances into a **ToggleGroup**. A **ToggleGroup** allows at most one RadioButton to be selected at any time.

Following example shows how to create and use the radio button control.

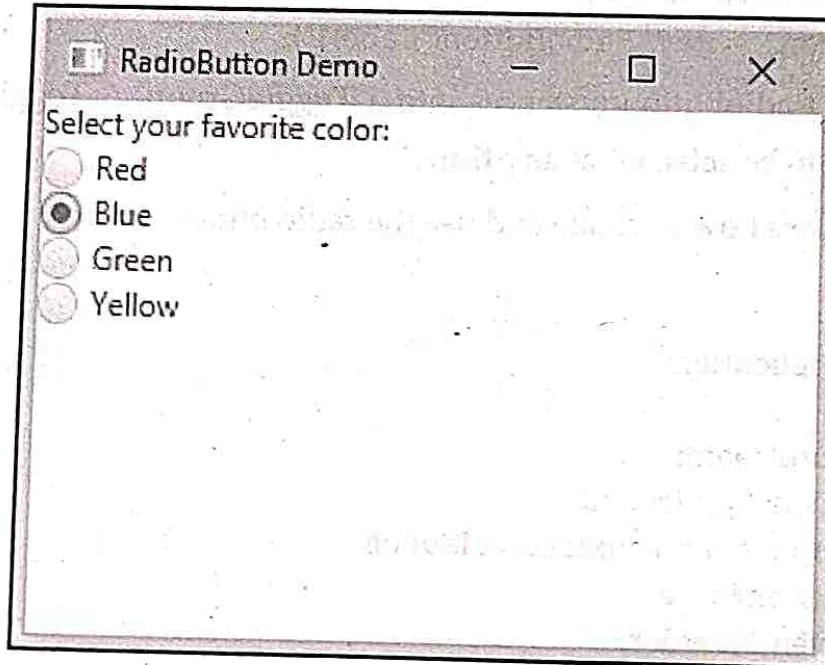
Programming Example

```
package MyJavaFXApplication;
```

```
import java.io.InputStream;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.RadioButton;
import javafx.scene.control.ToggleGroup;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
```

```
public class MyJavaFXApplication extends Application {  
    @Override  
    public void start(Stage primaryStage) throws Exception {  
        ToggleGroup group = new ToggleGroup();  
        Label L = new Label("Select your favorite color: ");  
        RadioButton rb1 = new RadioButton("Red");  
        RadioButton rb2 = new RadioButton("Blue");  
        RadioButton rb3 = new RadioButton("Green");  
        RadioButton rb4 = new RadioButton("Yellow");  
        rb1.setToggleGroup(group);  
        rb2.setToggleGroup(group);  
        rb3.setToggleGroup(group);  
        rb4.setToggleGroup(group);  
        VBox root = new VBox();  
        Scene scene = new Scene(root, 300, 200);  
        root.getChildren().addAll(L, rb1, rb2, rb3, rb4);  
        primaryStage.setTitle("RadioButton Demo");  
        primaryStage.setScene(scene);  
        primaryStage.show();  
    }  
    public static void main(String[] args) {  
        launch(args);  
    }  
}
```

Output



5.4.6 Textfield

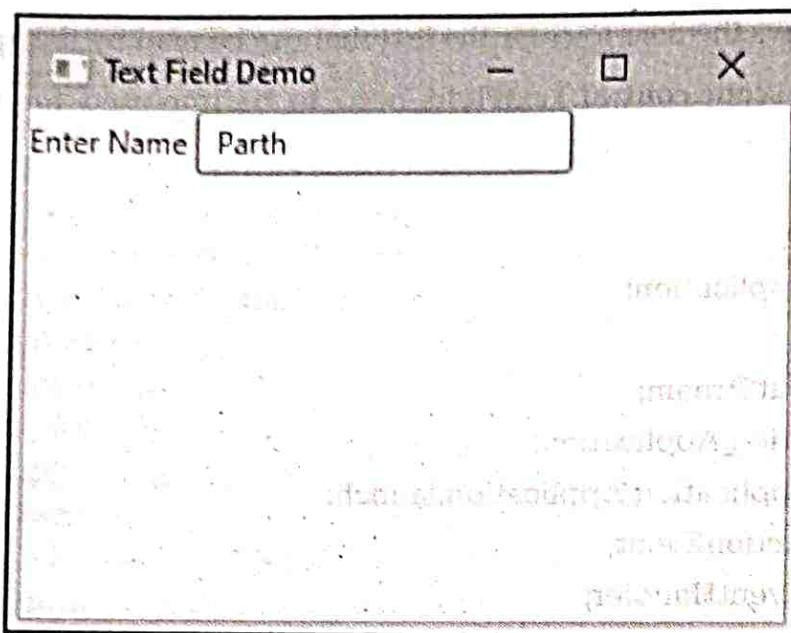
TextField control allows the user to enter the text that can be read by the application.

The package `javafx.scene.control.TextField` need to be imported for using the TextField control.

Programming Example

```
package MyJavaFXApplication;
```

```
import java.io.InputStream;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
public class MyJavaFXApplication extends Application {  
  
    @Override
    public void start(Stage primaryStage) throws Exception {
        Label L = new Label("Enter Name ");
        TextField tf = new TextField();
        GridPane root = new GridPane();
        root.addRow(0, L, tf);
        Scene scene = new Scene(root, 300, 200);
        primaryStage.setTitle("Text Field Demo");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

Output**Ex. 5.4.1 : Write a JavaFX application for creating a login form.****Sol. :**

```

package MyJavaFXApplication;

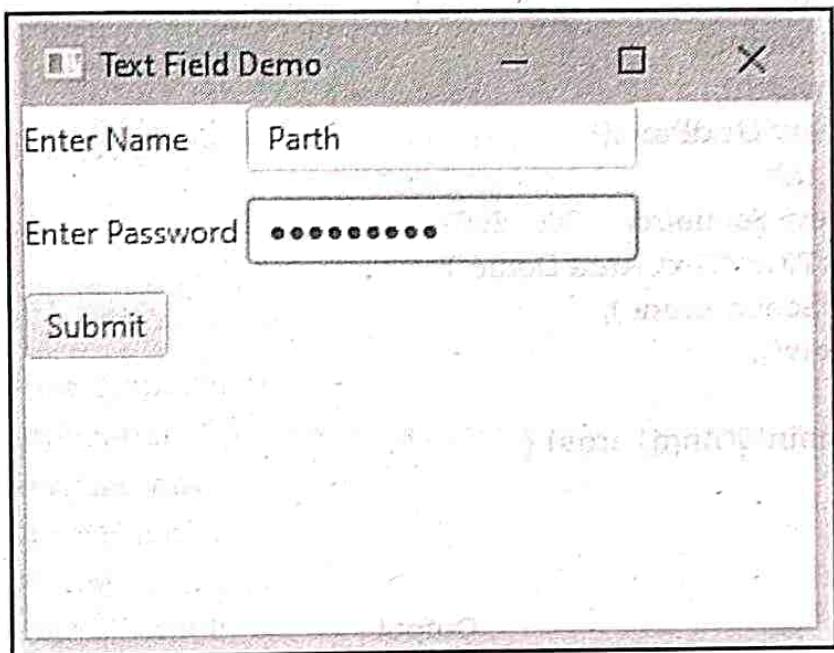
import java.io.FileInputStream;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.PasswordField;
import javafx.scene.control.TextField;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;
public class MyJavaFXApplication extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        Label L1=new Label("Enter Name ");
        TextField tf1=new TextField();
        Label L2=new Label("Enter Password ");
        TextField tf2=new PasswordField();
        Button btn=new Button("Submit");
        GridPane root=new GridPane();
        root.setVgap(10);
        root.addRow(0,L1,tf1);
    }
}
  
```

```

root.addRow(1,L2,tf2);
root.addRow(2,btn);
Scene scene = new Scene(root, 300, 200);
primaryStage.setTitle("Text Field Demo");
primaryStage.setScene(scene);
primaryStage.show();
}
public static void main(String[] args) {
    launch(args);
}
}
}

```

Output**5.4.7 TextArea**

The TextArea control allows to enter multiline text. This control is represented by the class `javafx.scene.control.TextArea`

The textarea control can be created using

```
TextArea ta=new TextArea()
```

We can set the size of the TextArea using `setPrefHeight()` and `setPrefWidth()` functions.

Following example shows how to create TextArea control

Programming Example

```
package MyJavaFXApplication;
```

```

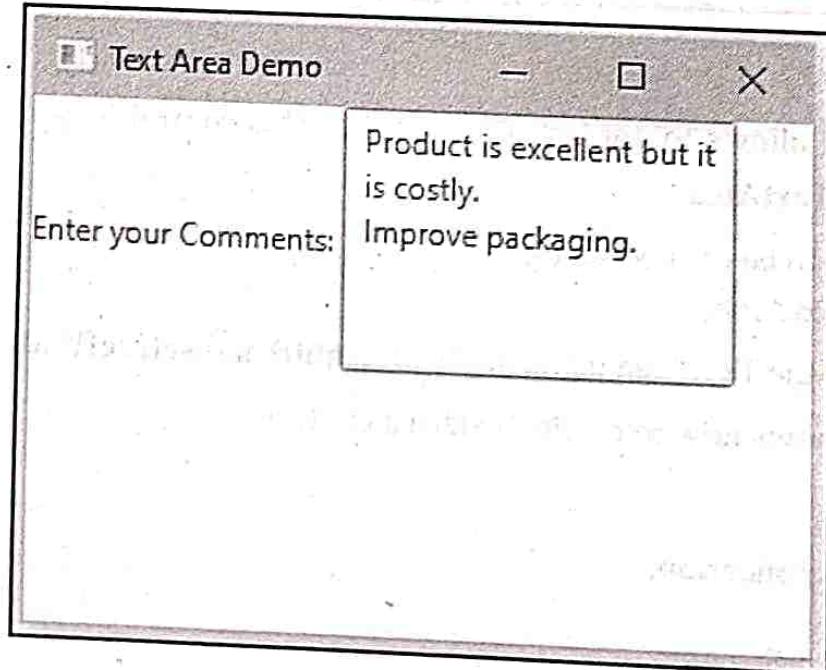
import java.io.FileInputStream;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;

```

```
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.TextArea;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;
public class MyJavaFXApplication extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        Label L = new Label("Enter your Comments: ");
        TextArea ta = new TextArea();
        double height = 100;
        double width = 150;
        ta.setPrefHeight(height);
        ta.setPrefWidth(width);
        GridPane root = new GridPane();
        root.addRow(0, L, ta);
        Scene scene = new Scene(root, 300, 200);
        primaryStage.setTitle("Text Area Demo");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

Output



5.4.8 ListView

The JavaFX ListView control enables users to choose one or more options from a predefined list of choices. The JavaFX ListView control is represented by the class `javafx.scene.control.ListView`.

The ListView can be created as follows –

```
ListView listView=new ListView()
```

The items can be added to the ListView control using `getItems().add()` method. For example

```
listView.getItems().add("Item 1");
```

```
listView.getItems().add("Item 2");
```

...

To make a ListView visible we have to add it to the scene graph or to some layout component which is then attached to the Scene object.

Following example program shows how to use ListView Control

Programming Example

```
package MyJavaFXApplication;
```

```
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.ListView;
import javafx.scene.control.SelectionMode;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;
public class MyJavaFXApplication extends Application {
```

```
@Override
```

```
public void start(Stage primaryStage) throws Exception {
```

```
    Label L=new Label("Select your favorite programming Language: ");
```

```
    ListView listView=new ListView();
```

```
    listView.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);
```

```
    listView.getItems().add("Java");
```

```
    listView.getItems().add("C++");
```

```
    listView.getItems().add("PHP");
```

```
    listView.getItems().add("Python");
```

```
    GridPane root=new GridPane();
```

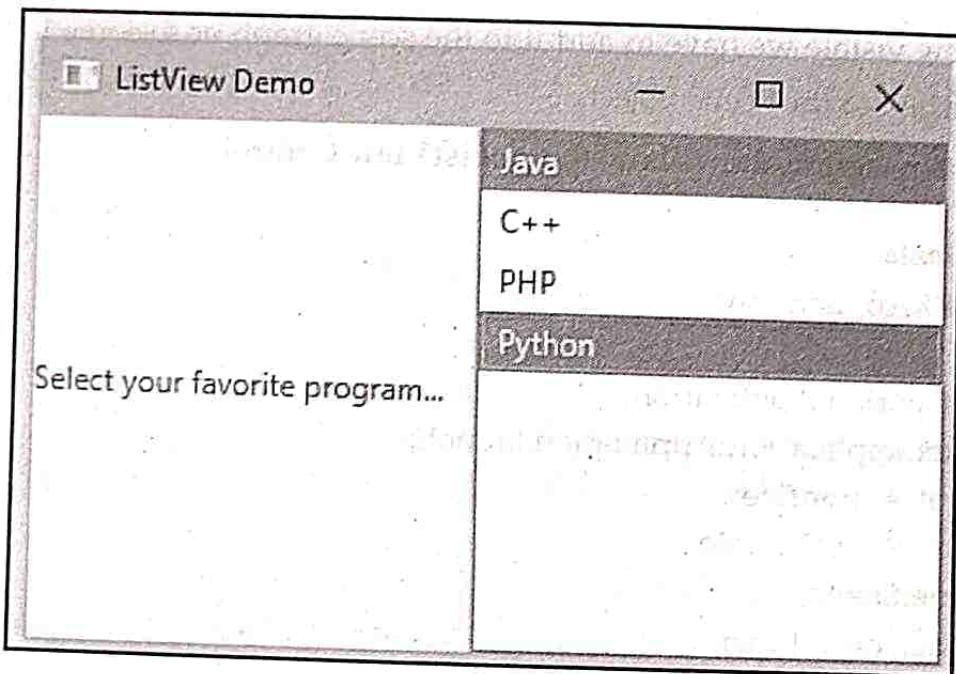
```
    root.addRow(0,L,listView);
```

```

Scene scene = new Scene(root, 350, 200);
primaryStage.setTitle("ListView Demo");
primaryStage.setScene(scene);
primaryStage.show();
}

public static void main(String[] args) {
    launch(args);
}
}

```

Output

In above code we have used

`listView.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);`

which allows multiple selection of items from listview.

5.4.9 ComboBox

We can have predefined list of choices using combo box.

This control is represented by `javafx.scene.control.ComboBox` class. Hence we need to import it at the beginning of our application program.

We can create the comboBox using following statement

`ComboBox cb=new ComboBox();`

Then we need to add the list of choices to the comboBox. This can be done using

```

cb.getItems().add("Option1")
cb.getItems().add("Option2")
...

```

And so on.

Following program illustrates the use of ComboBox control in the JavaFX application program.

Programming Example

```
package MyJavaFXApplication;
```

```
import java.io.InputStream;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.ComboBox;
import javafx.scene.control.Label;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;
```

```
public class MyJavaFXApplication extends Application {
```

```
    @Override
```

```
    public void start(Stage primaryStage) throws Exception {
```

```
        Label L = new Label("Select your favorite programming Language:");
```

```
        ComboBox cb = new ComboBox();
```

```
        cb.getItems().add("Java");
```

```
        cb.getItems().add("C++");
```

```
        cb.getItems().add("PHP");
```

```
        cb.getItems().add("Python");
```

```
        GridPane root = new GridPane();
```

```
        root.addRow(0, L, cb);
```

```
        Scene scene = new Scene(root, 350, 200);
```

```
        primaryStage.setTitle("ComboBox Demo");
```

```
        primaryStage.setScene(scene);
```

```
        primaryStage.show();
```

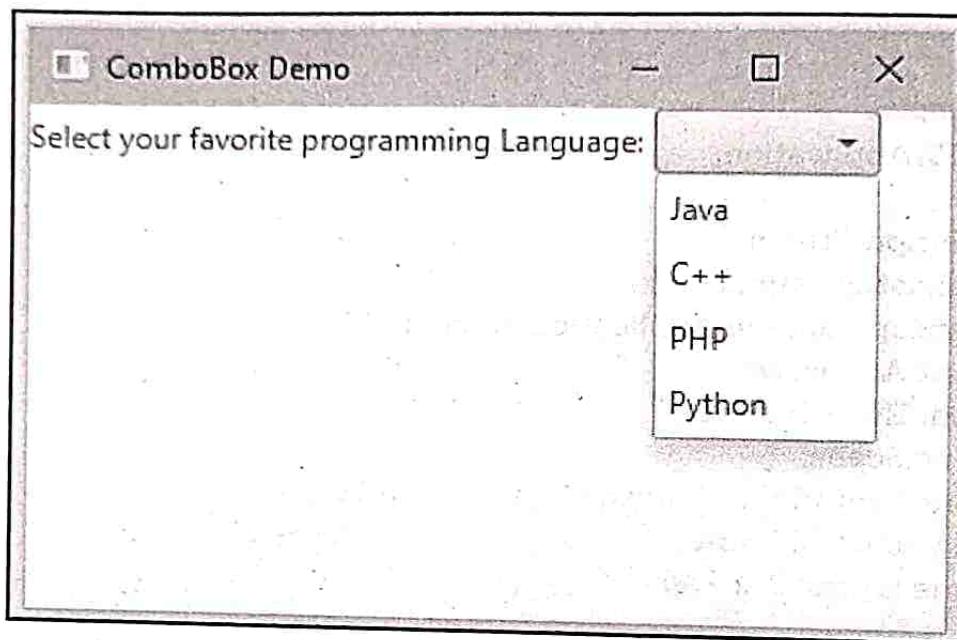
```
}
```

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

```
        launch(args);
```

```
}
```

```
}
```

Output**5.4.10 ChoiceBox**

- The ChoiceBox control shows the set of items and allows user to select a single choice. It will show the currently selected item on the top.
- By default the ChoiceBox control has no selected item. We have to specify the items from which the choice is to be made.
- For using the ChoiceBox control we need to import **javafx.scene.control.ChoiceBox** package
- Constructors of ChoiceBox Class are**

ChoiceBox(): It creates empty ChoiceBox

ChoiceBox(ObservableList items): It creates empty ChoiceBox with the given set of items.

Commonly Use Methods are

setItems(ObservableList value)	It adds all the list items to ChoiceBox instance.
setValue(T value)	Sets the single choice value at a time.
getItems()	It gives the items of the ChoiceBox.
getValue()	It reads the ChoiceBox value means get the value.
show()	Opens the list of all choices from ChoiceBox.
hide()	It closes all the list of choices from ChoiceBox.

Programming Example

```
package application;
```

```
import javafx.application.Application;  
import javafx.scene.Scene;  
import javafx.scene.layout.HBox;  
import javafx.geometry.Insets;  
import javafx.scene.control.Label;  
import javafx.scene.control.ChoiceBox;  
import javafx.stage.Stage;
```

```
public class Main extends Application {
```

```
    @Override
```

```
    public void start(Stage primaryStage) throws Exception {
```

```
        HBox root = new HBox();
```

```
        root.setPadding(new Insets(15));
```

```
        root.setSpacing(10);
```

```
        // Creating a Label
```

```
        root.getChildren().add(new Label("My Favorite color is: "));
```

```
        // Creating a ChoiceBox
```

```
        ChoiceBox<Object> choiceBox = new ChoiceBox<Object>();
```

```
        choiceBox.getItems().add("Blue");
```

```
        choiceBox.getItems().add("Black");
```

```
        choiceBox.getItems().add("White");
```

```
        choiceBox.getItems().add("Red");
```

```
        root.getChildren().add(choiceBox);
```

```
        Scene scene = new Scene(root, 300, 300);
```

```
        primaryStage.setTitle("ChoiceBox Demo");
```

```
        primaryStage.setScene(scene);
```

```
        primaryStage.show();
```

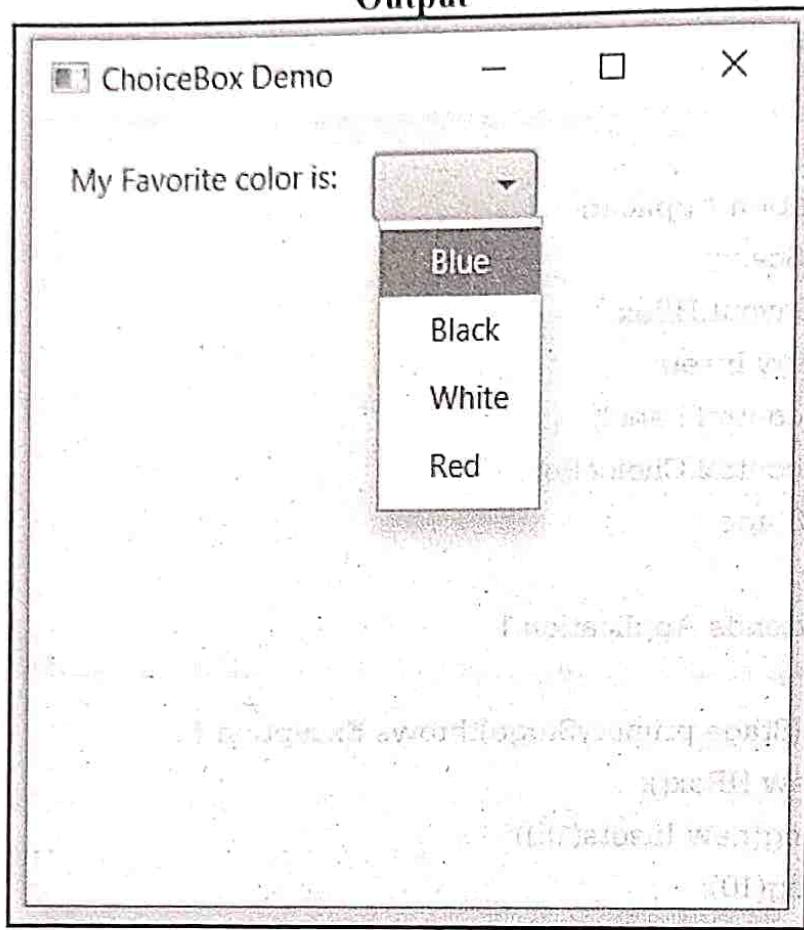
```
}
```

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

```
        launch(args);
```

```
}
```

Output



5.4.11 Scrollbar

Using the Scrollbar control the user can scroll down to the application page.

For creating this control we use `javafx.scene.control.ScrollBar` class.

Programming Example

```
package MyJavaFXApplication;

import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.ScrollBar;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;
```

```
public class MyJavaFXApplication extends Application {
```

```
    @Override
```

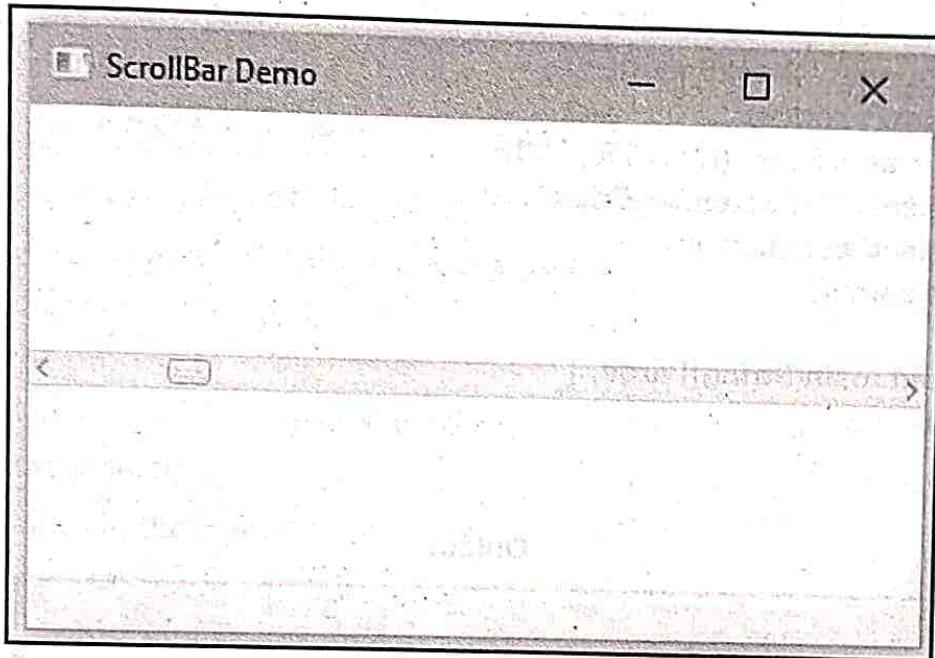
```
    public void start(Stage primaryStage) throws Exception {
        ScrollBar sb = new ScrollBar();
        StackPane root = new StackPane();
        root.getChildren().add(sb);
    }
}
```

TECHNICAL PUBLICATIONS® - an up-thrust for knowledge

```

Scene scene = new Scene(root, 350, 200);
primaryStage.setTitle("ScrollBar Demo");
primaryStage.setScene(scene);
primaryStage.show();
}
public static void main(String[] args) {
    launch(args);
}
}

```

Output

We can create the scrollbar in horizontal direction or in vertical direction. If we want the scrollbar to be displayed vertically then we call **setOrientation()** method
 sb.setOrientation(Orientation.VERTICAL);

Similarly we can set minimum, maximum or current value to scrollbar using following functions

- (1) **setMin()**
- (2) **setMax()**
- (3) **setValue()**

Following program makes use of these methods for the scrollbar control

```

package MyJavaFXApplication;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.geometry.Orientation;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.ScrollBar;
import javafx.scene.layout.StackPane;

```

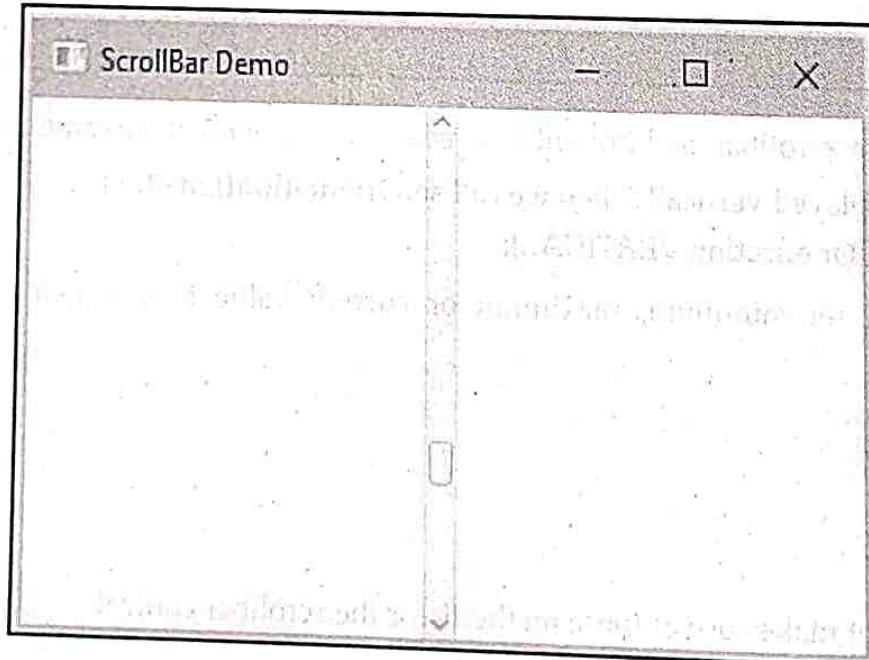
```
import javafx.stage.Stage;

public class MyJavaFXApplication extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        ScrollBar sb = new ScrollBar();
        sb.setMin(0);
        sb.setMax(50);
        sb.setValue(25);
        sb.setOrientation(Orientation.VERTICAL);
        StackPane root = new StackPane();
        root.getChildren().add(sb);
        Scene scene = new Scene(root, 350, 200);
        primaryStage.setTitle("ScrollBar Demo");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

Output



5.5 Layouts

- The arrangement of various components(nodes) in a scene within the container is called Layout of the container.
- For using the layout we must import the package `javafx.scene.layout`. The class named `Pane` is the base class for all the layouts in JavaFX.
- Various layout panes are summarized in the following table.

Layout Pane	Description
HBox	This layout pane arranges all the nodes in a single horizontal row.
VBox	This layout pane arranges all the nodes in a single vertical column.
StackPane	In this layout pane, the nodes are placed on top of each other in the center of the pane.
GridPane	The nodes are placed in cell in two dimensional grid.
FlowPane	This is a layout pane, in which the nodes are placed row by row horizontally or column by column vertically.
BorderPane	The nodes can be placed at the top, bottom, left, right or at the center regions.

For adding the nodes to the layout manager follow the following step -

(1) Instantiate the respective layout class. For example

```
VBox root=new VBox();
```

(2) Then set the properties of the layout.

For example - if we want the spacing between the created nodes then we use

```
root.setSpacing(10);
```

(3) Adding nodes to the layout object, for example -

```
root.getChildren().addAll(<NodeObjects>);
```

5.5.1 FlowPane

- A JavaFX FlowPane is a layout component which lays out its child components either vertically or horizontally, and which can wrap the components onto the next row or column if there is not enough space in one row.
- The JavaFX FlowPane layout component is represented by the class `javafx.scene.layout.FlowPane`
- Various methods defined in FlowPane layout are –

Method	Description
ObjectProperty<Pos> alignment	The overall alignment of the flowpane's content within its width and height.
ObjectProperty<HPos> columnHalignment	The horizontal alignment of nodes within each column of a vertical flowpane.
ObjectProperty<VPos> rowValignment	The vertical alignment of nodes within each row of a horizontal flowpane.
DoubleProperty hgap	The amount of horizontal space between each node in a horizontal flowpane or the space between columns in a vertical flowpane.

<code>DoubleProperty vgap</code>	The amount of vertical space between each node in a vertical flowpane or the space between rows in a horizontal flowpane.
<code>ObjectProperty<Orientation> orientation</code>	The orientation of this flowpane.

- The FlowPane can be created using following syntax

```
FlowPane flowpane = new FlowPane();
```

Programming Example

```
package myjavafxapplication;
```

```
import javafx.application.Application;
```

```
import javafx.scene.Scene;
```

```
import javafx.scene.control.Button;
```

```
import javafx.scene.layout.FlowPane;
```

```
import javafx.stage.Stage;
```

```
public class MyJavaFXApplication extends Application {
```

```
@Override
```

```
public void start(Stage primaryStage) {
```

```
    FlowPane root=new FlowPane();
```

```
    Button B1 = new Button("One");
```

```
    Button B2 = new Button("Two");
```

```
    Button B3 = new Button("Three");
```

```
    Button B4 = new Button("Four");
```

```
    Button B5 = new Button("Five");
```

```
    Button B6 = new Button("Six");
```

```
    root.getChildren().add(B1);
```

```
    root.getChildren().add(B2);
```

```
    root.getChildren().add(B3);
```

```
    root.getChildren().add(B4);
```

```
    root.getChildren().add(B5);
```

```
    root.getChildren().add(B6);
```

```
    Scene scene=new Scene(root,150,150);
```

```
    primaryStage.setScene(scene);
```

```
    primaryStage.setTitle("FlowPane Demo");
```

```
    primaryStage.show();
```

```
}
```

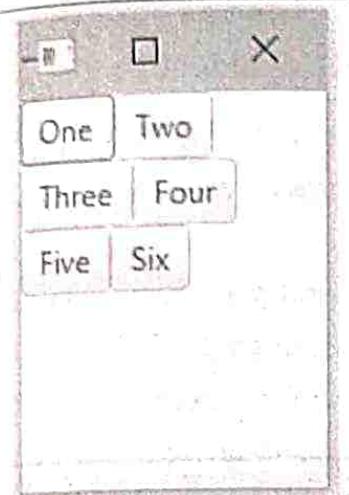
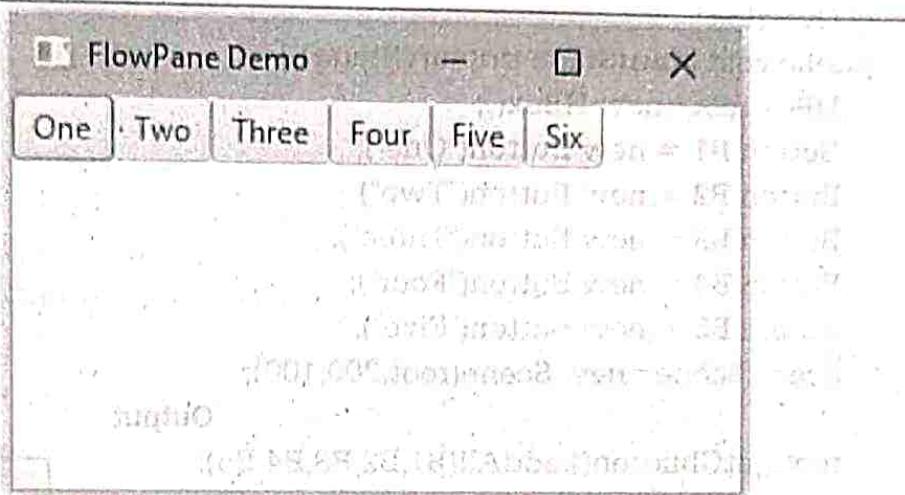
```
public static void main(String[] args) {
```

```
    launch(args);
```

```
}
```

```
}
```

Output

	
Initially we get this window	If we expand the window horizontally, the buttons get arranged horizontally automatically

5.5.2 HBox

- The HBox layout arranges the children in the form of horizontal rows.
- The HBox class extends Pane class.
- It requires `javafx.scene.layout.HBox` class which provides all the required methods of this pane.
- Various methods used for layout using HBox are -

Method	Purpose
<code>getAlignment()</code>	It returns the value of alignment property.
<code>getSpacing()</code>	It returns the spacing between its children.
<code>setAlignment(Pos Value)</code>	It sets the alignment of the HBox.
<code>getChildren()</code>	It returns the nodes in HBox.

- The constructors of HBox class are -

`HBox():` Creates an HBox object with no nodes.

`HBox(double s):` Creates an HBox with spacing in between nodes.

Programming Example

```
package myjavafxapplication;
```

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;
```

```
public class MyJavaFXApplication extends Application {
```

```

@Override
public void start(Stage primaryStage) {
    HBox root = new HBox();
    Button B1 = new Button("One");
    Button B2 = new Button("Two");
    Button B3 = new Button("Three");
    Button B4 = new Button("Four");
    Button B5 = new Button("Five");
    Scene scene = new Scene(root, 200, 100);
}
public static void main(String[] args) {
    launch(args);
}
}

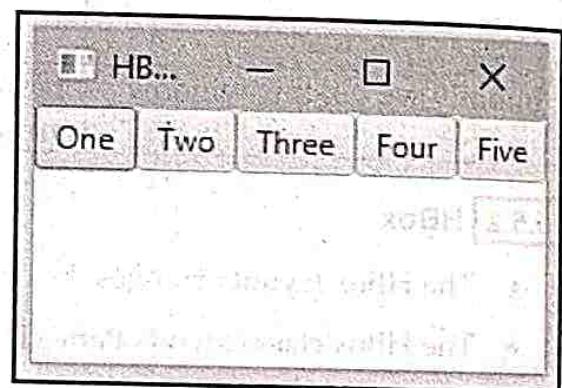
```

Output

```

root.getChildren().addAll(B1,B2,B3,B4,B5);
primaryStage.setScene(scene);
primaryStage.setTitle("HBOX Demo");
primaryStage.show();
}

```



Program Explanation : In above code,

- (1) We have created four button elements.
- (2) The layout pane is set as HBox. All the button elements are attached to the HBox pane by using `getChildren().addAll` function.
- (3) This pane is then added to the Scene.
- (4) Using `setScene` method this scene is then added to the PrimaryStage.

5.5.3 VBox

- The VBox layout arranges the children in the form of vertical rows.
- The VBox class extends Pane class.
- It requires `javafx.scene.layout.VBox` class which provides all the required methods of this pane.
- Various methods used for layout using VBox are -

Method	Purpose
<code>getAlignment()</code>	It returns the value of alignment property.
<code>getSpacing()</code>	It returns the spacing between its children.
<code>setSpacing(value)</code>	It sets some spacing around the nodes of VBox
<code>setAlignment(Pos Value)</code>	It sets the alignment of the VBox
<code>getChildren()</code>	It returns the nodes in VBox

- The constructors of VBox class are -

VBox(): Creates an VBox object with no nodes.

VBox(double s): Creates an VBox with spacing in between nodes.

Programming Example

```
package myjavafxapplication;
```

```
import javafx.application.Application;
```

```
import javafx.scene.Scene;
```

```
import javafx.scene.control.Button;
```

```
import javafx.scene.layout.VBox;
```

```
import javafx.stage.Stage;
```

```
public class MyJavaFXApplication extends Application {
```

```
    @Override
```

```
    public void start(Stage primaryStage) {
```

```
        VBox root = new VBox();
```

```
        Button B1 = new Button("One");
```

```
        Button B2 = new Button("Two");
```

```
        Button B3 = new Button("Three");
```

Output

```
        Button B4 = new Button("Four");
```

```
        Button B5 = new Button("Five");
```

```
        Scene scene = new Scene(root, 100, 150);
```

```
        root.getChildren().addAll(B1, B2, B3, B4, B5);
```

```
        primaryStage.setScene(scene);
```

```
        primaryStage.setTitle("VBOX Demo");
```

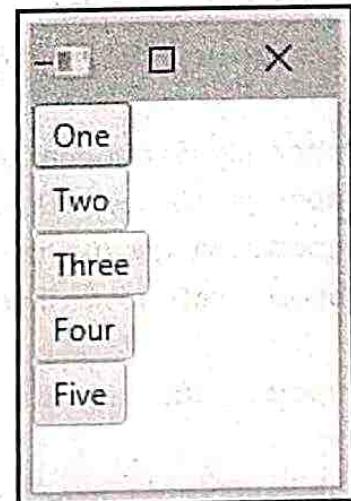
```
        primaryStage.show();
```

```
}
```

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

```
        launch(args);
```

```
}
```



5.5.4 BorderPane

- BorderPane lays out children in top, left, right, bottom, and center positions.
- It can be used to create the classic looking application layouts.
- It is represented by `javafx.scene.layout.BorderPane` class.
- Various methods of BorderPane layout are

Method	Description
<code>setBottom(node value)</code>	This method sets the bottom node of the border pane.
<code>setCenter(node value)</code>	This method sets the center node of the border pane.
<code>setLeft(node value)</code>	This method sets the left node of the border pane.
<code>setRight(node value)</code>	This method sets the right node of the border pane.
<code>setTop(node value)</code>	This method sets the top node of the border pane.

- The BorderPane layout is created as follows –
 - (1) `BorderPane()`: Creates a new Border Pane.
 - (2) `BorderPane(Node c)`: Creates a new Border Pane with specified node at center.
 - (3) `BorderPane(Node center, Node top, Node right, Node bottom, Node left)`: Creates an BorderPane layout with the given Nodes to use for each of the main layout areas of the Border Pane.

Programming Example

```
package myjavafxapplication;

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;

public class MyJavaFXApplication extends Application
{
    @Override
    public void start(Stage primaryStage)
    {
        Button B_top = new Button("One");
        Button B_center = new Button("Two");
        Button B_bottom = new Button("Three");
    }
}
```

```
Button B_left = new Button("Four");
```

Output

```
Button B_right = new Button("Five");
```

```
BorderPane root = new BorderPane();
```

```
root.setTop(B_top);
```

```
root.setCenter(B_center);
```

```
root.setBottom(B_bottom);
```

```
root.setLeft(B_left);
```

```
root.setRight(B_right);
```

```
Scene scene=new Scene(root,250,250);
```

```
primaryStage.setScene(scene);
```

```
primaryStage.setTitle("BorderPane
```

```
Demo");
```

```
primaryStage.show();
```

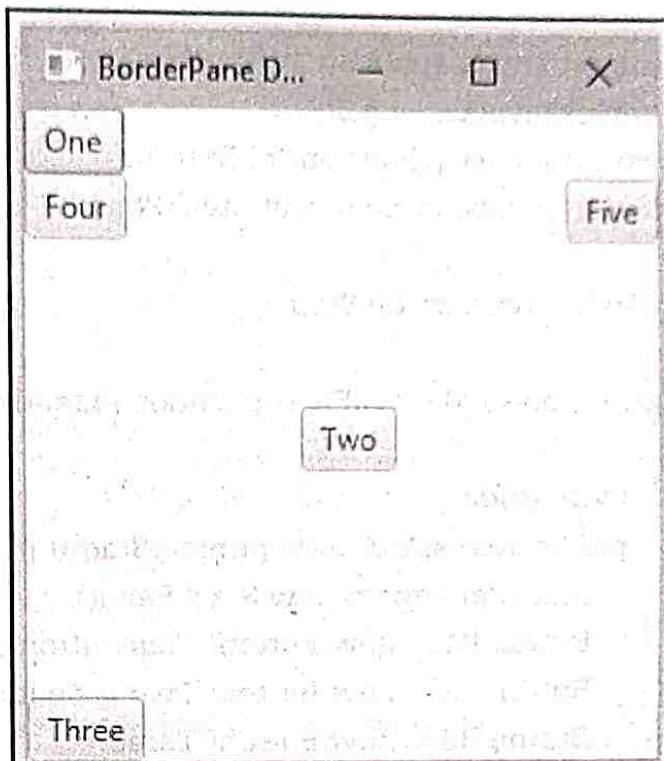
```
}
```

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

```
launch(args);
```

```
}
```

```
}
```

**5.5.5 StackPane**

- StackPane is a layout in which every new node is placed on the top of previous node.
- It makes use of `javafx.scene.layout.StackPane` class.
- Various methods of StackPane layout are -

Method	Purpose
<code>getAlignment()</code>	Returns the alignment of the StackPane.
<code>getAlignment(Node c)</code>	Returns the node's alignment
<code>getMargin(Node c)</code>	Returns the insets of the node
<code>setAlignment(Node n, Pos v)</code>	Sets the alignment of the node which is a part of StackPane
<code>setAlignment(Pos v)</code>	Sets the alignment of the StackPane
<code>setMargin(Node n, Insets v)</code>	Sets the margin of the node which is a part of StackPane.

- The constructor for StackPane layout is

1. `StackPane()`

2. `StackPane(Node... Children)`

Programming Example

```
package myjavafxapplication;
```

```

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;

import javafx.stage.Stage;

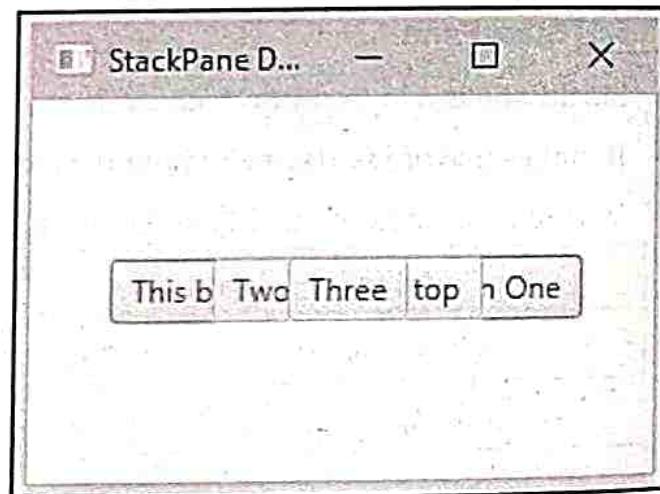
public class MyJavaFXApplication extends Application {

    @Override
    public void start(Stage primaryStage) {
        StackPane root = new StackPane();
        Button B1 = new Button("This button is with caption One");
        Button B2 = new Button("Two is on its top");
        Button B3 = new Button("Three");

        Scene scene = new Scene(root, 100, 150);
        root.getChildren().addAll(B1, B2, B3);
        primaryStage.setScene(scene);
        primaryStage.setTitle("StackPane Demo");
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}

```

Output**5.5.6 GridPane**

GridPane places its nodes into a grid of rows and columns. Nodes may span multiple rows or columns. **GridPane** is the most flexible built-in layout pane.

The **GridPane** is given by

Button1	Button2
Button3	Button4
Button5	Button6

- Various methods used in GridPane are -

Method	Description
void add(Node child, int columnIndex, int rowIndex)	Adds a child to the gridpane at the specified column, row position.
void add(Node child, int columnIndex, int rowIndex, int colspan, int rowspan)	Adds a child to the gridpane at the specified column, row position and spans.
void addColumn(int columnIndex, Node... children)	This method is for placing the specified nodes sequentially in a given column of the gridpane.
void addRow(int rowIndex, Node... children)	This method is for placing the specified nodes sequentially in a given row of the gridpane.
ObjectProperty<Pos> alignmentProperty()	The alignment of of the grid within the gridpane's width and height.
Pos getAlignment()	Gets the value of the property alignment.
void setHgap(double value)	Sets the value of the property hgap.
void setVgap(double value)	Sets the value of the property vgap

The constructor used for GridPane layout is

```
GridPane gridPane = new GridPane();
```

Following example shows the use of GridPane layout

Programming Example

```
package myjavafxapplication;
```

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.GridPane;

import javafx.stage.Stage;

public class MyJavaFXApplication extends Application {

    @Override
    public void start(Stage primaryStage) {
        GridPane grid=new GridPane();
        Button B1 = new Button("One");
        Button B2 = new Button("Two");
        Button B3 = new Button("Three");
        Button B4 = new Button("Four");
        Button B5 = new Button("Five");
        Button B6 = new Button("Six");
    }
}
```

```
grid.add(B1,0,0);
grid.add(B2,1,0);
grid.add(B3,2,0);
grid.add(B4,0,1);
grid.add(B5,1,1);
```

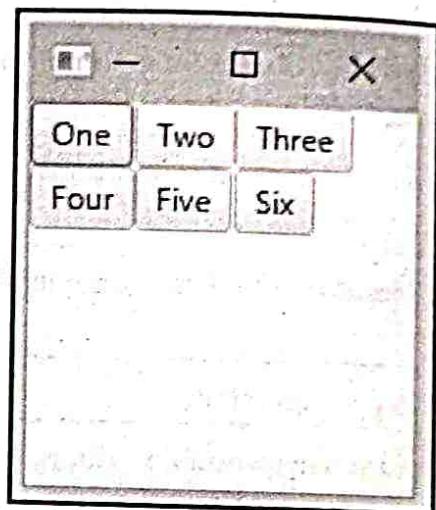
Output

```
grid.add(B6,2,1);
Scene scene=new Scene(grid,150,150);
```

```
primaryStage.setScene(scene);
primaryStage.setTitle("GridPane Demo");
primaryStage.show();
}
```

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

}



5.6 Menus

5.6.1 Menu

- In JavaFX we can create a menu using a **Menu** class.
- For using the **Menu** class in our JavaFX application we need to import **javafx.scene.control.Menu** so that the methods associated with this class can be used in the application.
- **Menu** is a popup menu that contains several menu items that are displayed when the user clicks a menu. The user can select a menu item from the menu.
- **Constructors for Menu class are**

Menu()	Creates an empty menu.
Menu(String str)	Creates a menu with string str as label.
Menu(String str,Node nd)	Creates a menu with string str as label and nd as graphics node.
Menu(String str,Node nd,Item...i)	Creates a menu with string str as label, nd as graphics node and add given item to the list

- Commonly used Methods are

<code>getItems()</code>	returns the items of the menu
<code>hide()</code>	hide the menu
<code>show()</code>	show the menu
<code>getMenus()</code>	The menus to show within thisMenuBar.

5.6.2 Menu Bars

- MenuBar is just like a navigation bar with menus on it. The menubar is located at the top of the screen.
- The Menubar class is available `javafx.scene.control.MenuBar` package.
- Constructors for Menubar class are

<code>MenuBar()</code>	Creates a new empty Menubar
<code>MenuBar(Menu... m)</code>	Creates a new menubar with given set of menus

5.6.3 MenuItem

- A MenuItem is a basic item that goes on a menu.
- Following Fig. 5.6.1 shows the menu,menubar and menuitems.

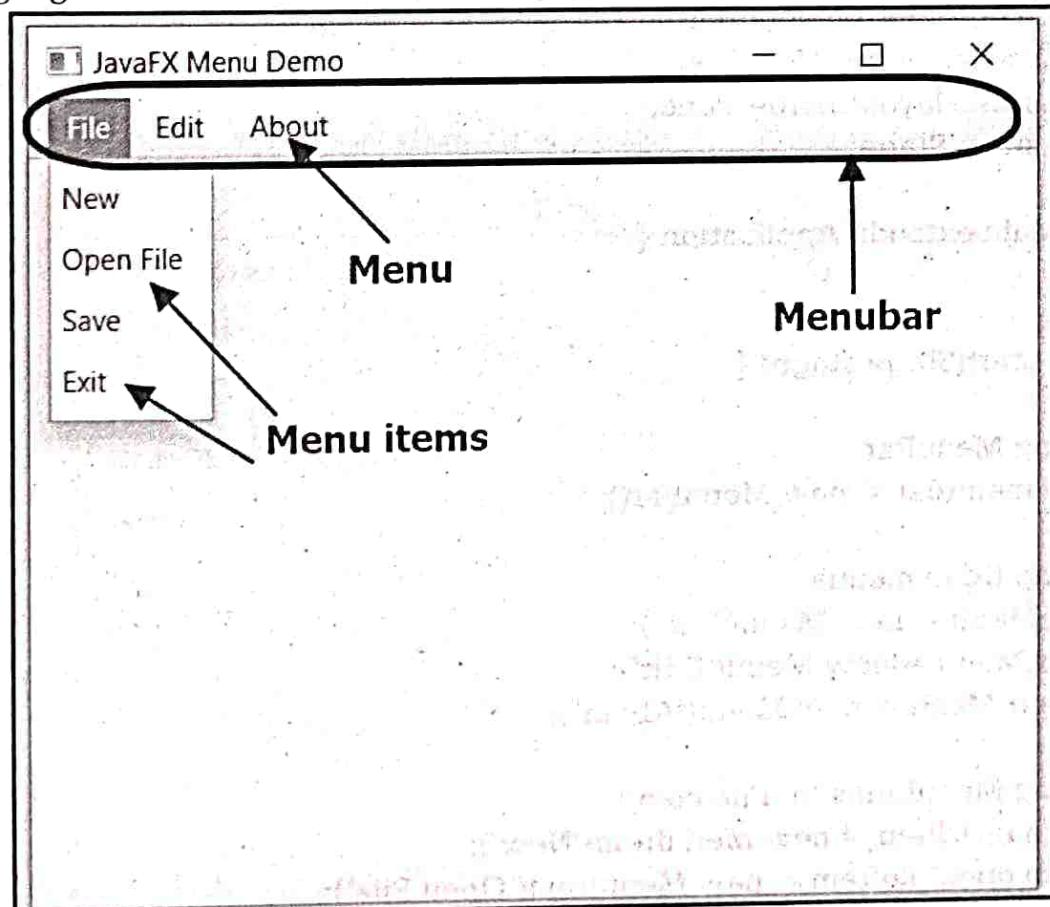


Fig. 5.6.1 Representing a menu

Steps for creating JavaFX application for creating Menu**Step 1 :** Create a Menu bar is the first step, MenuBar can be instantiated using new

```
MenuBar menuBar = new MenuBar();
```

Step 2 : Now create the menu. The name of the menu is passed as an argument to the Menu class.

```
Menu fileMenu = new Menu("File");
```

Step 3 : Create the menuitem. The name of the menu is passed as an argument to the MenuItem class.

```
MenuItem newItem = new MenuItem("New");
```

Step 4 : Add menu items to the menu using add or addAll method

```
fileMenu.getItems().addAll(newItem, openFileItem, saveItem, exitItem);
```

Step 5 : Add Menu to Menu Bar using add or addAll method.

```
menuBar.getMenus().addAll(fileMenu, editMenu, aboutMenu);
```

Programming Example

```
package application;
```

```
import javafx.application.Application;  
import javafx.scene.Scene;  
import javafx.scene.control.Menu;  
import javafx.scene.control.MenuBar;  
import javafx.scene.control.MenuItem;  
import javafx.scene.layout.BorderPane;  
import javafx.stage.Stage;
```

```
public class Main extends Application {
```

```
    @Override
```

```
    public void start(Stage stage) {
```

```
        // Creating MenuBar
```

```
        MenuBar menuBar = new MenuBar();
```

```
        // Creating three menus
```

```
        Menu fileMenu = new Menu("File");
```

```
        Menu editMenu = new Menu("Edit");
```

```
        Menu aboutMenu = new Menu("About");
```

```
        // Creating MenuItem for File menu
```

```
        MenuItem newItem = new MenuItem("New");
```

```
        MenuItem openFileItem = new MenuItem("Open File");
```

```
        MenuItem saveItem = new MenuItem("Save");
```

```
        MenuItem exitItem = new MenuItem("Exit");
```

```
// Creating MenuItem for Edit menu
MenuItem cutItem = new MenuItem("Cut");
MenuItem copyItem = new MenuItem("Copy");
MenuItem pasteItem = new MenuItem("Paste");

// There is no menu item for About menu

// Now Adding menuItems to the Menus
fileMenu.getItems().addAll(newItem, openFileItem, saveItem, exitItem);
editMenu.getItems().addAll(cutItem, copyItem, pasteItem);

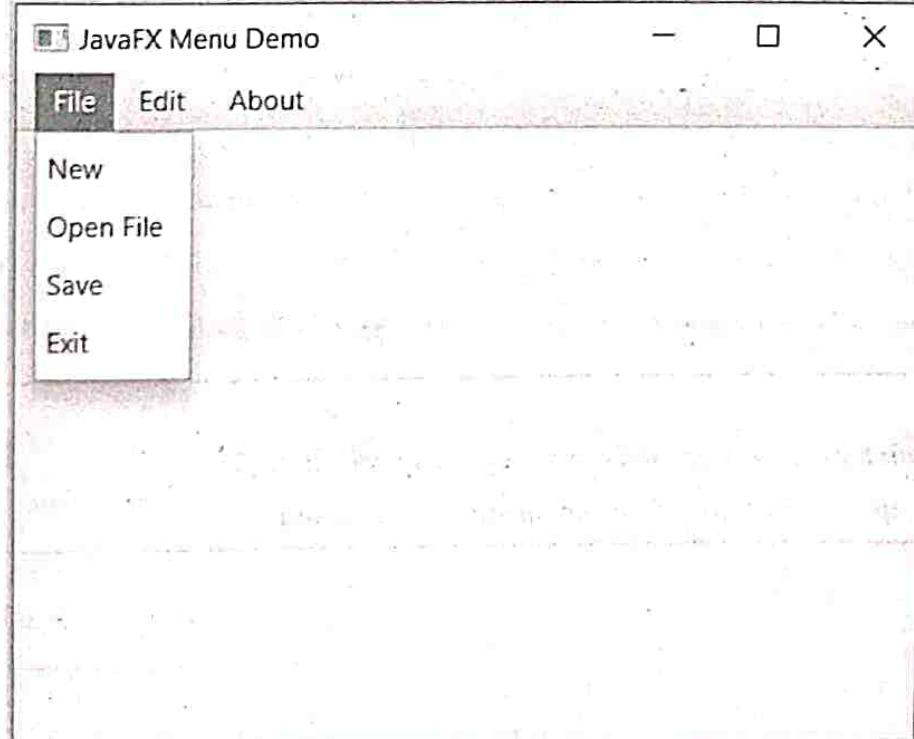
// Adding Menus to the MenuBar
menuBar.getMenus().addAll(fileMenu, editMenu, aboutMenu);

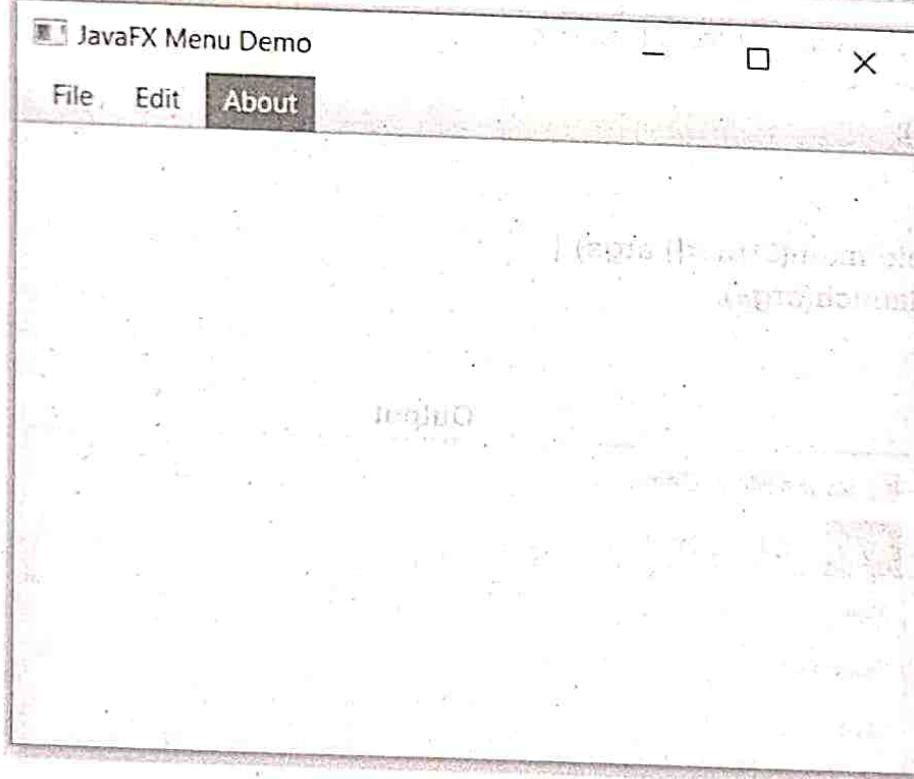
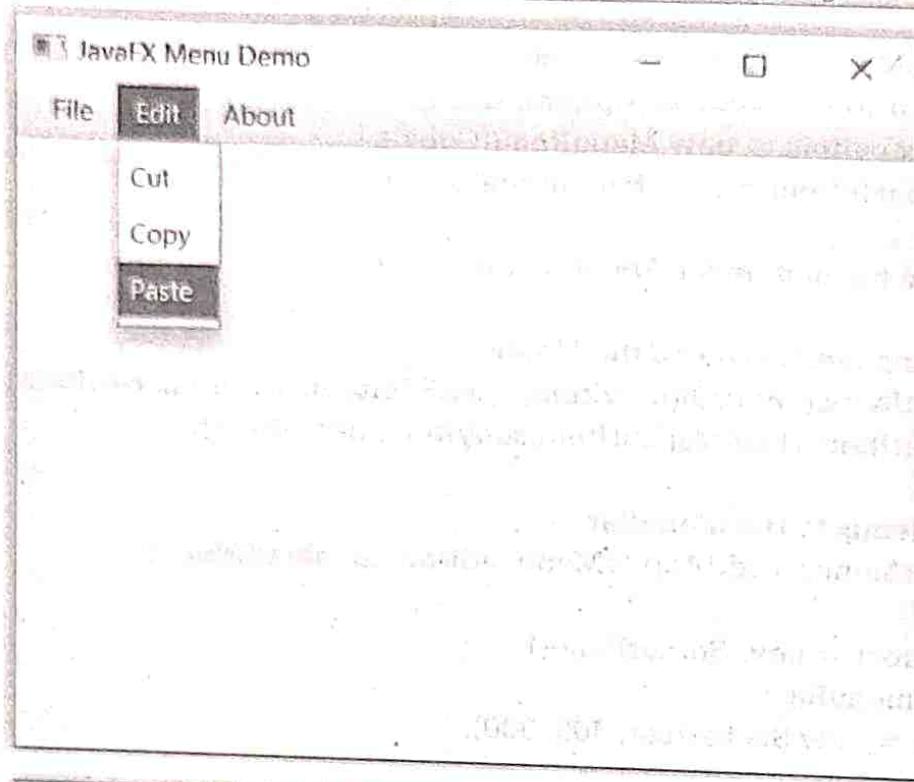
BorderPane root = new BorderPane();
root.setTop(menuBar);
Scene scene = new Scene(root, 400, 300);

stage.setTitle("JavaFX Menu Demo");
stage.setScene(scene);
stage.show();
}

public static void main(String[] args) {
    Application.launch(args);
}
```

Output



**Review Questions**

1. Write and explain the constructors and commonly used methods for Menu.
2. Write a JavaFX application to display menu, menubar and menuitems.

5.7 Two Marks Questions with Answers

Q.1 What is JavaFX ?

Ans. :

- JavaFX is a set of graphics and media packages that enables developers to design, create, test, debug and deploy rich client applications that operate consistently across different platforms.
- JavaFX makes it easier to create desktop applications and games in Java.

Q.2 What are various features of JavaFX ?

Ans. :

- 1) JavaFX library is written in java. Hence developer find it easy to learn and write the applications in JAVAFX.
- 2) JAVAFX library creates UI controls using which any GUI based application can be developed.
- 3) It provides the classes for 2D and 3D graphics.
- 4) JavaFX contains a set of ready-to-use chart components, so you don't have to code that from scratch every time you need a basic chart.

Q.3 Is JavaFX Open Source ?

Ans. : Yes.

Q.4 What are the main components of JavaFX application ?

Ans. : Every JAVAFX program is divided into three main components - Stage, Scene and Node and Scene graph. Refer Fig. 5.1.1.

Q.5 Which method is used to launch JavaFX application ?

Ans. : The launch() method is used to launch JavaFX application.

Q.6 Which method is used in which we write the code for JavaFX application ?

Ans. : The start() method is used in which we write the code for JavaFX application.

Q.7 Explain the terms - Stage and scene used in JavaFX application.

Ans. :

- 1) **Stage** : Stage is like main frame of the program. It acts as a container for all the objects of JavaFX application. The most commonly used stage is a PrimaryStage.
- 2) **Scene** : The JavaFx.scene.Scene class provides the methods to deal with the scene object. It contains the nodes or all the contents of Scene graph.

Q.8 Explain the terms - Scene graph and node used in JavaFX.**Ans. :**

- 1) The scene graph is a collection of various nodes.
- 2) The node is an element that can be visualized on the screen. The node can be button, textbox, radio button and so on.

Q.9 What is the use of pane in JavaFX application ?**Ans. :**

- Pane is a **container class** using which the UI components can be placed at any desired location with any desired size.
- Generally you place a node inside a pane and then place pane into a scene. Actually node is any visual component such as UI controls, shapes, or a image view.

Q.10 What is property binding in JavaFX ?

Ans. : Property binding is a technique that enables target object to bind with the source object. Due to this mechanism, when the value in source object changes then the target property also changes automatically.

Q.11 What is Event ?

Ans. : Event means any activity that interrupts the current ongoing activity.

For example : When user clicks button then it generates an event. To respond to button click we need to write the code to process the button clicking action.

Q.12 What is event source object ?

Ans. : The object that generates the event is called event source object. For example - If the event gets generated on clicking the button, then button is the event source object.

Q.13 Explain the terms - event handler and event listener.

Ans. : **Event Handler** : The event handling code written to process the generated event is called event handler.

Event Listener : The task of handling an event is carried out by event listener. When an event occurs, first of all an event object of the appropriate type is created. This object is then passed to a Listener.

Q.14 Explain the event methods in JavaFX that are associated with keyboard.

Ans. : Refer section 5.3.2.

Q.15 Explain any two UI controls used in JavaFX application.**Ans :**

Control	Description
Label	Label is a control that displays simple text on the screen. It is generally used to describe the other user controls.
Button	Button component is used to control specific function of the application.
RadioButton	The radio button is used to provide the choices to the user. The radio button can be selected or deselected.
CheckBox	The checkbox is used to provide various choices to the user. The check box can be checked(True) or unchecked(False).

Q.16 What is layoutpane ?**Ans. :**

- The arrangement of various components(nodes) in a scene within the container is called layout of the container.
- For using the layout we must import the package `javafx.scene.layout`. The class named `Pane` is the base class for all the layouts in JavaFX.

Q.17 Enlist various layout panes used in any JavaFX application ?**Ans. :** Refer section 5.5.**Q.18 Explain the terms - Menubar and Menu item.****Ans. :**

- `MenuBar` is just like a navigation bar with menus on it. The menubar is located at the top of the screen.
- A `MenuItem` is a basic item that goes on a menu.

