Object Oriented Programming

UNIT :- III

Methods and Inheritance in JAVA

Important Topic

1. Abstract Class and Method
2. Array, Multi Dimensional Array
3. Strings
4. Wrapper Classes
5. JAVA Enum
6. Inheritance in Java
7. The Super Keyword in Java, Constructors
8. Method Overriding
9. Multilevel Inheritance
10. Dynamic method dispatch
11. Using final with inheritance

Arrays and Strings

Outline

- Introduction to Java array
- One-dimensional array- creation
- Two-dimensional array
- Introduction to Java String
- Declaration and creation
- Different String methods
- String array

Introduction to Java Array

- It is a collection of related data items, that share a common name
- Also called subscripted variable.
- So complete set of values of similar type is referred to as an *Array,* whereas the individual values are called *elements.*
- Purpose of array is to make program more concise and efficient, this is because array has ability to represent collection of related item, and to refer to an item it has unique subscript or index.
- Remember array index can be integer constant, integer variable or expression that yield integer value.

Creating an Array(1D)

- Creation of array involves three steps:

1. Declaring the array

2. Creating memory locations

3. Putting values into the memory locations(initialization)

Declaring the array

- Arrays in Java may be declared in two ways;

  form1:

```
type arrayname[];
```

  form2:

```
type [] arrayname;
```

- Examples
  - ✓ int marks[];
  - ✓ float percentage[];
  - ✓ int [] number;
- Remember, we do not enter the size of the array in the declaration.

Creating memory locations

- For creation of array, Java allows us to use **new** operator.

```
arrayname = new type[size];
```

- Example:
  - ✓ number = **new** int[5];
  - ✓ average = **new** float[6];
- This step allocates required space for array, here in example array **number** and **average** are of type integer and float resp., so **number** refers to array of five integers and **average** refers to array of six floating point values.
- It is possible to combine the steps of declaration and creation of array as;

  int number[ ] = **new** int[5];

Initialization of Arrays

- Process of assigning values to array elements is called ***initialization***
- Syntax

```
arrayname [subscript]= value;
```

- We can also initialize array at the time of its declaration.

  int number[]={1,2,3,4,5};

- We can also use **for** loop to initialize the array.

```
Scanner input =new Scanner(System.in);
int x[]=new int [5];     //Declare and create an array
System.out.println("Define the size of array");
int n=input.nextInt(); //Define a size  of an array
System.out.println("Enter the array elements");
for(int i=0;i<n;i++)     // Initialize the array elements
{
     x[i]=input.nextInt();
}
    System.out.println("Display the array elements");
for(int i=0;i<n;i++)        // Display the array elements
{
     System.out.println("x["+i+"]="+x[i]);
}
```

Array Length

- In Java, all the array store the allocated space(size) in a variable called ***length.***
- To get the length of array we have syntax;

  int variable_name=arrayname.length;
- Example:

  int number[]=**new** int [5];

  int a= number.length;

  then, 5 value is value assigned to integer variable a.

Two Dimensional Array

- 2D Arrays in Java declare and create in two ways;

form1:

```
type arrayname[][];
arrayname=new int[size][size];
```

form2:

```
type arrayname [][]=new int[size][size];
```

- Examples
  - ✓ int marks [] [] = new int[3][4];
  - ✓ int table [2][3]= {0,0,0,1,1,1};
  - ✓ int table [2][3]= {{0,0,0},{1,1,1}};
- Remember, we do not enter the size of the array in the declaration.

Strings- Introduction

- Strings represent sequence of characters.
- The easiest way of strings representation in Java is by using *character Array.*
- *Example:_ char charArray[]=new char[4];*

  charArray[0]='J';

  charArray[1]='A';

  charArray[2]='V';

  charArray[3]='A';
- *Character array do not support the range of operations we may like to perform on strings*

Strings

- In Java, strings are class objects and implemented by using two class namely, String and StringBuffer.
- In Java, we have in built class **String.**
- So, Java strings are the objects of class **String**.
- ***Objects of string are immutable which means a constant and cannot be changed once created.***

Creating a Strings

- *There are two way to create a sting in JAVA*

1) String Literals

String s= "PICT";

2) By using new Keyword

Declaration and Creation using new Keyword

- Syntax for Java string declaration and creation is,

String *stringname;* ← declaration

***stringname* = new String (" string ");** ← creation

- These two statements may be combined as follows;

String *stringname* = **new String** (" string ");

  - ✓ Here **String** is the inbuilt class and **stringname** is the object of that class.

- Example

```
String name= new String("PICT");
```

String Methods

How to call Method	Task performed by method
s1.equalsIgnoreCase(s2)	Returns true if s1=s2,ignoring the case of characters
s1.equals(s2)	Returns 'true' if s1 is equal to s2
s1.length()	Gives length of string s1
s1.compareTo(s2)	Returns negative if s1<s2, positive if s1>s2, and zero if s1 is equal to s2
s2=s1.toLowerCase;	Converts all the elements of string s1 to lowercase
s2=s1.toUpperCase;	Converts all the elements of string s1 to uppercase
s2=s1.replace('A','B');	Replace all appearances of element 'A' by 'B'
s1.concat(s2)	Concatenates s1 and s2

String Constructors

1) String (char [] chars):- Allocates a new string from the given character array

   e.g. char chars[]={'P','I','C','T'};

   String s =new String(chars);

2) String(char[] chars, int start_Index, int numChars):- Allocates the string from given character array but choose count characters from start index.

   e.g. char chars[]={'P','I','C','T'};

   String s=new String(chars, 1,3);

3) String(byte ascii[]):- Construct the new string by decoding the byte array

   e.g. byte ascii[]={65,66,67,68,69,70};

   String s=new String(ascii);

4) String(byte ascii[],int start_index,int length):- Construct the new string from the byte array depending on the start index and length

   e.g. . byte ascii[]={65,66,67,68,69,70};

   String s=new String(ascii,2,3);

String Array

- We can also create and use arrays that contains strings as an elements.
- Syntax

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

- Example:

String department [ ] = **new** String[5];

this statement will create, ***department*** of size 5 to hold five string constants.

Assignment no. 5

- Write Programs in Java to sort

  i) List of integers

  ii) List of names.

The objective of this assignment is to learn Arrays and Strings in Java

Assignment no. 6

- Write Programs in Java to add two matrices.

The objective of this assignment is to learn two dimensional Arrays in Java

String Buffer

- StringBuffer supports to modifiable string.
- As you know, String represents fixed length, immutable character sequence.
- In contrast, StringBuffer represents growable and writable character sequences.
- StringBuffer may have characters and substrings inserted in the middle or appended to the end

StringBuffer Constructors

1) StringBuffer():- It reserves room for 16 characters without reallocation

   e.g. StringBuffer s =new StringBuffer();

2) StringBuffer(int size):- Allocates Accepts an integer argument that explicitly sets the size of the buffer

   e.g. StringBuffer s =new StringBuffer(20);

3) StringBuffer(String str):- :- It accepts a string argument that sets the initial contents of the StringBuffer object and reserves room for 16 more characters without reallocation.

   e.g. StringBuffer s=new StringBuffer("PICT");

Methods	**Action Performed**
append()	Used to add text at the end of the existing text.
length()	The length of a StringBuffer can be found by the length( ) method
capacity()	the total allocated capacity can be found by the capacity( ) method
charAt()	specifies the index of the character being obtained
delete()	Deletes a sequence of characters from the invoking object
deleteCharAt()	Deletes the character at the index specified by *loc*
ensureCapacity()	Ensures capacity is at least equals to the given minimum.
insert()	Inserts text at the specified index position
length()	Returns length of the string
reverse()	Reverse the characters within a StringBuffer object
replace()	Replace one set of characters with another set inside a StringBuffer object

StringBuilder

- **StringBuilder** in Java represents a mutable sequence of characters.
- String Class in Java creates an immutable sequence of characters, the StringBuilder class provides an alternative to String Class, as it creates a mutable sequence of characters.
- The function of StringBuilder is very much similar to the StringBuffer class, as both of them provide an alternative to String Class by making a mutable sequence of characters.
- StringBuilder class differs from the StringBuffer class on the basis of synchronization. The StringBuilder class provides no guarantee of synchronization whereas the StringBuffer class does.
- StringBuilder more faster than StringBuffer

Wrapper Class

- Def:- Wrapper class are predefined class by JAVA which can contain the primitive data type.
- We can wrap a primitive values to wrapper class object.

Primitive type	Wrapper class
boolean	Boolean
byte	Byte
char	Character
float	Float
int	Integer
long	Long
short	Short
double	Double

Wrapper Class Method

- PARSE : It is a method which take a string(input) as an argument and convert in other formats as like :

  1.Integer

  2.Float

  3.Double

- THE TYPES OF PARSE METHODS :

  1.parseInt();

  2.parseDouble();

  3.parseFloat();

Syntax: ParseInt()

- **Integer.parseInt(String s);**
- **Float.parseFloat(String s);**
- **Double.parseDouble(String s);**

Example1:

```
String s=”156″; //156 is not a number it is string;
   System.out.println(s+1);  //output:1561 :”156″+1=1561 string concatenation.

int x=Integer.parseInt(“156”);

System.out.println(x+1); //output:157 :156+1=157
```

Autoboxing and Unboxing

- Autoboxing:- It is automatic conversion that JAVA makes between the primitive data types to corresponding object wrapper class.
- Unboxing:- when we convert object of wrapper class to respective primitive data types.

When to use

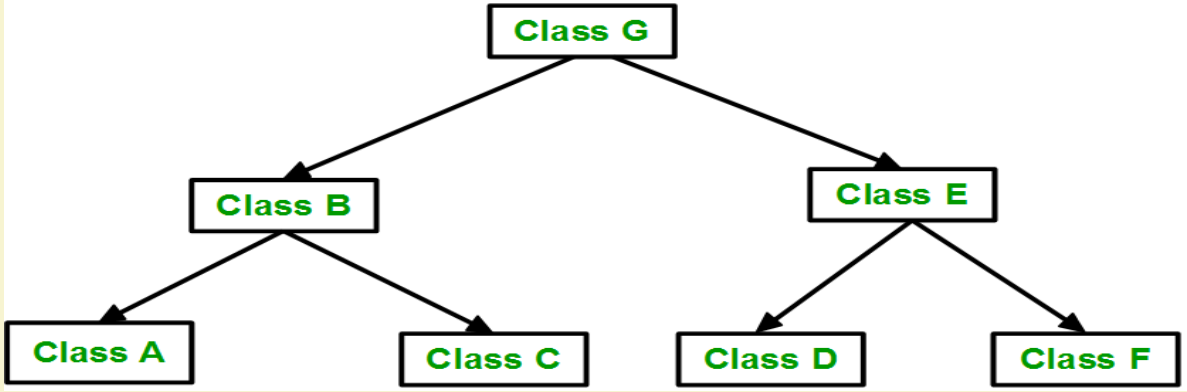
- While using collection framework
- To convert primitive data type to objects and vice versa
- Many libraries in java deals with objects not with primitive data type

Fundamental of JAVA Programming

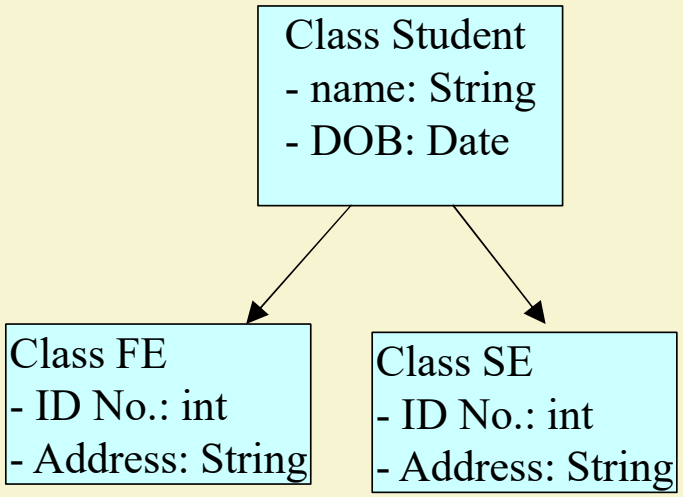
Inheritance in JAVA

Inheritance Concept

- Inheritance is a fundamental concept of Object Oriented Programming
- Mechanism of deriving a new class from an old one is called as Inheritance
- Java classes can be reused by creating new classes i.e. reusing the properties of existing ones



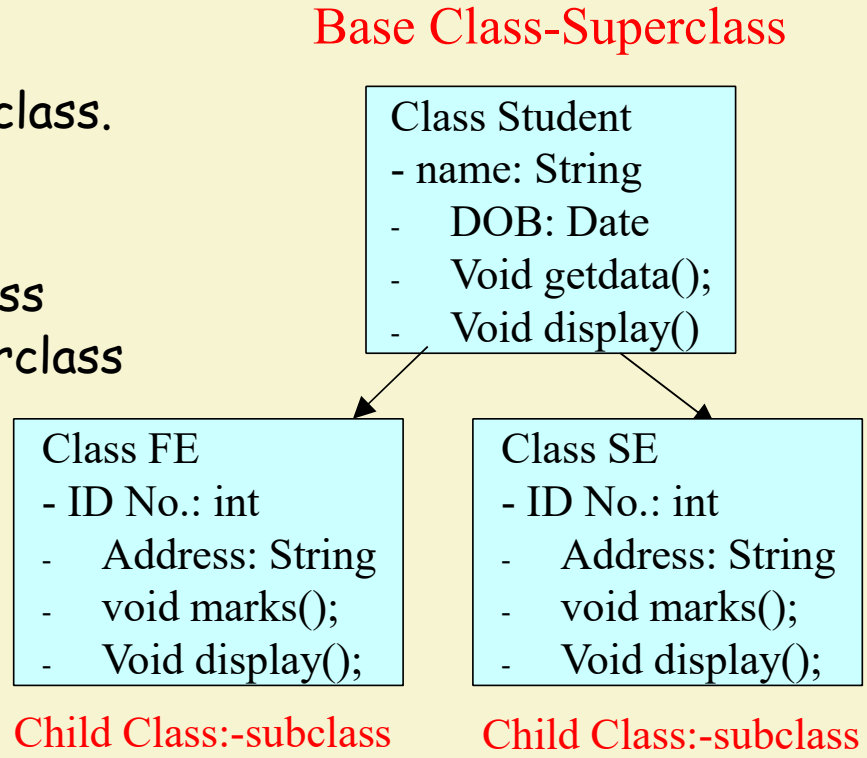
Base Class-Superclass



Child Class:-subclass Child Class:-subclass

Inheritance Concept

- A class can be defined as a "subclass" of another class.
  - The subclass inherits all data attributes of its superclass
  - The subclass inherits all methods of its superclass
  - The subclass inherits all associations of its superclass
- The subclass can:
  - Add new functionality
  - Use inherited functionality
  - Override inherited functionality



Inheritance Concept

Superclass

Person
- name: String
- dob: Date

is a kind of

Employee
- employeeID: int
- salary: int
- startDate: Date

Subclass

Superclass

Person
name = "John Smith"
dob = Jan 13, 1954

Subclass

Employee
name = "John Smith"
dob = Jan 13, 1954
employeeID = 37518
salary = 65000
startDate = Dec 15, 2000

Advantages of Inheritance

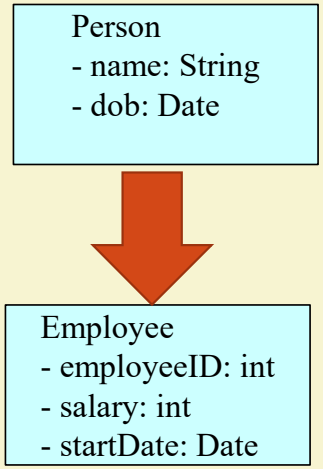
1. To use existing functionality
2. Override existing functionality
3. Provide new functionality
4. Combination of existing and new functionality

Inheritance Syntax (Defining a subclass)

- The Extends keyword is used to define a new class that derives from an existing class.
- The meaning of "extends" is to increase the functionality.

```
class <subclass_name> extends <superclass_name>
{
   //Data and methods in this sublass
}
```

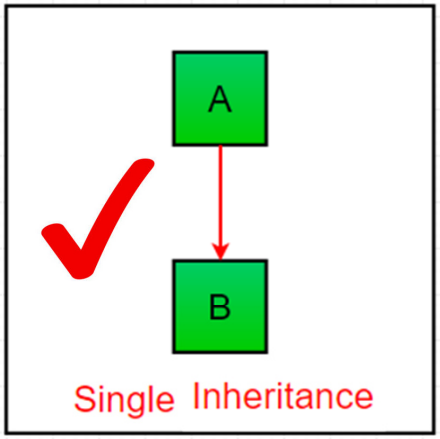
```
class Person
{
  private String name;
  private Date dob;
  [...]
```

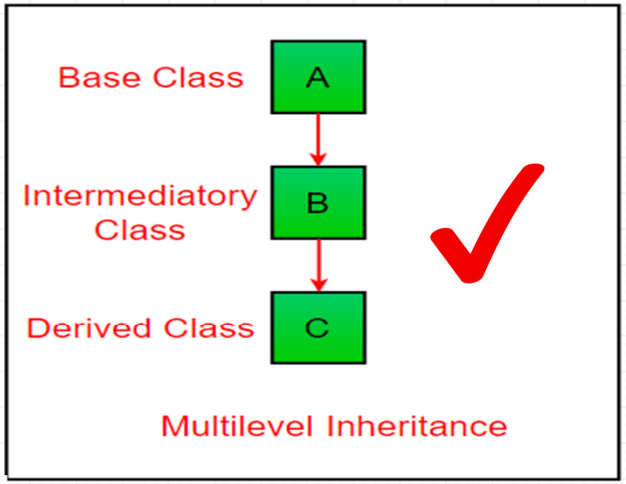


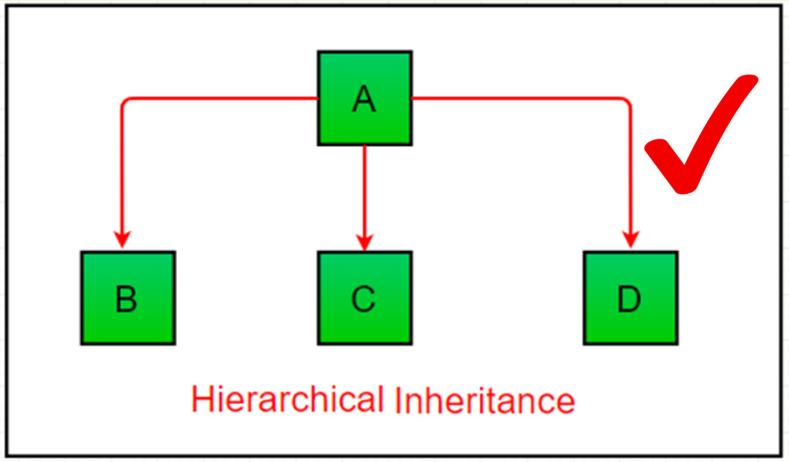

```
class Employee extends Person
{
  private int employeID;
  private int salary;
  private Date startDate;
  [...]
```

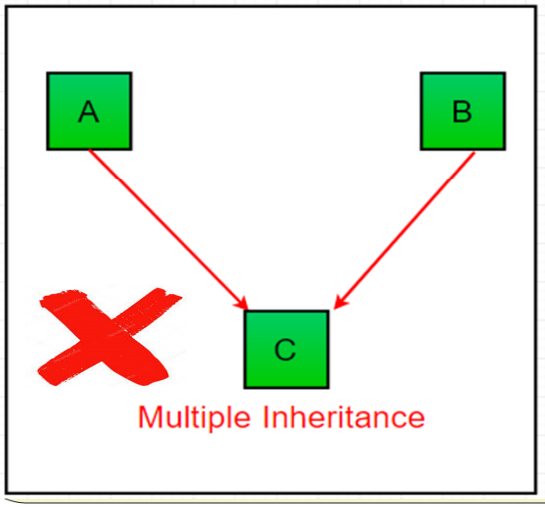
```
Employee anEmployee = new
  Employee();
```

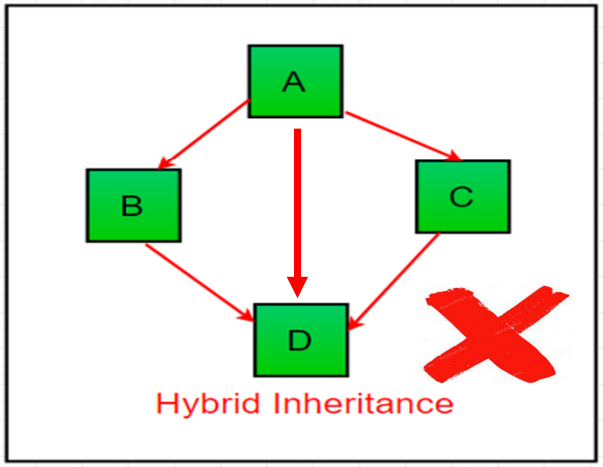
Types of Inheritance











Single Inheritance Example

Employee
- basic_salary
- basic_display()

Salary
- no_of_increment
- Salary();
- salary();

```
class Employee
{
    int basic_salary=50000;
    void basic_display()
    {
        System.out.println("Employee Salary:");
        System.out.println("Basic Salary:"+basic_salary);
    }
}
```

```
class Salary extends Employee
{
    int no_of_increment;
    int final_salary;
    Salary(int num)
    {
        no_of_increment=num;
    }
    void salary()
    {
        basic_display();
        final_salary=(no_of_increment*2500)+basic_salary;
        System.out.println("Final Salary:"+final_salary);
    }
}
```

```
public class Practice1 {

    public static void main(String[] args)
    {
        Salary e1=new Salary(2);
        e1.salary();
    }
}
```

```
Employee Salary:
Basic Salary:50000
Final Salary:55000
```

MultiLevel Inheritance Example

```java
import java.util.Scanner;
```

```java
class Employee1
{
    int basic_salary=50000;
    void basic_display()
    {
        System.out.println("Employee Salary:");
        System.out.println("Basic Salary:"+basic_salary);
    }
}
```

```java
class Salary1 extends Employee1
{
    int no_of_increment;
    int final_salary;
    Scanner obj=new Scanner(System.in);
    void salary()
    {
        basic_display();
        System.out.println("Enter the no of increment=");
        no_of_increment=obj.nextInt();
        final_salary=(no_of_increment*2500)+basic_salary;
        System.out.println("Final Salary:"+final_salary);
    }
}
```

```java
class Deposit extends Salary1
{
    double tax_value=0.10;
    int deposit_salary;
    void deposit()
    {
        salary();
        deposit_salary=(int) (final_salary-(final_salary*tax_value));
        System.out.println("Deposit_salary="+deposit_salary);
    }
}
```

```java
public class Pracrice2 {
    public static void main(String[] args)
    {
        Deposit e1=new Deposit();
        e1.deposit();
    }
}
```

```
Employee Salary:
Basic Salary:50000
Enter the no of increment
2
Final Salary:55000
deposit_salary=49500
```

Hierarchical Inheritance Example

Multilevel Inheritance Example

```
import java. util. Scanner;
Class A
{
  int x;
 Scanner in=new Scanner(System.in);
void get_data()
{
  System.out.println("Enter the value of x:-");
  x= in.nextInt();
  System.out.println("x="+x);
 }
}
```

```
Class B extends A
{
  int y;
void enter_data()
{
  System.out.println("Enter the value of y:-");
  y= in.nextInt();
  System.out.println("y="+y);
}
Void display()
{
  System.out.println("x+y="+(x+y));
}
}
```

```
Class C extends B
{
  int z;
void put_data()
{
  System.out.println("Enter the value of z:-");
  z= in.nextInt();
  System.out.println("z="+z);
 }
Void display1()
{
  System.out.println("x+y+z="+(x+y+z));
}
}
```

```
class Multilevel
{
   public static void main(String args[])
{
    C subobj1=new C();
     subobj1.get_data();
     subobj1.enter_data();
     subobj1.display();
      subobj1.put_data();
     subobj1.display1();
 }
}
```

Output:- Enter the value of x:- 2
x=2
Enter the value of y:-3
y=3
x+y=5
Enter the value of z:- 7
z=7
x+y+z=12

Method Overriding Concept

- Declaring a method in a subclass which is already present in the parent class is known as Method Overriding
- In this case the method in parent class is called as overridden method and method in the subclass is called as overriding method.
- **Usage of JAVA Method Overriding :-**

1. Method overriding is used to provide the specific implementation of a method which is already provided by its superclass
2. Method overriding is used for runtime polymorphism

- **Rules:-**

1. The method must have the same name as in the parent class.
2. The method must have the same parameter as in the parent class.

Method Overriding Example

```
import java. util.Scanner;
Class A
{
  int x;
 Scanner in=new Scanner(System.in);
void get_data()
{
  System.out.println("Enter the value of x:-");
  x= in.nextInt();
  System.out.println("x="+x);
 }
}
```

```
Class B extends A
{
  int y;
void enter_data()
{
  System.out.println("Enter the value of y:-");
  y= in.nextInt();
  System.out.println("y="+y);
 }
Void display()
{
  System.out.println("x+y="+(x+y));
}
}
```

```
Class C extends B
{
  int z;
void put_data()
{
  System.out.println("Enter the value of z:-");
  z= in.nextInt();
  System.out.println("z="+z);
 }
Void display()
{
  System.out.println("x+y+z="+(x+y+z));
}
}
```

```
class Multilevel
{
   public static void main(String args[])
{
    C subobj1=new C();
     subobj1.get_data();
     subobj1.enter_data();
     subobj1.display();
     subobj1.put_data();
     subobj1.display();
 }
}
```

Output:- Enter the value of x:- 2
x=2
Enter the value of y:-3
y=3
x+y=5
x+y+z=2+3+0=5
Enter the value of z:- 7
z=7
x+y+z=12

The Keyword Super

- Super keyword in Java is a reference variable
- super is used to access something (any protected or public field or method) from the super class that has been overridden

- super.<variable_name> refers to the variable of parent class.
- super() invokes the constructor of immediate parent class.
- super.<method_name> refers to the method of parent class.

Super: Referring Super Class Instance Variables

```
class Animal
{
   String color= “white”;
}
class Dog extends Animal
{
   String color =“black”;
void printcolor()
{
   System.out.println(color);
   System.out.println(super.color);
  }
}
class TestSuper1
{
  public static void main(String args[])
{
  Dog d =new Dog();
  d.printcolor();
  }
}
```

Output :- black
white

Super: Invoking Super Class Method

```
class Animal
{
   String color= “white”;
 void eat ()
{ System.out.println(“eating……”);
   }
}
class Dog extends Animal
{
   String color =“black”;
void printcolor()
{
   System.out.println(color);
   System.out.println(super.color);
 }
void eat()
{
   super.eat();
    System.out.println(“eating bread….”);
 }
}
```

```
class TestSuper1
{
 public static void main(String args[])
{
 Dog d =new Dog();
 d.printcolor();
 d.eat();
 }
}
```

```
black
white
eating……..
eating bread……
```

Super: Invoking Parent Class Constructor

```
class Animal
{
 Animal ()
{ System.out.println("Animal color is white and
eating……");
   }
}
```

```
class Dog extends Animal
{
   Dog()
{
   super();
   System.out.println("bread");
  }
}
```

```
class TestSuper1
{
  public static void main(String args[])
{
  Dog d =new Dog();
}
}
```

Animal colour is white
and eating…….
bread

Super();

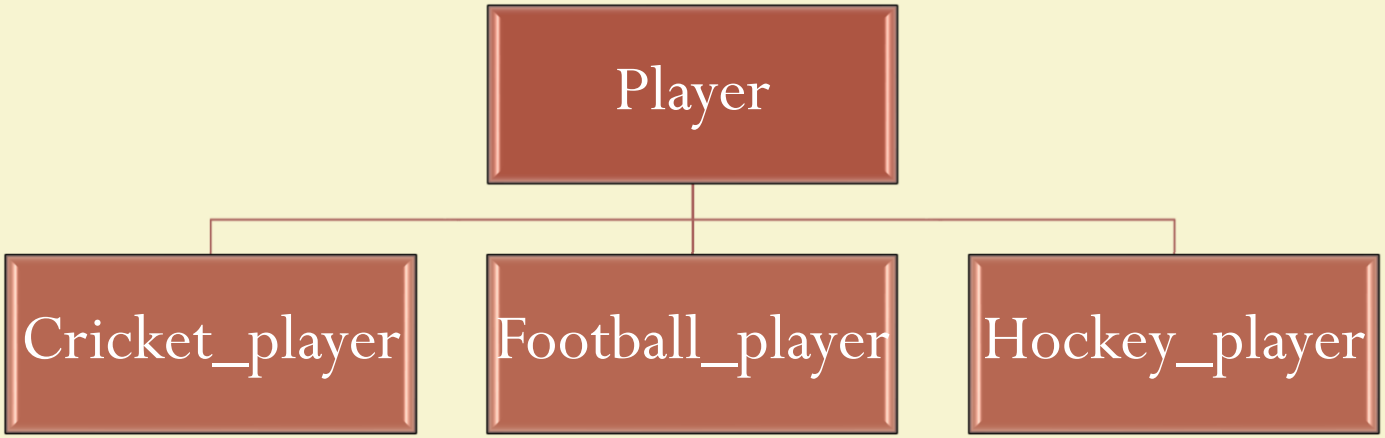
- A subclass constructor is used to construct the *instance variables* of both the subclass and the superclass.
- Purpose of super():

  Subclass constructor uses the keyword *super* to invoke the constructor of superclass.

- Conditions inorder to use *super* keyword inside subclass constructor:
  1. *super()* may only be used within a subclass constructor method
  2. The call to super class constructor must appear as the first statement within the subclass constructor
  3. The parameters in the *super()* call must match the order and type of the instance variable declared in the super class

Assignment No.7

- Write a program in Java to create a player class. Inherit the classes Cricket_player, Football_player and Hockey_player from player class. The objective of this assignment is to learn the concepts of inheritance in Java



```
import java.util.Scanner;
 class Player
  {
            String name;
            int age;
            String gameName;
            int noOfGamesPlayed;
            String address;
            String type;

Scanner in=new Scanner(System.in);

void getDetails()
     {
System.out.println("Enter the details Name ,  Age,
Address, Name of Game ,No of Games Played and
Type ");
       name=in.next();
      age=in.nextInt();
      address=in.next();
      gameName=in.next();
            noOfGamesPlayed=in.nextInt();
            type=in.next();
   }


void display()
    {
            System.out.println("Name : "+ name
     +"Age: " + age
     +" Game Name:"+ gameName
     + " Total Matches: " +noOfGamesPlayed
   +" Address: "+ address
+" International or National : " +type );
          }
 }
```

Super Class

```
class Cricket_Player extends Player
   {

            int totalRuns;
            int totalWickets;

            void getDetails()
     {
       super.getDetails();
  System.out.println("Enter the Total Runs and
Wickets: ");

   totalRuns=in.nextInt();
            totalWickets=in.nextInt();
            }

void display()
     {
                        super.display();
 System.out.println("Total Runs: " +totalRuns +"
Total
        Wickets :"+totalWickets);
            }

}
```

Cricket Sub Class

```
class FootBall_Player extends Player
  {
            int noOfGoals;

   void getDetails()
    {
              super.getDetails();
  System.out.println("Enter the total no of Goals");
                    noOfGoals=in.nextInt();
          }

void display()
      {
                  super.display();
   System.out.println("Total Goals: " +noOfGoals );
              }
    }
```

Football Sub Class

```
public class TenthAssignment
{

        public static void main(String[] args)
    {

Cricket_Player cp=new Cricket_Player();
                    cp.getDetails();
                    cp.display();

FootBall_Player fp=new FootBall_Player();
                    fp.getDetails();
                    fp.display();

        }

}
```

Main Method

Dynamic Method Dispatch Concept

- Dynamic method dispatch is a process in which a call to an overridden method is resolved at running time rather than compile time.
- It is also called as runtime polymorphism .
- Method overriding is resolved at runtime instead of compile time
- That means choice of the version of the overridden method to be executed in response to method call is done at runtime.
- In this process, an overridden method is called through the reference variable of superclass. But determination of the method to be called is based on the object being referred to by the reference variable.

```
class Super
 {
 public void method()
  {
    System.out.println("Super Method");
  }
}

class Sub extends Super
 {
  public void method()
 {
    System.out.println("Sub Method");
  }
}

class dyn_dis
{
public static void main(String args[])
 {
   Super A =new Super();
   A. method();
   Sub B=new Sub();
   B. method();
   Super C= new Sub();
   //Sub's object reference assigned Super type reference
   variable
        C.method();
   //Sub's Version of method will be called at runtime
}
 }
```

Dynamic Method Dispatch Example

Output:-Super Method
Sub Method
Sub Method

Final Keyword Concept

Final keyword in Java is used to restrict the access of an item from its super class to subclass

1. Using *final* to prevent overriding:-

   methods declared as final cannot be overridden

   variable cannot be access in subclass

2. Using *final* to prevent inheritance:-

   class declared as final cannot be inherited

Hey, I am final !
You can not change my value
You can not override me
You can not inherit me

Abstraction in Java

- **Abstraction in Java**
  - **Abstraction** is a process of hiding the implementation details and showing only functionality to the user.
  - It shows only important things to the user and hides the internal details
  - Ex: sending SMS.
  - **Abstraction lets you focus on what the object does instead of how it does it.**

Abstract Class in Java

- A class that is declared as abstract is known as **abstract class**
- It can have abstract and non-abstract methods (method with the body).
- It needs to be extended and its method implemented. It cannot be instantiated.
- **Ex:**

```
abstract class A
{
}
```

- **abstract method:** A method that is declared as abstract and does not have implementation is known as abstract method.(i.e. method with the body only without it's definition)
- **Ex: abstract** void print_Status();

Rules about Abstract Classes

- An abstract class must be declared with an abstract keyword
- It can have abstract & non abstract method
- It can not be instantiated (i.e object of that class can not be created)
- It can have constructors & static methods also.
- It can have final method which will force the subclass not to change the body of the method.

Abstract Method

- A method which is declared as abstract and does not have implementation is known as an abstract method.
- E.g. **abstract void** printStatus();//no method body and abstract

Example of Abstract Class that has Abstract Method

```
abstract class Bike
{
  abstract void run();
}

  class Honda extends Bike
{
  void run()
    {
     System.out.println("running safely..");
    }
}
```

```
public class Test {
   public static void main(String args[])
    {
       Bike obj = new Bike();
       Bike obj = new Honda();
       obj.run();
    }
   }
```

Output:- running safely

Example of Abstract Class that has Constructor, Abstract Method

```
abstract class Bike
{
  Bike()
  {
    S.o.p("Bike Starts");
  }
  abstract void run();
  void changeGear()
  {
    S.o.p("Gear Changed");
  }
}
class Honda extends Bike
{
 void run()
  {
    System.out.println("running safely..");
  }
}
public class Test {
  public static void main(String args[])
  {
    Bike obj = new Honda();
    obj.run();
    obj.changeGear();
  }
}
```

Output:-Bike Starts
Running Safely
Gear Changed

Packages

- Java two most innovative features are packages and Interfaces
- Package is the set of collection of classes and / or interfaces
- It act as a container for classes that are used.

Why Packages ?

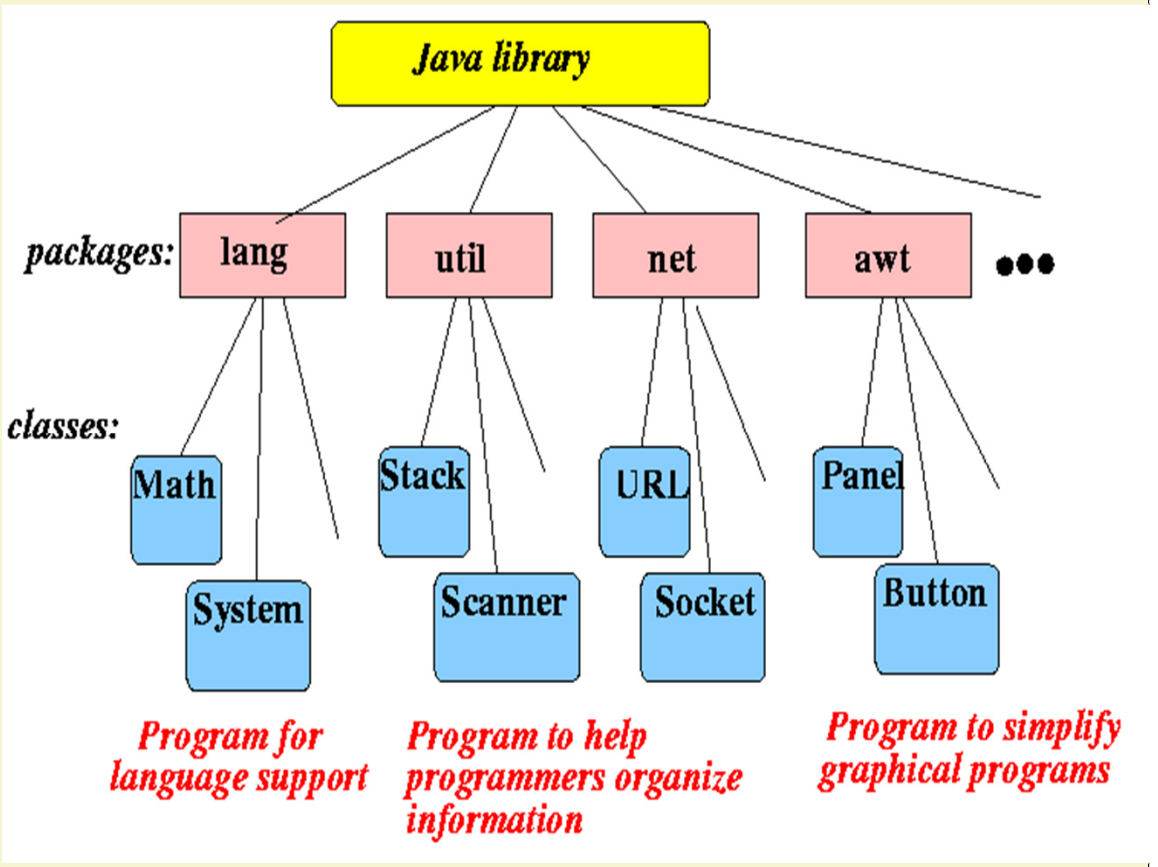
- Limitation to reusing the classes within a program
- If we need to use classes from other programs without physically copying them into the program under development?

Advantages of Packages

1. Code Reusability :- Because package contain the group of classes (i.e Reusability and Maintainability)
2. Resolving Name Collision :- when multiple packages have classes with the same name
3. Package also provides the hiding of class facility (i.e other programs can not use the class from hidden packages)
4. Access Limitation can be applied with the help of package
5. Nesting of package

Packages

- In Java, already many predefined packages are available those are to help programmers to develop their software in any easy ways.
- E.g. java.swing ,java.lang-
- All build in package are bundle called API bundled with the JDK



Simple example of java package

```
package mypack;
 public class Simple
  {
    public static void main(String args[])
      {
        System.out.println("Welcome to package");
      }
  }
```

How to access package from another package?

- There are three ways to access the package from outside the package.

1. import package.*;

2.import package.classname;

3.fully qualified name.

Usingpackagename.*

File Name:-A.java

```
package pack;
 public class A
{
   public void msg()
   {
    System.out.println("Hello");
   }
  }
```

File Name:-B.java

```
package mypack;
import pack.*;
   class B
   {
      public static void main(String args[])
   {
       A obj = new A();
      obj.msg();
   }
   }
```

Using packagename.classname

File Name:-A.java

```
package pack;
 public class A
{
   public void msg()
   {
    System.out.println("Hello");
   }
  }
```

File Name:-B.java

```
package mypack;
import pack.A;
   class B
   {
      public static void main(String args[])
    {
       A obj = new A();
      obj.msg();
    }
   }
```

Using fully qualified name

File Name:-A.java

```
package pack;
 public class A
{
   public void msg()
   {
    System.out.println("Hello");
   }
  }
```

File Name:-B.java

```
package mypack;
import pack.A;
    class A
    {
       public static void main(String args[])
      {
         A obj = new A();

          pack.A obj = new pack.A();
             obj.msg();
     }
    }
```

Access Modifiers in java

- There are 4 types of java access modifiers:

1.private

2.default

3.protected

4.public

Access Modifier

1) private access modifier

The private access modifier is accessible only within class.

2) default access modifier

If you don't use any modifier, it is treated as **default** by default. The default modifier is accessible only within package.

3) protected access modifier

The **protected access modifier** is accessible within package and outside the package but through inheritance only.

4) public access modifier

The **public access modifier** is accessible everywhere. It has the widest scope among all other modifiers

Access Protection

Access Modifier / Access Location	Public	Protected	Default	Private
Same Class	Yes	Yes	Yes	Yes
Subclass in Same Package	Yes	Yes	Yes	No
Non Subclasses in same package	Yes	Yes	Yes	No
Subclass in Different Packages	Yes	Yes	No	No
Non Subclasses in Different packages	Yes	No	No	No

Multiple Inheritance

- Java does not support multiple inheritance
- Classes in Java can not have more than one superclass

E.g.
```
class A extends B extends C
    {
        ----------------
        ----------------
    }
```

- Is not permitted in Java
- Large number of real life applications require the use of multiple inheritance
- In C++ , implementation of multiple inheritance proves difficult and adds complexity to the language.
- But Java provides an alternate approach known as interfaces to support the concept of multiple inheritance

Interface

- Basically it is a kind of class
- Like classes, interface contain methods and variables but with a major difference
- Difference is that interfaces define only abstract methods and final fields
- This means that interfaces do not specify any code to implement these methods and data fields contain only constants
- Responsibility of the class that implements an interface to define the code for implementation of these methods.
- Syntax for defining an interface is very similar to that for defining a class

Syntax for Interface

- General form of an interface definition is

```
interface InterfaceName
{
    variable declaration;
    method declaration;
}
```

Interface is just keyword

- Variables are declared as

```
static final type variablename= value;
```

- Methods declaration will contain only a list of methods without any body statements

```
return type methodsname1 (parameter_list);
```

```
Intreface Area
{
    final static float pi=3.142F;
    float comput ( float x, float y);
    void show();
}
```

Difference Between Class and Interface

Class	Interface
1. The members of a class can be constant or variables	1. The members of an interface are always declared as constant, i.e., their values are final
2. The class definition can contain the code for each of its methods. That is, methods can be abstracted or non-abstracted in class.	2. The methods in an interface are abstracted in nature, i.e., there is no code associated with them. It is later defined by the class that implements the interface.
3. It can be used to declare objects	3. It cannot be used to declare objects. It can only be inherited by class.
4. Use various access specifies like public, private or protected.	4. Supports only public access specifier.

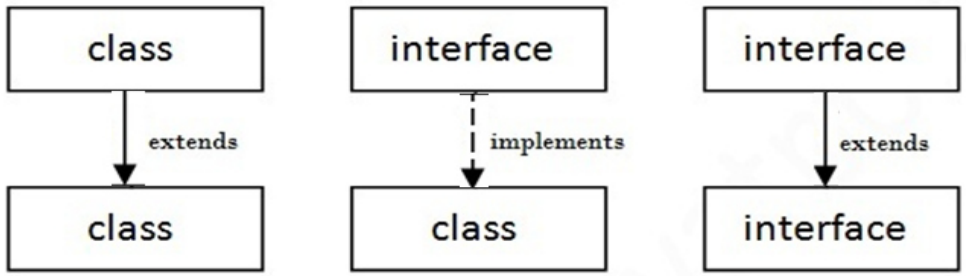
Extending Interface

- As like classes, we can apply reusability concept on existing interface to create new interface.
- That is, an interface can be sub-interfaced from existing interface.
- So this existing interface is called superinterface.
- The new subinterface will inherit all the characteristics of superinterface in the manner similar to subclass.
- Syntax for this is:

```
interface name2  extends name1
{
        body of name2;
}
```

Extending Interface

```
interface Itemconstants
{
   int code=1001;
   string name="Fan";
}
interface Item extends ItemConstants
{
   void display();
}
```




```
interface Itemconstants
{
   int code=1001;
   string name="Fan";
}
interface ItemMethods extends ItemConstants
{
   void display();
}
Interfaces Item extends ItemConstants,ItemMethods
{
      ---------------
      ----------------
}
```

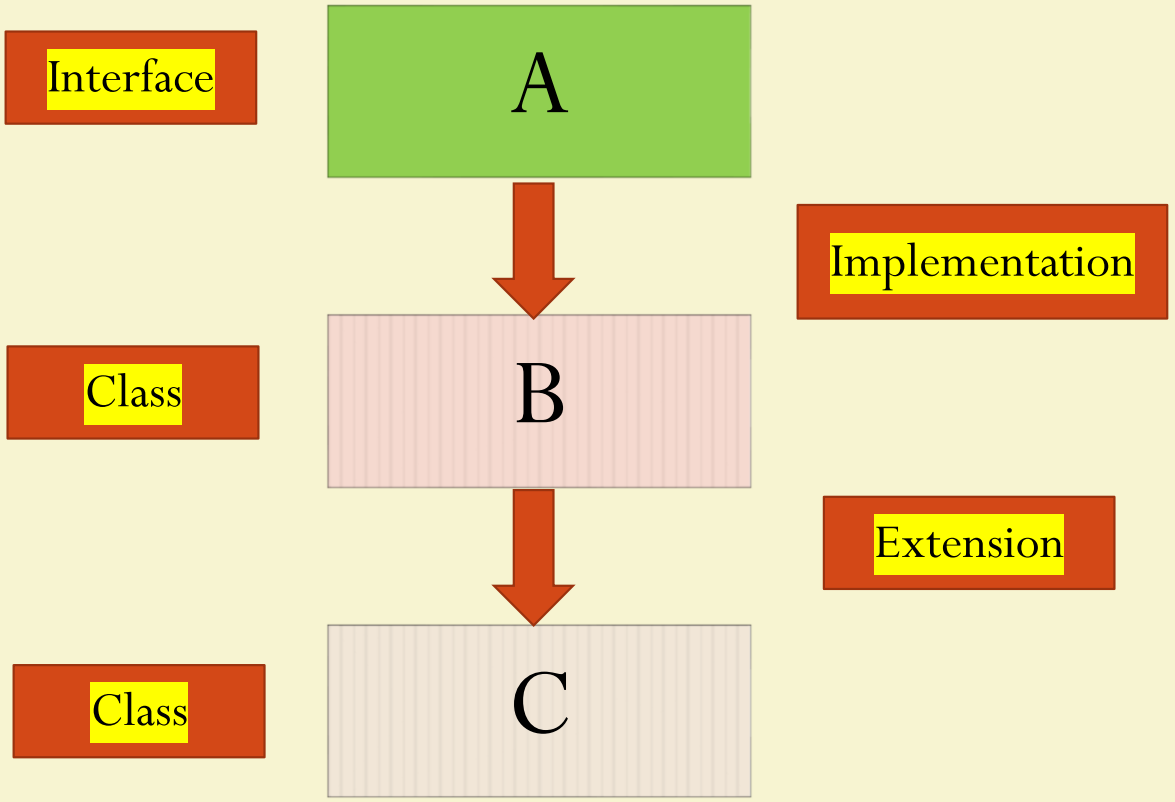
Implementing Interfaces

- Interfaces are used as "Superclasses" whose properties are inherited by classes.
- Therefore necessary to create a class that inherits the given interface

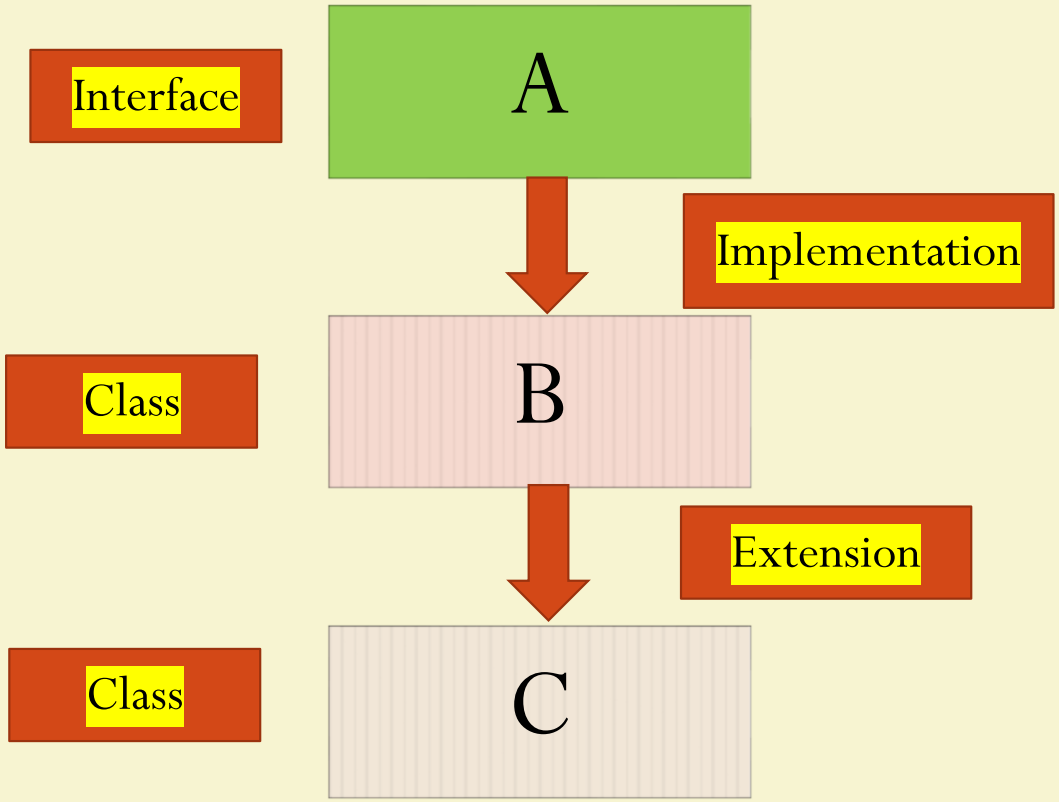
```
Class  classname implements interfacename
{
      Body of classname
}
```

```
Class  classname extends superclass
          implements interfacename1,interfacename2,……..
{
      Body of classname
}
```

Various forms of Interface Implementation



Various forms of Interface Implementation



Default Method in Interface

```
interface Drawable
{
  void draw();
default void msg()
{
s.o.p("default method");
} }
```

```
class Rectangle implements Drawable
{
public void draw()
{s.o.p("drawing rectangle");
} }
```

```
class TestInterfaceDefault{
p.s.v.m.(String args[])
  {
  Drawable d=new Rectangle();
d.draw();
 d.msg();
 }
}
```

Static Method in Interface

```
interface Drawable
{
 void draw();
 static int cube(int x)
{return x*x*x;}
 }
```

```
 class Rectangle implements Drawable
{
public void draw()
{
s.o.p("drawing rectangle");
}
 }
```

```
class TestInterfaceStatic
{
 psvm(String args[])
{
  Drawable d=new Rectangle();
   d.draw();
s.o.p( Drawable.cube(3) );
}
}
```

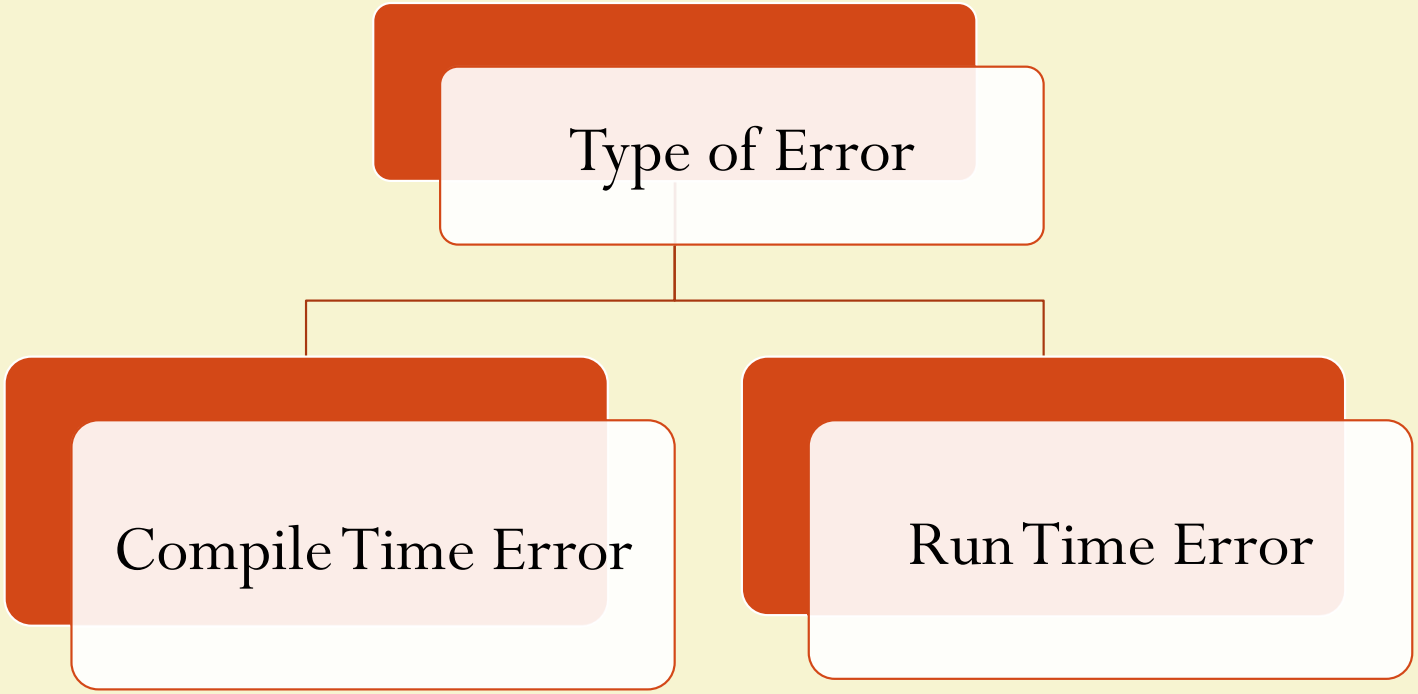
ABSTRACT CLASS	INTERFACE
1) Abstract class can have abstract and non-abstract methods.	Interface can have only abstract methods. Since Java 8, it can have default and static methods also.
2) Abstract class doesn't support multiple inheritance.	Interface supports multiple inheritance.
3) Abstract class can have final, non-final, static and non-static variables.	Interface has only static and final variables.
4) Abstract class can provide the implementation of interface.	Interface can't provide the implementation of abstract class.
5) The abstract keyword is used to declare abstract class.	The interface keyword is used to declare interface.
6) Example: public abstract class Shape{ public abstract void draw();}	Example: public interface Drawable{ void draw();}

Object Oriented Programming

UNIT :- VI

Exception Handling , Multithreading, File I/o, Applet

Types of Errors



Compile Time Errors

- All syntax errors will be detected and displayed by the Java compiler ,these errors are called as compile time errors.
- Whenever the compiler displays an error, it will not create the .class file
- It is necessary to fix all these errors before we can successfully compile and run the program

```
class Myclass
{
   public static void main (String args [])
  {
     System.out.println(“Hello Java”)
   }
}
```

Syntax error, insert ";" to complete BlockStatements

Reasons for Compile Time Errors

- Missing Semicolons
- Missing brackets in classes and methods
- Misspelling of identifiers and keywords
- Missing double quotes in string
- Used of undeclared variables
- Bad Reference to objects
- Use of = in place of == operator

Run Time Errors

- Sometime. A program may compile successfully creating the .class file but may not run properly.
- Such programs may produce wrong results due to wrong logic or may terminate due to errors.
- Most Run time Errors are

1)Dividing an Integer by Zero

2) Out of the bounds of an array

3)Store a value into an array of an incompatible type

4) Passing a parameter that is not in a valid range or value for method

Attempting negative size of an array

Converting invalid string to a number

Accessing a character that is out of bounds of a string

Run Time Error Example

```
class Myclass
{
   public static void main (String args [])
  {
     int a=10;
     int b=5;
     int c=5;
     int x=a/(b-c);  //Division by zero
      System.out.println(" x="+x);
      int y=a/(b+c);
     System.out.println("y="+y);
   }
}
```

```
Java.lang.ArithmeticException: /by zero
At Myclass.main(Myclass.java:)
```

Exceptions

- Exception is a condition that is caused by a run time error in the program.
- When the Java interpreter encounters an error such as dividing an integer by zero, it creates an exception object and throws
- If the exception object is not caught and handled properly, the interpreter will display an error message as shown in the output of program and will terminate the program.
- If we want the program to continue with the execution of the remaining code, then we should try to catch the exception object thrown by the error condition and then display an appropriate message for taking corrective actions. This task is known as exception handling.
- Find the problem (Hit the exception)
- Inform that an error has occurred (throw the exception)
- Receive the error information (catch the exception)
- Take corrective actions (handle the exception)

Exception Type	Cause of Exception
ArithmaticException	Caused by math errors such as division by zero
ArrayIndexOutOfBoundException	Caused by bad array indexes
ArrayStoreException	Caused when a program tries to store the wrong type of data in array
FileNotFoundException	Caused by an attempt to access a nonexistent file
IOException	Caused by general I/O failures, such as inability to read from a file

Fundamental of JAVA Programming

UNIT :- IV

Interface and Packages in JAVA

Packages

- Java two most innovative features are packages and Interfaces
- Package is the set of collection of classes and / or interfaces
- It act as a container for classes that are used.

Why Packages ?

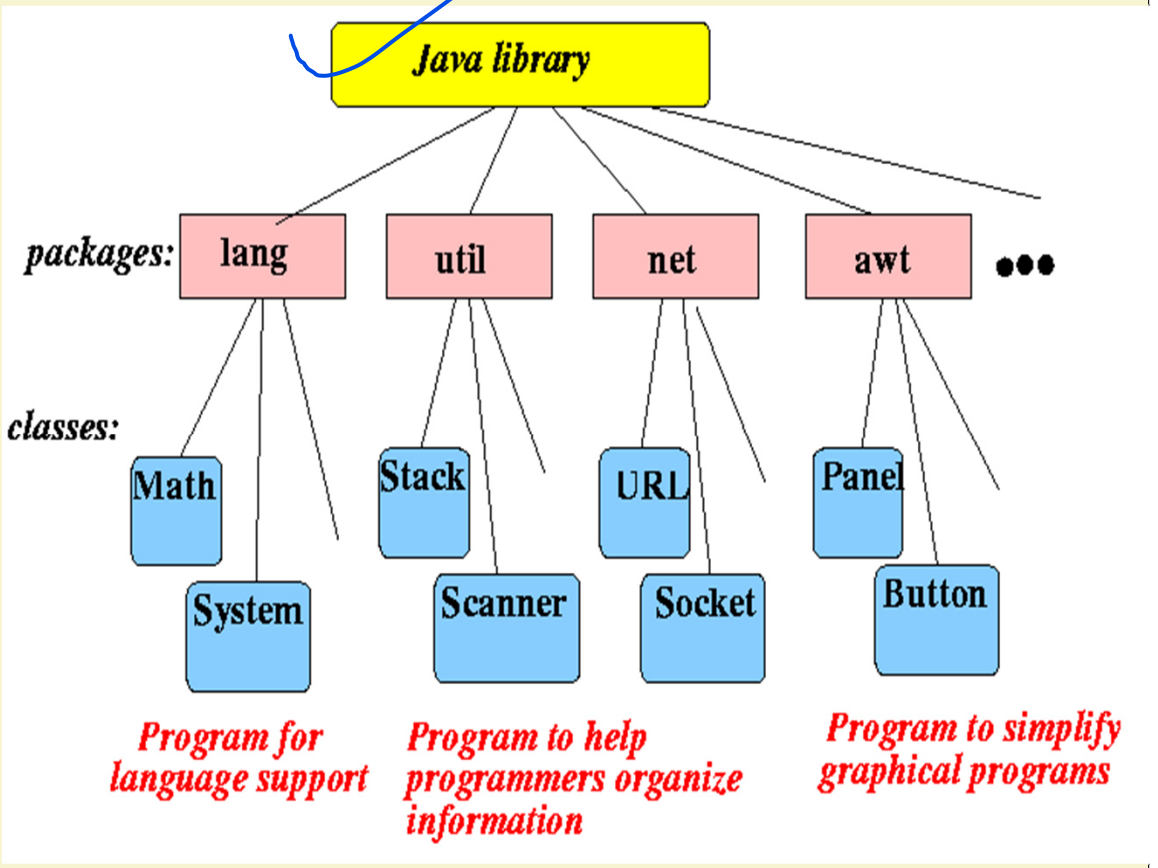
- Limitation to reusing the classes within a program
- If we need to use classes from other programs without physically copying them into the program under development?

Advantages of Packages

1. Code Reusability :- Because package contain the group of classes (i.e Reusability and Maintainability)
2. Resolving Name Collision :- when multiple packages have classes with the same name
3. Package also provides the hiding of class facility (i.e other programs can not use the class from hidden packages)
4. Access Limitation can be applied with the help of package
5. Nesting of package

Packages

- In Java, already many predefined packages are available those are to help programmers to develop their software in any easy ways.
- E.g. java.swing ,java.lang-
- All build in package are bundle called API bundled with the JDK



Simple example of java package

```
package mypack;
 public class Simple
  {
    public static void main(String args[])
      {
        System.out.println("Welcome to package");
      }
  }
```

How to access package from another package?

- There are three ways to access the package from outside the package.

1. import package.*;

2.import package.classname;

3.fully qualified name.

Usingpackagename.*

File Name:-A.java

```
package pack;
 public class A
{
   public void msg()
   {
    System.out.println("Hello");
   }
  }
```

File Name:-B.java

```
package mypack;
import pack.*;
   class B
   {
      public static void main(String args[])
   {
       A obj = new A();
      obj.msg();
   }
   }
```

Using packagename.classname

File Name:-A.java

```
package pack;
 public class A
{
   public void msg()
   {
    System.out.println("Hello");
   }
  }
```

File Name:-B.java

```
package mypack;
import pack.A;
   class B
    {
      public static void main(String args[])
     {
       A obj = new A();
      obj.msg();
     }
    }
```

Using fully qualified name

File Name:-A.java

```
package pack;
 public class A
{
   public void msg()
   {
    System.out.println("Hello");
   }
  }
```

File Name:-B.java

```
package mypack;
   class A
    {
      public static void main(String args[])
     {
          pack.A obj = new pack.A();
          obj.msg();
obj1=new A ();
          o A bj1.display();
     }
  void display()
{
}
```

Access Modifiers in java

- There are 4 types of java access modifiers:

1.private

2.default

3.protected

4.public

Access Modifier

1) private access modifier

The private access modifier is accessible only within class.

2) default access modifier

If you don't use any modifier, it is treated as **default** by default. The default modifier is accessible only within package.

3) protected access modifier

The **protected access modifier** is accessible within package and outside the package but through inheritance only.

4) public access modifier

The **public access modifier** is accessible everywhere. It has the widest scope among all other modifiers

Access Protection

Access Modifier / Access Location	Public	Protected	Default	Private
Same Class	Yes	Yes	Yes	Yes
Subclass in Same Package	Yes	Yes	Yes	No
Non Subclasses in same package	Yes	Yes	Yes	No
Subclass in Different Packages	Yes	Yes	No	No
Non Subclasses in Different packages	Yes	No	No	No

Multiple Inheritance

- Java does not support multiple inheritance
- Classes in Java can not have more than one superclass

E.g.
```
class A extends B extends C
    {
        ----------------
        ----------------
    }
```

- Is not permitted in Java
- Large number of real life applications require the use of multiple inheritance
- In C++ , implementation of multiple inheritance proves difficult and adds complexity to the language.
- But Java provides an alternate approach known as interfaces to support the concept of multiple inheritance

Interface

- Basically it is a kind of class
- Like classes, interface contain methods and variables but with a major difference
- Difference is that interfaces define only abstract methods and final fields
- This means that interfaces do not specify any code to implement these methods and data fields contain only constants
- Responsibility of the class that implements an interface to define the code for implementation of these methods.
- Syntax for defining an interface is very similar to that for defining a class

Syntax for Interface

- General form of an interface definition is

```
interface InterfaceName
{
    variable declaration;
    method declaration;
}
```

Interface is just keyword

- Variables are declared as

```
static final type variablename= value;
```

- Methods declaration will contain only a list of methods without any body statements

```
return type methodsname1 (parameter_list);
```

```
Intreface Area
{
    final static float pi=3.142F;
    float comput ( float x, float y);
    void show();
}
```

Difference Between Class and Interface

Class	Interface
1. The members of a class can be constant or variables	1. The members of an interface are always declared as constant, i.e., their values are final
2. The class definition can contain the code for each of its methods. That is, methods can be abstracted or non-abstracted in class.	2. The methods in an interface are abstracted in nature, i.e., there is no code associated with them. It is later defined by the class that implements the interface.
3. It can be used to declare objects	3. It cannot be used to declare objects. It can only be inherited by class.
4. Use various access specifies like public, private or protected.	4. Supports only public access specifier.

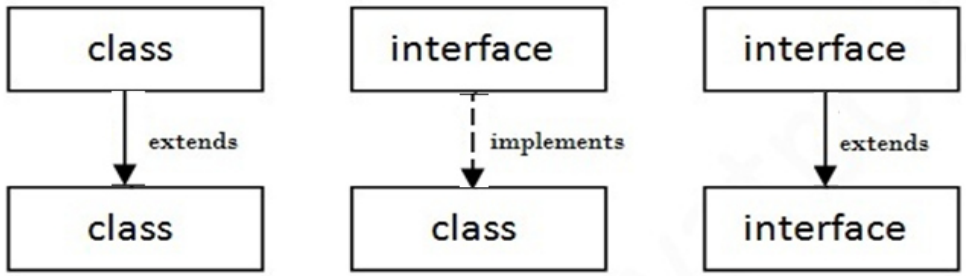
Extending Interface

- As like classes, we can apply reusability concept on existing interface to create new interface.
- That is, an interface can be sub-interfaced from existing interface.
- So this existing interface is called superinterface.
- The new subinterface will inherit all the characteristics of superinterface in the manner similar to subclass.
- Syntax for this is:

```
interface name2  extends name1
{
        body of name2;
}
```

Extending Interface

```
interface Itemconstants
{
   int code=1001;
   string name="Fan";
}
interface Item extends ItemConstants
{
   void display();
}
```




```
interface Itemconstants
{
   int code=1001;
   string name="Fan";
}
interface ItemMethods extends ItemConstants
{
   void display();
}
Interfaces Item extends ItemConstants,ItemMethods
{
      ---------------
      ----------------
}
```

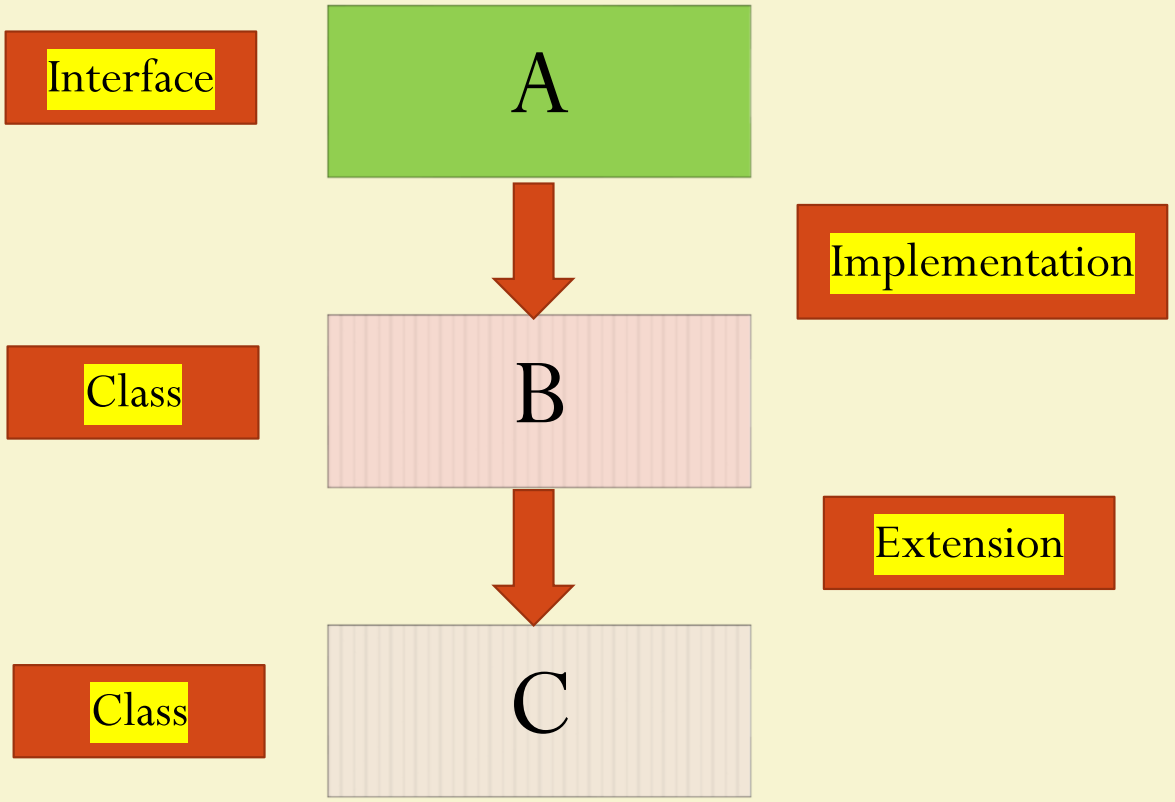
Implementing Interfaces

- Interfaces are used as "Superclasses" whose properties are inherited by classes.
- Therefore necessary to create a class that inherits the given interface

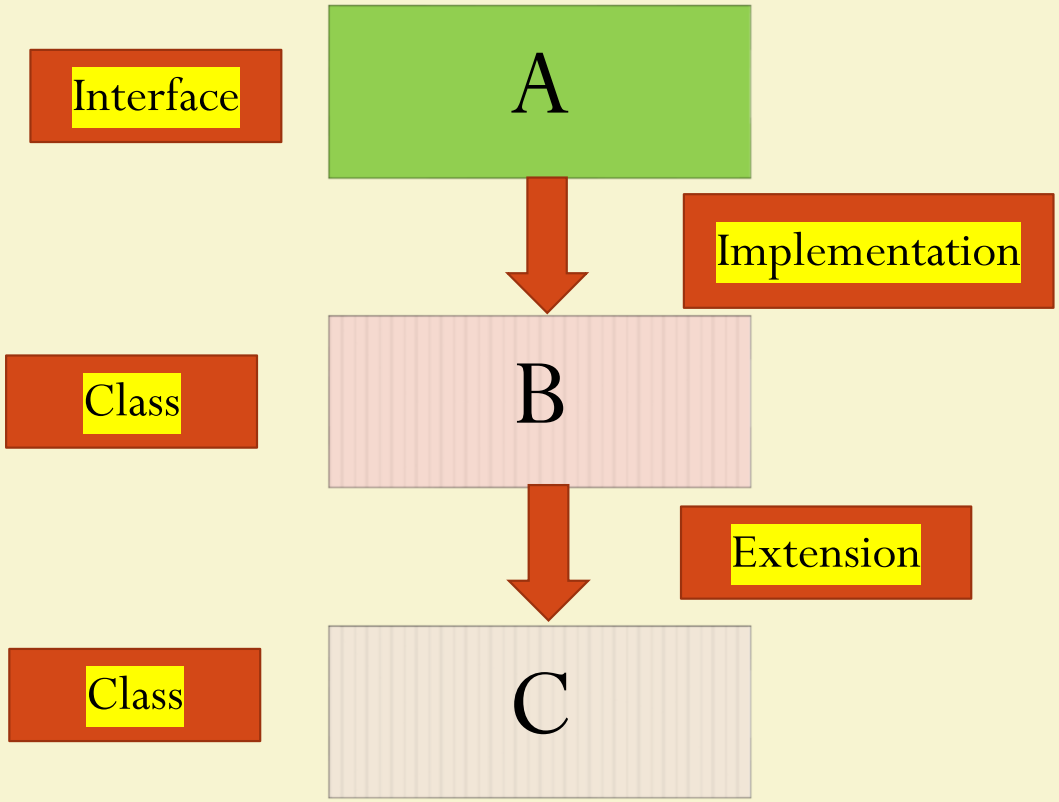
```
Class  classname implements interfacename
{
      Body of classname
}
```

```
Class  classname extends superclass
          implements interfacename1,interfacename2,……..
{
      Body of classname
}
```

Various forms of Interface Implementation



Various forms of Interface Implementation



Important Points

Default Method in Interface

```
interface Drawable
{
  void draw();
default void msg()
{
s.o.p("default method");
} }
```

```
class Rectangle implements Drawable
{
public void draw()
{s.o.p("drawing rectangle");
} }
```

```
class TestInterfaceDefault{
p.s.v.m.(String args[])
  {
  Drawable d=new Rectangle();
d.draw();
 d.msg();
 }
}
```

Static Method in Interface

```
interface Drawable
{
 void draw();
 static int cube(int x)
{return x*x*x;}
 }
```

```
 class Rectangle implements Drawable
{
public void draw()
{
s.o.p("drawing rectangle");
}
 }
```

```
class TestInterfaceStatic
{
psvm(String args[])
{
 Drawable d=new Rectangle();
  d.draw();
  s.o.p( Drawable.cube(3)
  );
}
}
```

Sr. No.	Key	Static Interface Method	Default Method
1	Basic	It is a static method which belongs to the interface only. We can write implementation of this method in interface itself	It is a method with default keyword and class can override this method
2	Method Invocation	Static method can invoke only on interface class not on class.	It can be invoked on interface as well as class
3	Method Name	Interface and implementing class , both can have static method with the same name without overriding each other.	We can override the default method in implementing class
4.	Use Case	It can be used as a utility method	It can be used to provide common functionality in all implementing classes

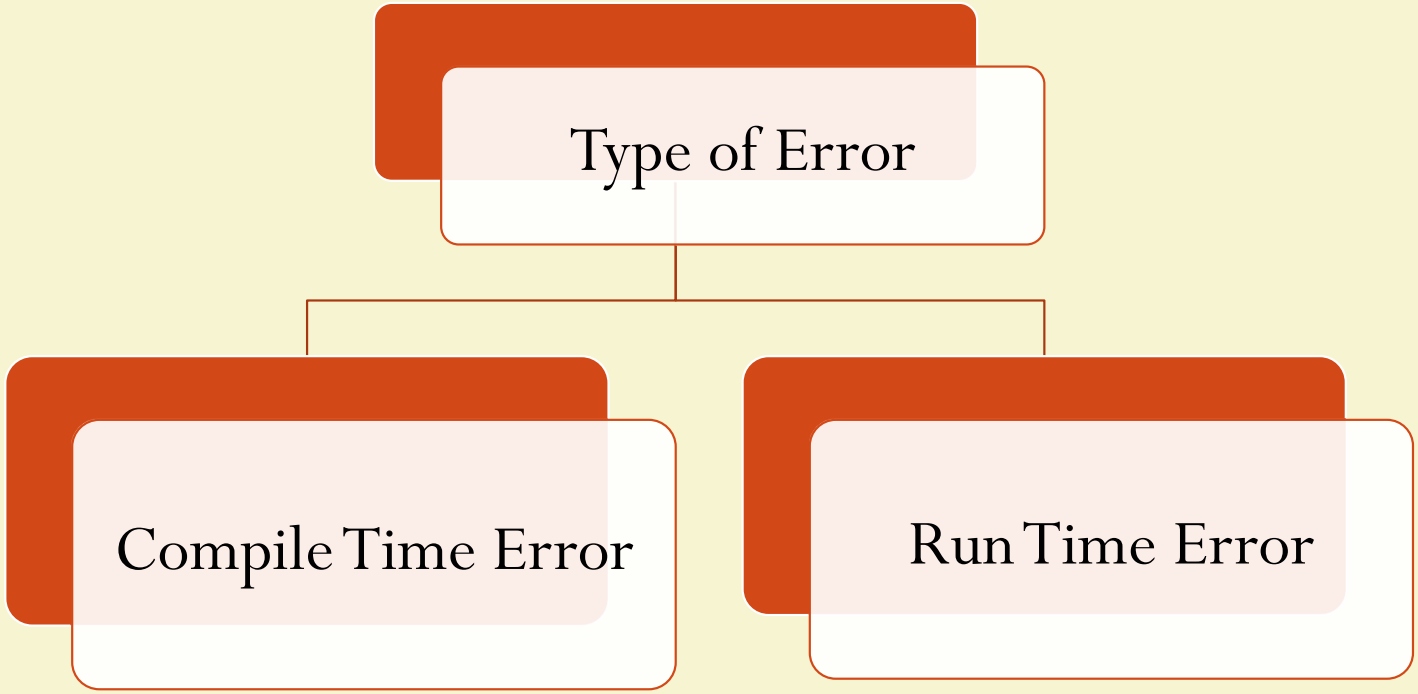
ABSTRACT CLASS	INTERFACE
1) Abstract class can have abstract and non-abstract methods.	Interface can have only abstract methods. Since Java 8, it can have default and static methods also.
2) Abstract class doesn't support multiple inheritance.	Interface supports multiple inheritance.
3) Abstract class can have final, non-final, static and non-static variables.	Interface has only static and final variables.
4) Abstract class can provide the implementation of interface.	Interface can't provide the implementation of abstract class.
5) The abstract keyword is used to declare abstract class.	The interface keyword is used to declare interface.
6) Example: public abstract class Shape{ public abstract void draw();}	Example: public interface Drawable{ void draw();}

Fundamental of JAVA Programming

UNIT :- V

Exception Handling , Multithreading, Applet

Types of Errors



Compile Time Errors

- All syntax errors will be detected and displayed by the Java compiler ,these errors are called as compile time errors.
- Whenever the compiler displays an error, it will not create the .class file
- It is necessary to fix all these errors before we can successfully compile and run the program

```
class Myclass
{
   public static void main (String args [])
  {
     System.out.println("Hello Java")
   }
}
```

Syntax error, insert ";" to complete BlockStatements

Reasons for Compile Time Errors

- Missing Semicolons
- Missing brackets in classes and methods
- Misspelling of identifiers and keywords
- Missing double quotes in string
- Used of undeclared variables
- Bad Reference to objects
- Use of = in place of == operator

Run Time Errors

- Sometime, a program may compile successfully creating the .class file but may not run properly.
- Such programs may produce wrong results due to wrong logic or may terminate due to errors.
- Most Run time Errors are

1. Dividing an Integer by Zero
2. Out of the bounds of an array
3. Store a value into an array of an incompatible type
4. Passing a parameter that is not in a valid range or value for method
5. Attempting negative size of an array
6. Converting invalid string to a number
7. Accessing a character that is out of bounds of a string

Run Time Error Example

```
class Myclass
{
   public static void main (String args [])
  {
     int a=10;
     int b=5;
     int c=5;
     int x=a/(b-c);  //Division by zero
      System.out.println(" x="+x);
      int y=a/(b+c);
     System.out.println("y="+y);
   }
}
```

```
Java.lang.ArithmeticException: /by zero
At Myclass.main(Myclass.java:)
```

Exceptions

- Exception is a condition that is caused by a run time error in the program.
- When the Java interpreter encounters an error such as dividing an integer by zero, it creates an exception object and throws
- If the exception object is not caught and handled properly, the interpreter will display an error message as shown in the output of program and will terminate the program.
- If we want the program to continue with the execution of the remaining code, then we should try to catch the exception object thrown by the error condition and then display an appropriate message for taking corrective actions. This task is known as exception handling.

Exceptions Type

Exception Type	Cause of Exception
ArithmaticException	Caused by math errors such as division by zero
ArrayIndexOutOfBoundException	Caused by bad array indexes
ArrayStoreException	Caused when a program tries to store the wrong type of data in array
FileNotFoundException	Caused by an attempt to access a nonexistent file
IOException	Caused by general I/O failures, such as inability to read from a file
NumberFormatException	Caused when a conversion between strings and numbers fails
OutOfMemoryException	Caused when there's not enough memory to allocate a new object
SecurityException	Caused when an applet tries to perform an action not allowed by the browser security setting
StackOverFlowException	Caused when the system runs out of stack space
StringIndexOutOfBoundException	Caused when a program attempts to access a nonexistent character position in a String

AAAINFOSSS

Error Handling

- For Error Handling performs the following tasks

1. Find the problem (Hit the exception)
2. Inform that an error has occurred (throw the exception)
3. Receive the error information (catch the exception)
4. Take corrective actions (handle the exception)

- Java Exception Handling is managed by five keywords

```
try{.....}
catch{.....}
throw
throws
finally{.....}
```

Syntax of Exception Handling Code

```
……..
……..
try
{
   statement;   //Generates an Exception
}
catch (Exception-Type e)
{
   statement ;     //Processes the exception
}
……..
……..
```

try block
Statement that cause an exception

Exception Object Creator

Throw Exception Object

catch block
Statement that handle an exception

Exception Handler

Exception Handling In Java

```
class Myclass
{
   public static void main (String args [])
  {
     int a=10;
     int b=5;
     int c=5;
     int x,y;
     try
    {
     x=a/(b-c);  //Exception Here
      System.out.println(" x="+x);
    }
    catch(ArithmeticException e)
    {
      System.out.println("Division by zero"+ "Please take a
      different value of c" );
     }
     int y=a/(b+c);
     System.out.println("y="+y);
}
```

Output:-

Division by zero Please take a different value of c

y=1

Exception Handling In Java

```
class CLineInput
{
   public static void main (String args [])
  {
     int invalid= 0;     //Number of Invalid Argument
     int number, count=0
     for(int i=0;i<args.length;i++)
     {
       try
       {
         number=Integer.parseInt(args[i]);
       }
     catch(NumberFormatException e)
     {
       invalid=invalid +1;   //Caught an Invalid Number
       System.out.println("Invalid Number:- "+args[i]);
       continue;
     }
       count=count+1;
     }    //End of For loop
    System.out.pirntln("Total Valid Number="+count);
    System.out.println("Total Invalid Number="+invalid);
  }
}
```

Java ClineInput 15 25.75 40 Java 10.5 65

Output:-

Invalid Number:- 25.75

Invalid Number:- Java

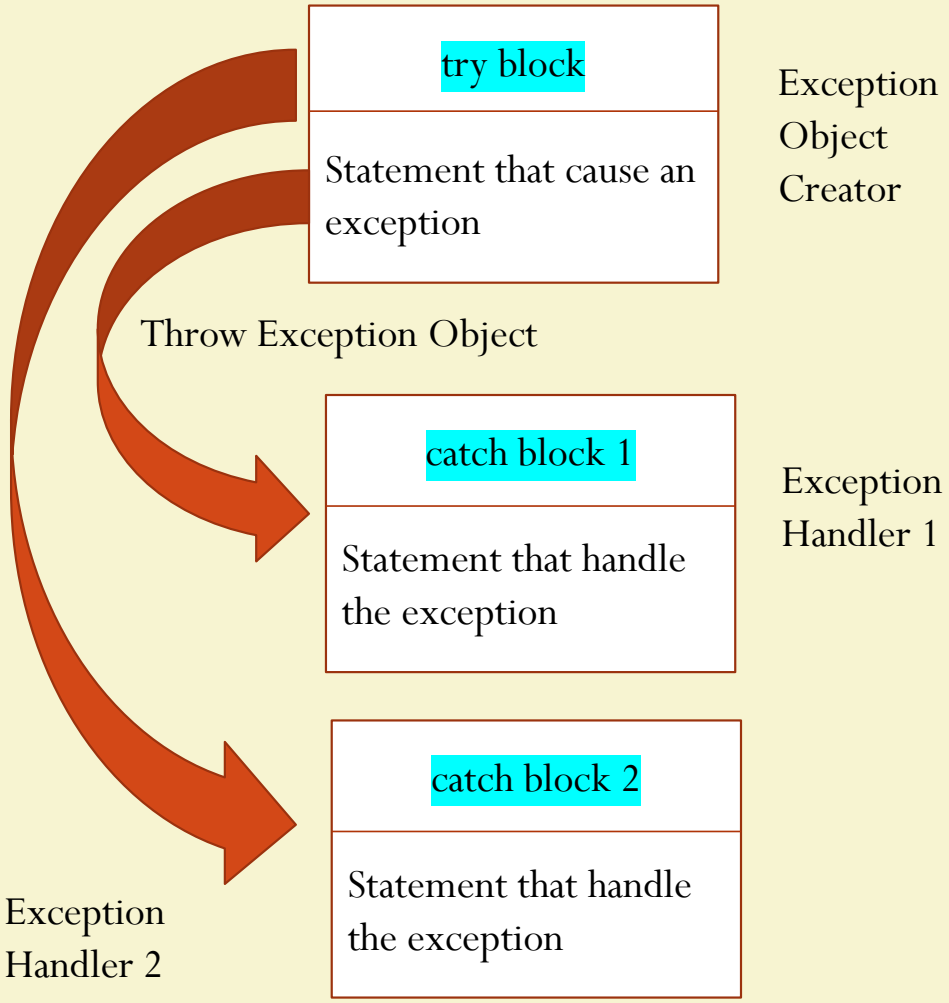
Invalid Number:- 10.5

Total Valid Number:- 3

Total Invalid Number:- 3

Multiple Catch Statement

```
.....
try
{
   statement;   //Generates an Exception
}
catch (Exception-Type-1 e)
{
   statement ;     //Processes the exception type 1
}
catch(Exception-Type-2 e)
{
  statement; //Process the exception type 2
}
.
.
.
catch(Exception Type-N e)
{
    statement;
}
```



Example of Multiple Catch Blocks

```
class MyClass
{
   public static void main (String args [])
  {
     int a[] = {5,10};
     int b=5;
      try
      {
         int x=a[2]/b – a[1];
      }
     catch(ArithmeticException e)
     {
      System.outprintln("Division by zero");
      }
      catch (ArrayIndexOutOfBoundsException e)
     {
         System.out.println("Array Index Error");
      }
catch(ArrayStoreException e)
      {
         System.out.println("Wrong  Data Type");
      }
      int y=a[1]/a[0];
     System.out.println("y="+y);
  }
}
```

Output:-

Array Index Error

y =2

Exception Handling Code with Finally Statement

- Finally statement that can be used to handle an exception that is not caught by any of the previous catch statements.
- Finally block can be used to handle any exception
- It may be added immediately after the try block or after the last catch block
- When finally block is defined, then it is guaranteed to execute, regardless of whether or not an exception is thrown

```
try
{
   statement;   //Generates an Exception
}
finally
{
   statement ;     //Processes the exception
}
.....................
```

```
try
{
   statement;   //Generates an Exception
}
catch (Exception-Type e)
{
   statement ;     //Processes the exception
}
catch (Exception-Type e)
{
   statement ;     //Processes the exception
}
finally
{
   statement ;     //Processes the exception
}
```

Example of Multiple Catch Blocks with Finally

```
class MyClass
{
   public static void main (String args [])
  {
     int a[] = {5,10};
     int b=5;
      try
      {
         int x=a[2]/b – a[1];
      }
     catch(ArithmeticException e)
     {
      System.outprintln("Division by zero");
      }
      catch (ArrayIndexOutOfBoundsException e)
      {
         System.out.println("Array Index Error");
      }
     catch(ArrayStoreException e)
      {
         System.out.println("Array Index Error");
      }
      catch(ArrayStoreException e)
      {
         System.out.println("Wrong  Data Type");
      }
      finally
      {    int y=a[1]/a[0];
           System.out.println("y="+y);
      }
}
```

Output:-

Array Index Error

y =2

Unit No.5, Part I Assignment No.9

- Implement the exception handling using try and catch statements to solve runtime errors.
- Assignment Deadline :-

Fundamental of JAVA Programming

UNIT :- V

Exception Handling , Multithreading, Applet

Topic

Applet

Applet

- Java Programs are available in two flavours
- Application :- It is similar to all other kind of programs like in C , C++ etc. to solve a real time problem
- Applet :- Applet is small Java programs that are primarily used in internet computing
- They can be transported over the Internet from one computer to another and run using the Applet viewer or any web browser that supports Java.
- An applet , like any application program, can do many things for us. It can perform arithmetic operations, display graphics, play sounds, accept user input, create animation and play interactive games.
- A web page can now contain not only a simple text or a static image but also a Java applet which run graphics, sounds and moving images.

Local and Remote Applet

- We can embed applets into web pages in two ways
- One , we can write our own applet and embed them into web pages
- Second, we can download an applet from a remote computer system and then embed into a web page.

How Applets Differ from Applications

- Applets and stand alone applications are Java programs
- Significant difference between them:- Applets are not full featured application programs.
- They are usually written to complete a small task or a component of a task.
- They are usually designed for use on the internet

1. Applets do not use the main() method for initiating the execution of the code
2. Unlike stand alone application, applets can not be run independently
3. Applets can not read form or write to the files in the local computer
4. Applets can not communicate with other servers on the network
5. Applets can not run any program from the local computer

Preparing to Write Applets

- Before we try to write applets, we must make sure that Java is installed properly, and also ensure that either the Java appletviewer or a Java enabled browser is available
- Steps involved in developing and testing in applet are

1. Building an applet code (.java file)
2. Creating an executable applet (.class file)
3. Designing a web page using HTML tag
4. Preparing <APPLET>tag
5. Incorporating <APPLET> tag into the web page
6. Creating HTML file
7. Testing the applet code

Building Applet Code

- Our applet code use the services of two classes, namely, Applet and Graphics from the Java class library.
- Applet class which is contained in the java.applet package provides life and behaviour of applet through its methods such as init(), start(), point(), close().
- Therefore Applet class is the life cycle of applet.
- paint () method of the Applet class, when it is called, actually displays the result of the applet code on the screen. Output may be text, graphics, sound etc.
- paint() method, which requires a Graphics object as an argument is defined as

```
public void paint(Graphics g)
```

- This requires that the applet code import the java.awt package that contain the Graphics class.
- All output operation of an applet are performed using the methods defined in the Graphics class.

Building an Applet Code

- Process-1) Import Applet and Graphics class from applet and awt package respectively
- Process -2) Define applet class from extending the properties of Applet class
- Process-3) Override the method paint that shows the result of applet code
- Process -4) By using object of Graphics class access the methods that perform the all operation of applet
- Compile this file to creating a .class file for execution of applet code

```
//File Name is :- AppletClassName.java
import java.applet.Applet;
import java.awt.Graphics;

public class AppletClassName extends Applet
{
    ----------------
     public void paint(Graphics g)
     {
       ----------
       ----------
     }
      ---------------
      ---------------
}
```

Testing a Applet Code

- We know the applet are programs that reside on web pages. In order to run a Java applet, it is first necessary to have a web page that references that applet.
- Basically web page made up of text and HTML tags that can be interpreted by web browser or an applet viewer.
- A web page also called as HTML page
- A web page is marked by an opening HTML tag <HTML> and a closing HTML tag </HTML> and its divided into three major sections:

1. Comment Section (Optional)
2. Head Section (Optional)
3. Body Section

Testing a Applet Code

<HTML>

<!
---- -------

>

<HEAD>
Title Tag
</HEAD>

<BODY>
Applet Tag
</BODY>

</HTML>

Comment Section

Head Section

Body Section

<applet code = “AppletClassName.class” width =“300” Height=“300”>
</applet>

Save as .html and use appletviewer filename.html commaned to run our applet code

HTML File that hosts an APPLETN

```
<HTML>
  <HEAD>
     <TITLE> A sim
  </HEAD>
  <BODY>
    <APPLET CODE="
    <H1>
       A browser s                                    / it on the screen.
    </H1>
    <B><I>These are some text in Italics and Bold font.</I></B>
```

Note:

1. HTML is case insensitive language.
2. Browser interprets each tag and ignores portion, which is unable to interpret or if there is any error.

Element					
<applet>	Not supported	Not supported	Yes	Yes	Not supported

Note: There is still some support for the <applet> tag in some browsers, but it requires additional plug-ins/installations to work.

Note: The <applet> tag is supported in Internet Explorer 11 and earlier versions, using a plug-in.

The HTML APPLET Tag : Syntax

The syntax for the standard APPLET tag is shown here. *Bracketed items are optional.*

```
<APPLET
  [CODEBASE = URL to an applet]
   CODE = applel class filename
  [ARCHIVES = Name of a JAR file]
  [OBJECT Serialized applet filename]
  [ALT = alternate text]
  [NAME = applet instance name]
   WIDTH = pixels HEIGHT = pixels
  [ALIGN = alignment type]
  [VSPACE = pixels] [HSPACE = pixels]
>
  [< PARAM NAME = AttributeName VALUE = AttributeValue>]
  [< PARAM NAME = AttributeName2 VALUE = AttributeValue>]
  . . .
  [Some text to be displayed in the absence of Applet]
</APPLET>
```

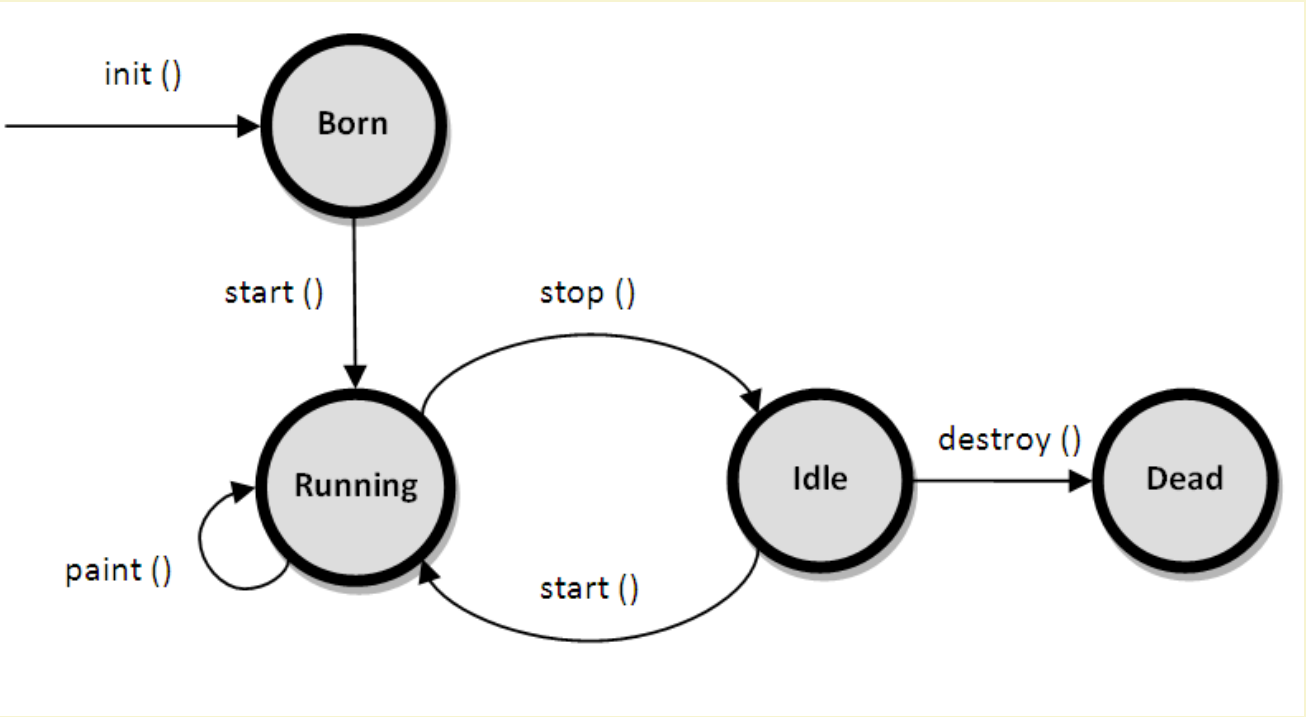
Running an HTML file: An example

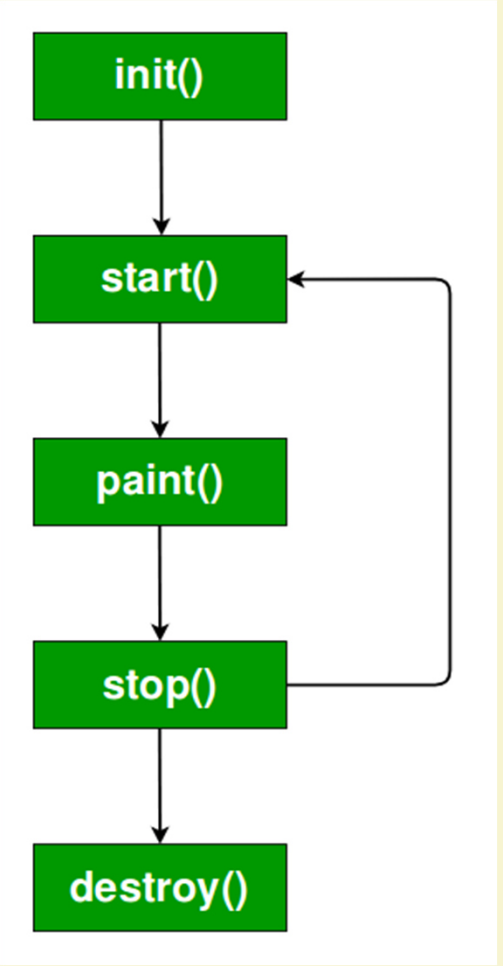
```
import java.awt.*;
import java.applet.*;
public class StatusWindow extends Applet{
  public void init() {
    setBackground(Color.cyan);
  }
  public void paint(Graphics g) {
    g.drawString("This is in the applet window.", 10, 20);
    showStatus("This is shown in the status window.");
  }
/*
<html>
   <body>
      <applet width="300" height="50" code="StatusWindow.class">
      </applet>
   </body>
</html>
*/
```

Tips:

1. Save this file as `StatusWindow.java`
2. Compile it.
3. Run it as : **`appletviewer StatusWindow`**

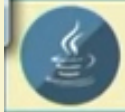
Applet Life Cycle





Basic Methods of Applet

- public void init() – To initialize or pass input to an applet
- public void start()- start() called after the init() method and it starts an applet
- public void stop()- To stop running applet
- public void paint(Graphics G)- To draw something within an applet
- Public void destroy() – To remove an applet from memory completely



Order of invocation of Applet methods

- These are abstract methods defined in abstract class Applet and they need to be overridden in applet programs. It is important to note the order in which the various methods are called.

- When an applet begins, the AWT calls the method in the sequence

`init()` → `start()` → `paint()`

- When an applet is terminated, the following sequence of methods call takes place

`stop()` → `destroy()`

- Note: All these methods are optional.

Applet initialization and start : Methods

init( )

The `init()` method is the first method to be called. This is where you should initialize an applet. This method is called only once during the run time of your applet.

start( )

The `start()` method is called after `init()`. It is also called to restart an applet after it has been stopped. Whereas `init()` is called once—the first time an applet is loaded—`start()` is called each time an applet's HTML document is displayed onscreen. So, if a user leaves a web page and comes back, the applet resumes execution at `start()`.

Applet paint method

paint( )

The `paint()` method is called each time the applet's output to be redrawn. This situation can occur for several reasons. For example, the window in which the applet is running may be overwritten by another window and then uncovered. Or the applet window may be minimized and then restored. `paint()` is also called when the applet begins execution. Whatever the cause, whenever the applet must redraw its output, `paint()` is called. The `paint()` method has one parameter of type `Graphics`. This parameter will contain the graphics context, which describes the graphics environment in which the applet is running. This context is used whenever output to the applet is required.

Applet termination : Methods

stop( )

The `stop()` method is called when a web browser leaves the HTML document containing the applet—when it goes to another page, for example. When `stop()` is called, the applet is probably running. You should use `stop()` to suspend threads that do not need to run when the applet is not visible. One can restart them when `stop()` is called if the user returns to the page.

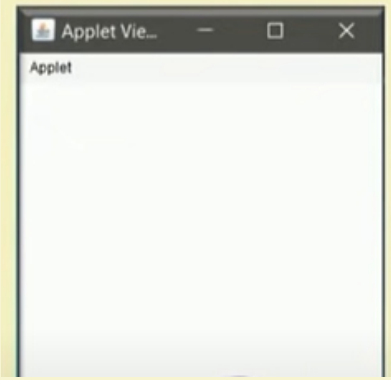
destroy( )

The `destroy()` method is called when the environment determines that your applet needs to be removed completely from memory. At this point, you should free up any resources the applet may be using. The `stop()` method is always called before `destroy()`.

Applet with blank methods

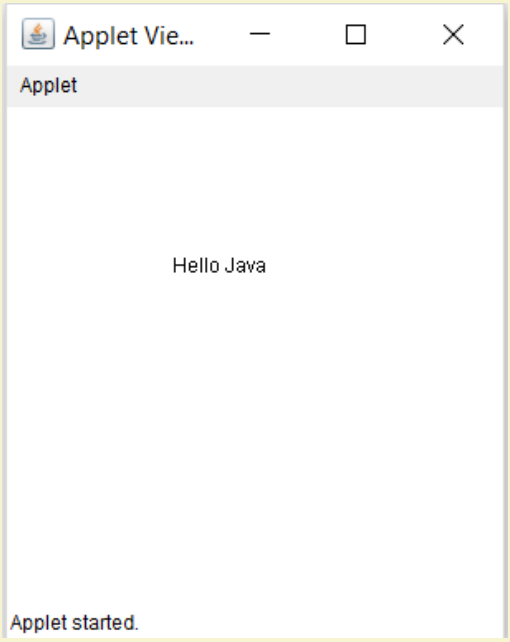
```
import java.awt.*;
import java.applet.*;

public class AppletSkeleton extends Applet {
  public void init() {  }
  public void start() {  }
  public void stop() {  }
  public void destroy() {  }
  public void paint(Graphics g) {  }
}
```

```
<html>
   <body>
      <applet width="300" height="300" code="AppletSkeleton.class">
      </applet>
   </body>
</html>
```



Example :- Print the Hello Java on Applet Window

```
import java.awt.*;
import java.applet.*;
public class HelloJava extends Applet
{
    public void paint(Graphics g)
    {
      g.drawString("Hello Java", 100,100);
    }
}

/*
  <html>
     <body>
       <applet code= "HelloJava.class" width ="300" height="300" >
       </applet>
     </body>
  </html>
*/
```

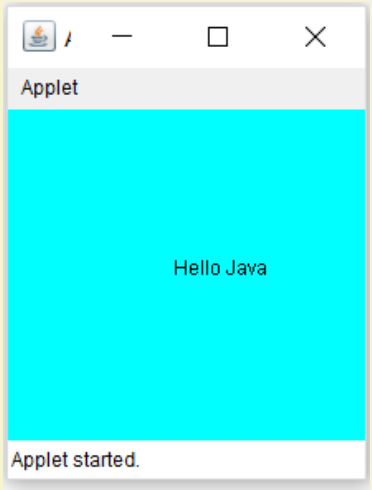


Commonly used methods of Graphics class:

1. **public abstract void drawString(String str, int x, int y):** is used to draw the specified string.
2. **public void drawRect(int x, int y, int width, int height):** draws a rectangle with the specified width and height.
3. **public abstract void fillRect(int x, int y, int width, int height):** is used to fill rectangle with the default color and specified width and height.
4. **public abstract void drawOval(int x, int y, int width, int height):** is used to draw oval with the specified width and height.
5. **public abstract void fillOval(int x, int y, int width, int height):** is used to fill oval with the default color and specified width and height.
6. **public abstract void drawLine(int x1, int y1, int x2, int y2):** is used to draw line between the points(x1, y1) and (x2, y2).
7. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.
8. **public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used draw a circular or elliptical arc.
9. **public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used to fill a circular or elliptical arc.
10. **public abstract void setColor(Color c):** is used to set the graphics current color to the specified color.
11. **public abstract void setFont(Font font):** is used to set the graphics current font to the specified font.

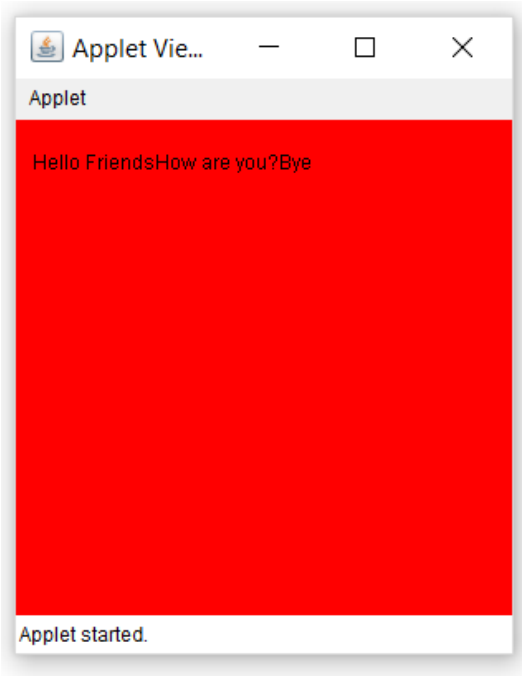
Example

```
import java.awt.*;
import java.applet.*;
public class HelloJava extends Applet
{
   public void init()
   {
     resize(200,200);
     setBackground(Color.cyan);
    }
   public void paint(Graphics g)
   {
      g.drawString("Hello Java", 100,100);
   }
}
/*
  <html>
   <body>
    <applet code= "HelloJava.class" width ="300" height="300" >
</applet>
</body>
</html>
```

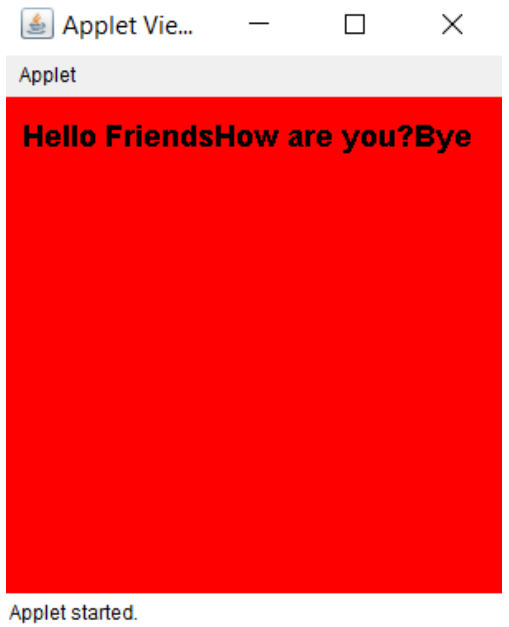


```
import java. applet.*;
import java. awt.*;

public class Sample extends Applet
{
  String msg;
  public void init()
  {
    setBackground(Color.red);   // set background and forground color
                                   of applet window
    setForeground(Color.black);
    msg="Hello Friends …. ";          // initialize the string to dispaly
  }
  public void start()
  {
    msg += "How are you?….";
   }
   public void paint(Graphics g)
   {
     msg+= "Bye…….";
    g.drawString(msg,10,30);
  }
}
```

```
 /*
<html>
 <body>
  <applet code="Sample.class" width="300" height="300">
   </applet>
 </body>
</html>
 */
```



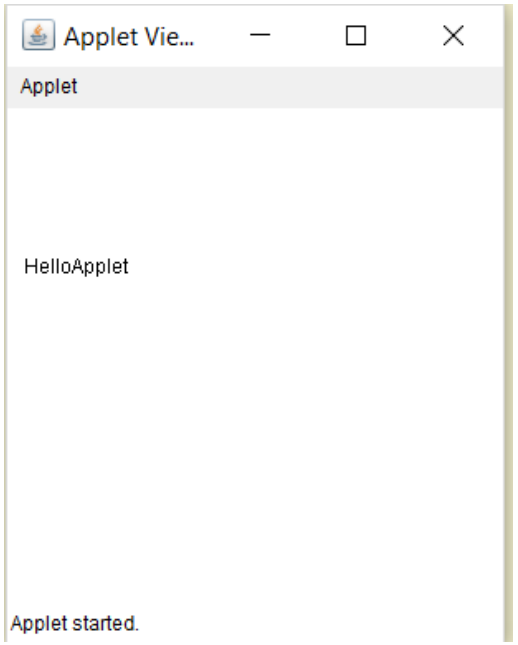
```
import java. applet.*;
import java. awt.*;

public class Sample extends Applet
{
  String msg;
   Font f1;
  public void init()
  {
   setBackground(Color.red);   // set background and forground color
                                  of applet window
   setForeground(Color.black);
   msg="Hello Friends …. ";          // initialize the string to display
   f1 =new Font (“Aerial”,Font.BOLD, 18);
  }
  public void start()
  {
    msg += "How are you?….";
   }
   public void paint(Graphics g)
   {
    msg+= "Bye…….";
    g.setFont(f1);
    g.drawString(msg,10,30);
   }
}


 /*
<html>
 <body>
  <applet code="Sample.class" width="300" height="300">
   </applet>
 </body>
</html>
 */
```



Passing a Parameter to Applet

```
import java.applet.Applet;
import java.awt.Graphics;

public class Test extends Applet
{
   String str;
  public void init()
   {

       str=getParameter("string");
       if (str ==null)
         str= "Java";
      str="Hello"+str;
   }
   public void paint(Graphics g)
   {
     g.drawString(str,10,100);
  }
}
```

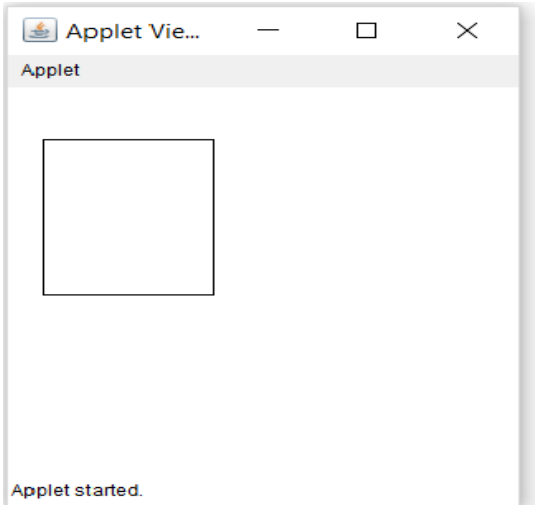
```
/*
<html>
  <body>
    <applet code="Test.class" width="300"
height="300">
  <param name="string" value="Applet">
</applet>
</body>
</html>
*/
```



Passing a Parameter to Applet

```
import java.awt.Graphics;
import java.applet.Applet;

public class Rectangle extends Applet
{
  int x,y,w,h;
  public void init()
  {
    x=Integer.parseInt(getParameter("xValue"));
    y=Integer.parseInt(getParameter("yValue"));
    w=Integer.parseInt(getParameter("wValue"));
    h=Integer.parseInt(getParameter("hValue"));
  }

  public void paint(Graphics g)
  {
    g.drawRect(x,y,w,h);
  }
}
```

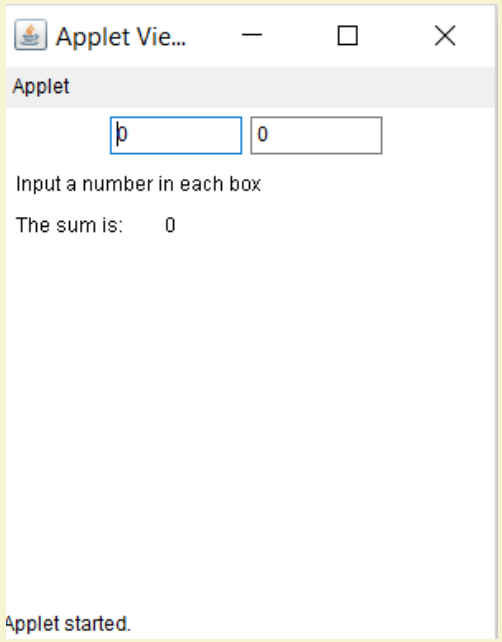
```
/*
<html>
  <body>
    <applet code="Rectangle.class" width="300"
height="300">
    <param name="xValue" value="20">
    <param name="yValue" value="40">
    <param name="wValue" value="100">
    <param name="hValue" value="120">
  </applet>
  </body>
  </html>
  */
```

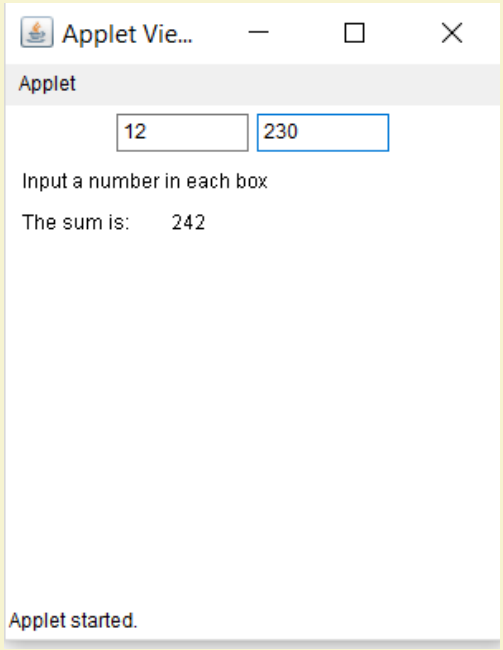


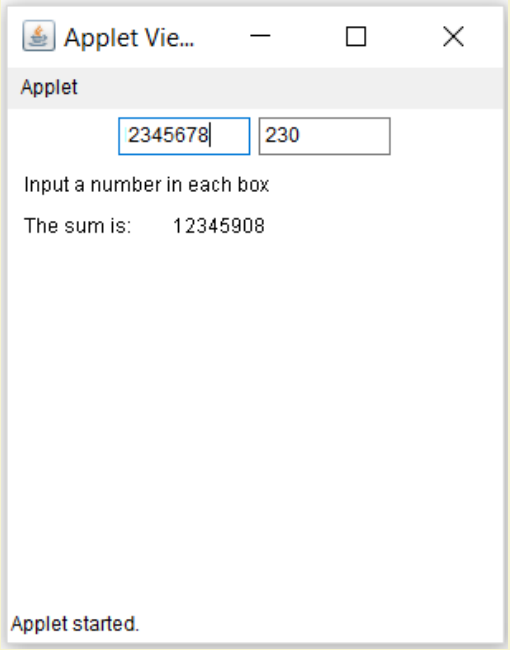
Passing a Parameter to Applet

```
import java.awt.*;
import java.applet.*;

public class UserIn extends Applet
{
  TextField text1, text2;
   public void init()
   {
    text1=new TextField(8);
    text2=new TextField(8);
   add(text1);
   add(text2);
   text1.setText("0");
    text2.setText("0");
  }
public void paint(Graphics g)
{
  int x=0,y=0,z=0;
  String s1,s2,s;
   g.drawString("Input a number in each
box",10,50);

   try
    {
     s1=text1.getText();
     x= Integer.parseInt(s1);
     s2=text2.getText();
      y=Integer.parseInt(s2);
    }
  catch(Exception e)
   {}
    z=x+y;
    s= String.valueOf(z);
    g.drawString("The sum is:",10,75);
    g.drawString(s,100,75);
  }

public boolean action (Event event, Object
object)
{
  repaint();
  return true;
}
}
/*
<html>
 <body>
  <applet code ="UserIn.class"
width="300" height="300">
 </applet.
 </body>
<html>
*/
```

Passing a Parameter to Applet







Unit No.5, Part III Assignment No.

- Create an applet with three text fields and four buttons add, substract, multiply and divide. User will enter two values in the Text Fields. When any button is pressed, the corresponding operation is performed and the results is displayed in the third text fields.

- Assignment Deadline :-

Fundamental of JAVA Programming

UNIT :- V

Exception Handling , Multithreading, Applet

Topic

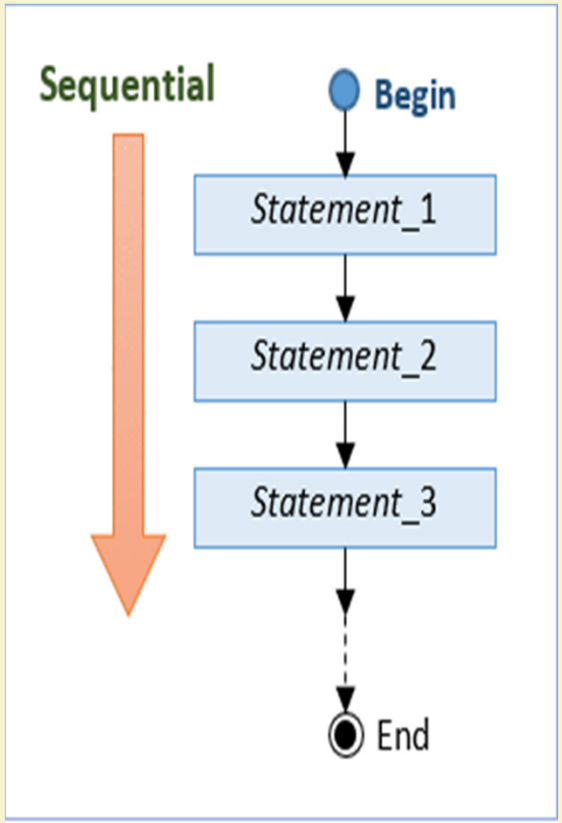
Multithreading

Multiple Threading

- Multitasking :- Whenever system can execute the several task simultaneously, then this ability called as Multitasking
- In system terminology it is called as multithreading
- Multithreading :- It is a conceptual programming paradigm where a program is divided into two or more subprograms, which can be implemented at the same time in parallel.
- Example:- One subprogram can display an animation on the screen while another may build the next animation to be displayed
- Similar to dividing a task into subtasks and assigning them to different people for execution independently and simultaneously.

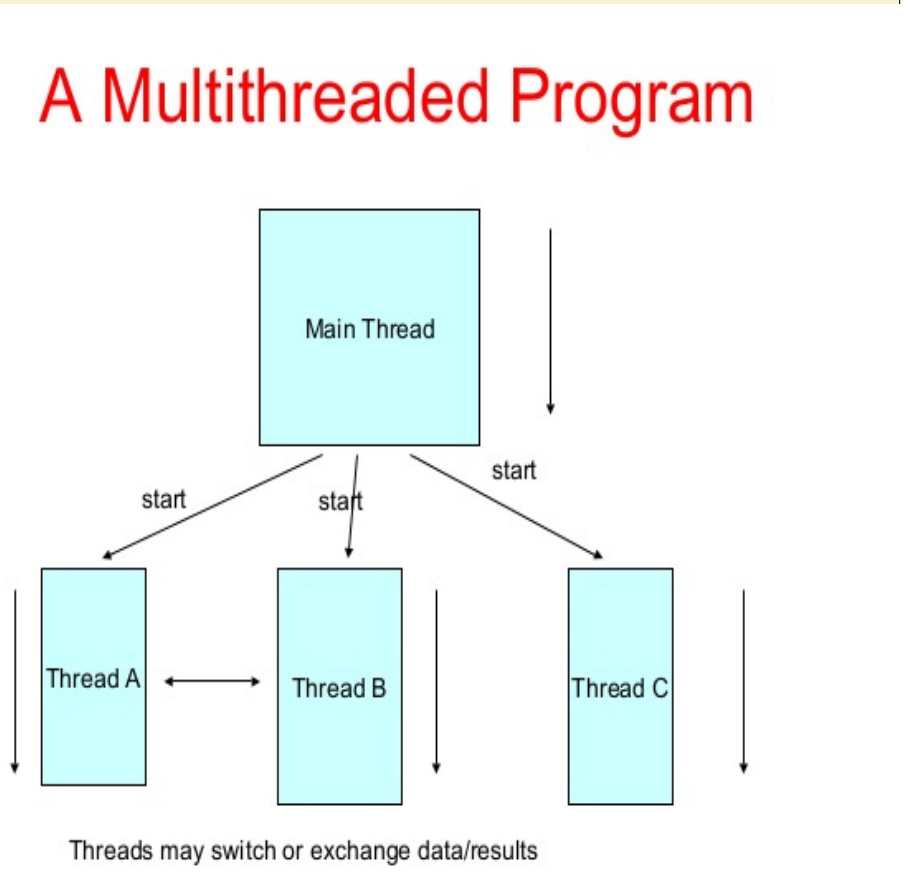
Multiple Threading

- Java programs that we have seen and discussed up to now, it has single sequential flow of control.
- Program begins, runs through a sequence of execution and finally ends.
- At a time, there is only one statement under execution
- Thread is similar to a program that has a single flow of control. (i.e all earlier examples can be called as single threaded programs.
- Every program will have at least one thread



Multiple Threading

- Unique property of java is its support for multithreading.
- Multiple flow of control, each flow of control may be through as a thread that runs in parallel.
- Multithreaded Program :- A program that contain multiple flow of control is known as multithreaded program
- It is important to remember that threads running in parallel does not mean that they actually run at the same time



Creating Threads

- Threads are implemented in the form of objects that contain a method called run().
- Run() method is the heart and soul of thread
- New thread can be created in two ways

1. By Creating a thread class
2. By converting a class to thread

By Creating a Thread Class

- Define a class thread that extends Thread class and override its run() method with the code required by the thread
- Process-1) Creating new thread class by using extends the Thread class which is mention in the lang package
- Process -2) Override the run() method which is mention the Thread class and implement the thread code
- Process-3) Create the object for new thread class
- Process -4) Invoked the run() method by using object of new thread class and start() method
- By using start method, we have to start the execution of run() method.

```
import java.lang.Thread;
class Mythread extends Thread
{
     public void run()
     {
       ----------
       ----------
     }
}

class Main
{
 public static void main(String args [])
{
   Mythread obj=new Mythread();
    obj.start();
 }
}
```

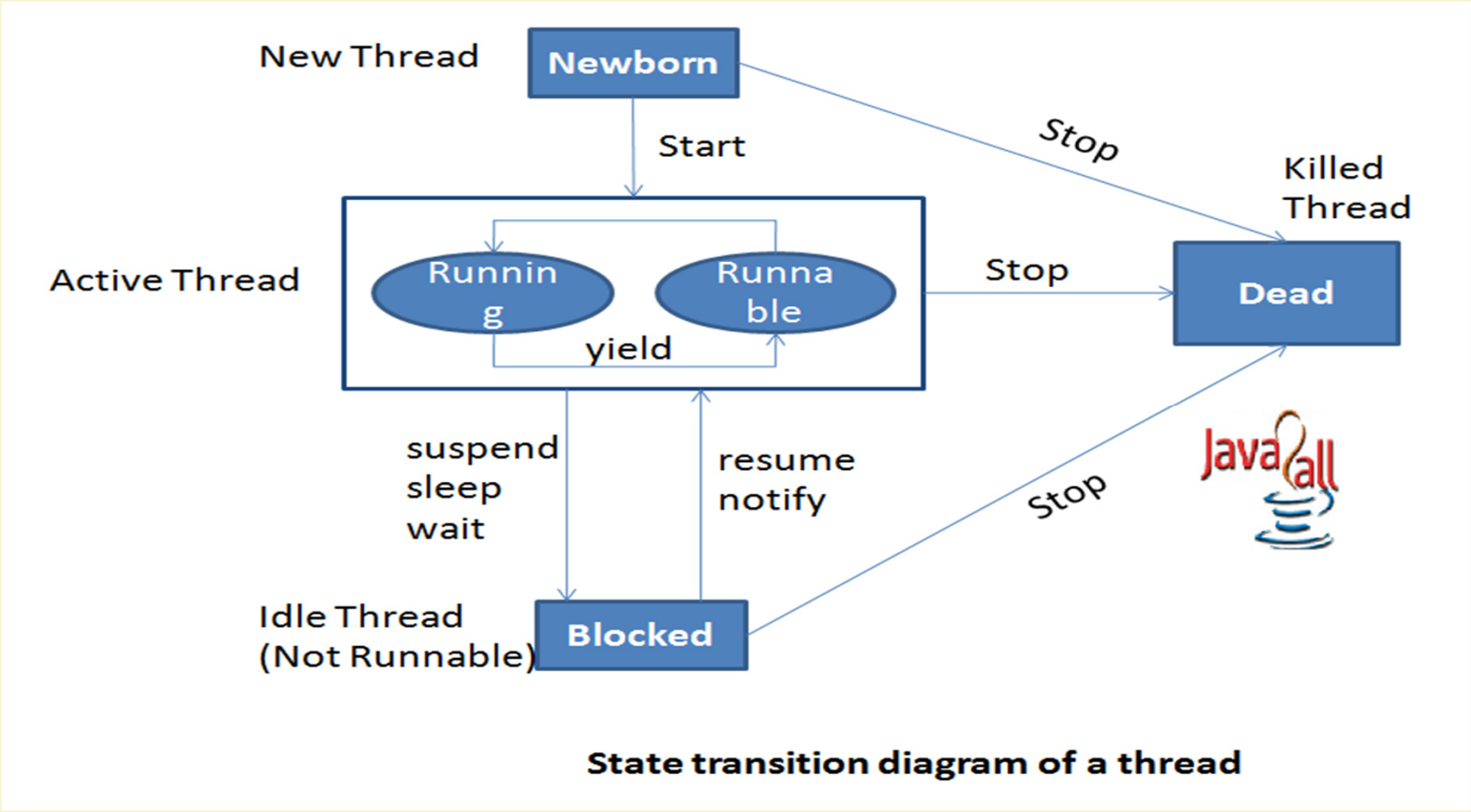
Stopping & Blocking Thread

- Stopping Thread:- Whenever we want to stop a thread from running further, we may calling stop() method
- e.g. aThread.stop();
- Blocking Thread :- A thread can also be temporarily suspended or blocked from entering into runnable and subsequently running state by using either of the following methods

```
sleep()          //Blocked for a specified time
suspended()      //Blocked until further orders
wait()           // Blocked until certain condition occurs
```

- The thread will return to the runnable state when the specified time is elapsed in the case of sleep(), the resume() method is invoked in the case of suspend(), and the notify() method is called in the case of wait().

Life Cycle ofThread

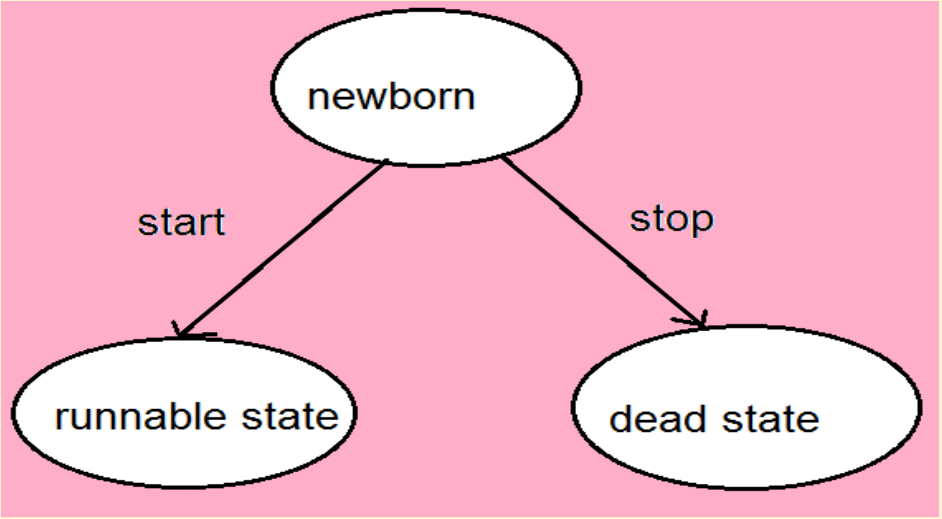


State transition diagram of a thread

Life Cycle ofThread

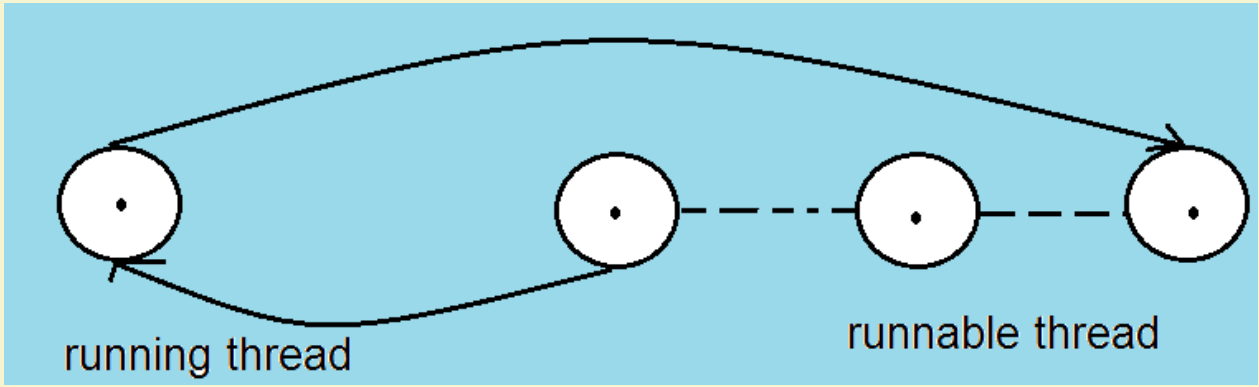
- Newborn State :- when we create a thread object the thread is born and is said to be in newborn state.
- Thread is not yet scheduled for running
- At this state, we can do only one of the following things with it

1. Schedule it for running using start() method
2. Kill it using stop() method



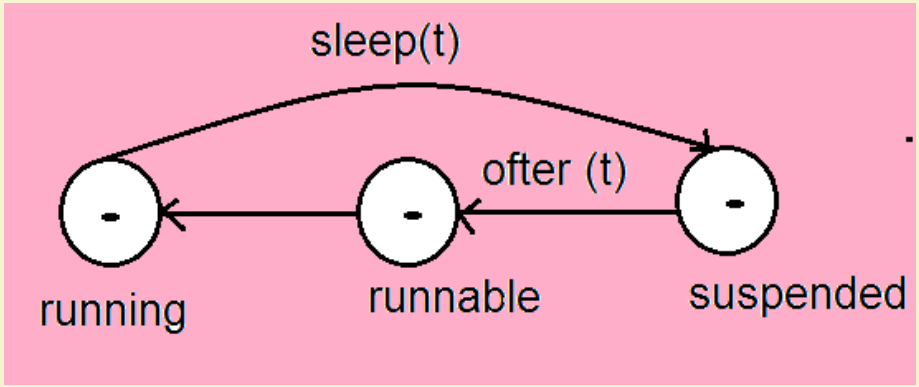
Life Cycle of Thread

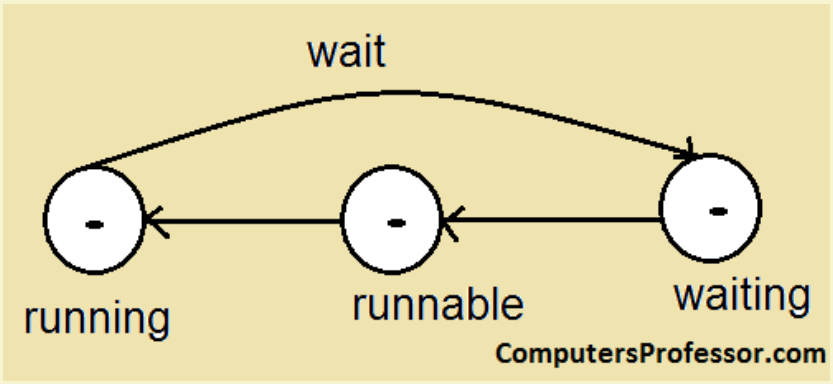
- Runnable State :- Runnable state means that the thread is ready for execution and is waiting for the availability of the processor
- That means thread has joined the queue of threads that are waiting for execution



Life Cycle ofThread

- Running State :- Running state means that the processor has given its time to the thread for its execution
-





Thread Priority

- Thread Priority :-
- In Java, each thread is assigned a priority, which affects the order in which it is scheduled for running.
- Java permits us to set the priority of a thread using the setPriority() method as follows
- ThreadName.setPriority(intNumber);
- The intNumber is an integer value to which the threads priority set.
- intNumber is any value between 1 to 10
- Note that default setting is NORM_PRIORITY

MIN_PRIORITY =1

NORM_PRIORITY =5

MAX_PRIORITY =10

304185C

Unit 6: Graphics Programming & File Handling

TE E&TE

Syllabus

- Graphics class, Introduction to AWT packages, Handling events on AWT components, Introduction to Swing package, components and containers.
- Managing input/output files: Concept of streams, Stream Classes, Byte stream, Character stream, Using Stream, creation of files, reading or writing characters / bytes, Concatenating and buffering files, Random access files.

Graphics Class in JAVA

public abstract class Graphics extends Object

- The Graphics class is the abstract base class for all graphics contexts that allow an application to draw onto components that are realized on various devices, as well as onto off-screen images.
- A Graphics object encapsulates state information needed for the basic rendering operations that Java supports.

Graphics Class

This state information includes the following properties:

The Component object on which to draw.

- A translation origin for rendering and clipping coordinates.
- The current clip.
- The current color.
- The current font.
- The current logical pixel operation function (XOR or Paint).
- The current XOR alternation color (setXORMode(java.awt.Color))

Java AWT

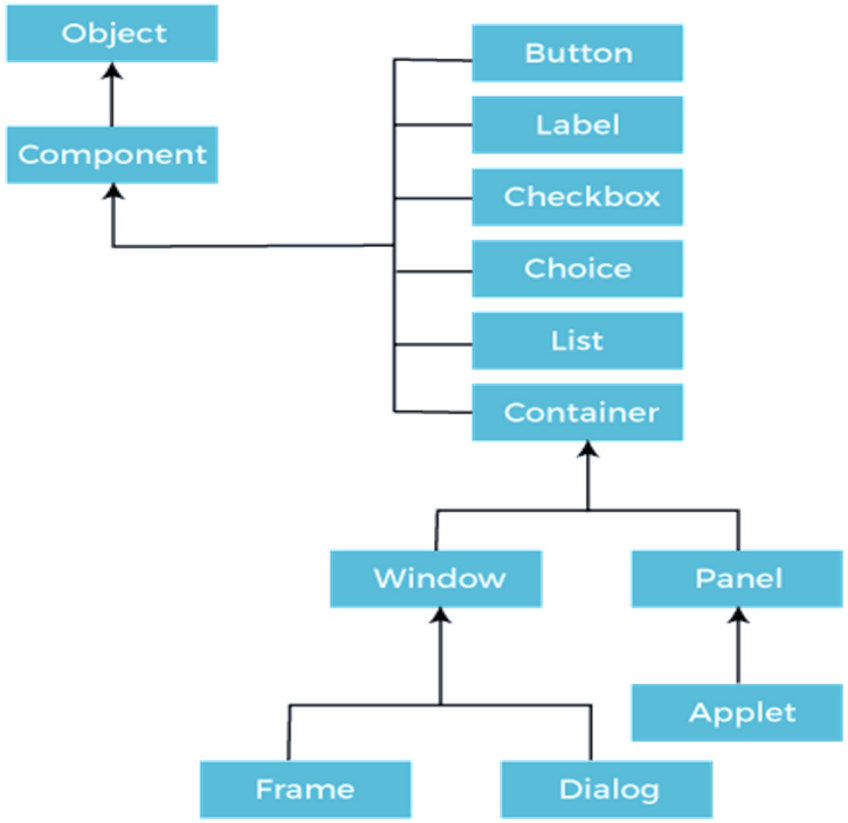
- Java AWT (Abstract Window Toolkit) is an API to develop Graphical User Interface (GUI) or windows-based applications in Java.
- Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavy weight i.e. its components are using the resources of underlying operating system (OS).
- The java.awt package provides classes for AWT API such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

Java AWT is Platform Independent

Pune Institute of Computer Technology, Pune

- Java AWT calls the native platform calls the native platform (operating systems) subroutine for creating API components like TextField, ChechBox, button, etc.
- For example, an AWT GUI with components like TextField, label and button will have different look and feel for the different platforms like Windows, MAC OS, and Unix. The reason for this is the platforms have different view for their native components and AWT directly calls the native subroutine that creates those components.
- In simple words, an AWT application will look like a windows application in Windows OS whereas it will look like a Mac application in the MAC OS.

Java AWT Hierarchy



https://www.javatpoint.com/java-awt

Component Class

- The Component class is the superclass of all components.
- A component class can be linked with a page, components of web applications. Component is the graphical representation of an Object.
- Types of Components in Component Class
  - Container
  - Button
  - Label
  - Checkbox
  - Choice
  - List
- All these components are present in **java.awt** package. We can import each of the components individually i.e., **import java.awt.Button**, **import java.awt.Container** etc.

Java AWT Controls

- **Container:** The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. The classes that extends Container class are known as container such as Frame, Dialog and Panel.
- **Window:** The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window.
- **Panel:** The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.
- **Frame:** The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.

Useful Methods of Component Class

Method	Description
public void add(Component c)	Inserts a component on this component.
public void setSize(int width,int height)	Sets the size (width and height) of the component.
public void setLayout(LayoutManager m)	Defines the layout manager for the component.
public void setVisible(boolean status)	Sets the visibility of a component to visible or not. If it sets to true, then the component will be visible in the output else if it sets to false or not defined component won't be visible in the output.

AWT UI Elements

- Label - A Label object is a component for placing text in a container.
- Button - This class creates a labeled button.
- Check Box - A check box is a graphical component that can be in either an on (true) or off (false) state.
- Check Box Group - The CheckboxGroup class is used to group the set of checkbox.
- List - The List component presents the user with a scrolling list of text items.
- Text Field - A TextField object a text component that allows for the editing of a single line of text.
- Text Area - A TextArea object is a text component that allows for the editing of a multiple lines of text.
- Choice - A Choice control is used to show pop up menu of choices. Selected choice is shown on the top of the menu.

AWT UI Elements

- Canvas - A Canvas control represents a rectangular area where application can draw something or can receive inputs created by user.
- Image - An Image control is superclass for all image classes representing graphical images.
- Scroll Bar - A Scrollbar control represents a scroll bar component in order to enable user to select from range of values.
- Dialog - A Dialog control represents a top-level window with a title and a border used to take some form of input from the user.
- File Dialog - A FileDialog control represents a dialog window from which the user can select a file.

Java Event Handling

Event:

- Change in the state of an object is known as event i.e., event describes the change in state of source.
- Events are generated as result of user interaction with the graphical user interface components.
- For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page are the activities that causes an event to happen.

Types of Events: The events can be broadly classified into two categories:

- Foreground Events
- Background Events

Types of Events

Foreground Events:

- The events which require direct interaction of user.
- They are generated as consequences of a person interacting with the graphical components in Graphical User Interface.
- For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page etc.

Background Events:

- The events that require the interaction of end user are known as background events.
- Operating system interrupts, hardware or software failure, timer expires, an operation completion are the examples of background events.

Event Handling

- Event Handling is the mechanism that controls the event and decides what should happen if an event occurs.
- This mechanism have the code which is known as event handler that is executed when an event occurs.
- Java Uses the Delegation Event Model to handle the events. This model defines the standard mechanism to generate and handle the events.
- The Delegation Event Model has the following key participants namely:
- Source
  - The source is an object on which event occurs.
  - Source is responsible for providing information of the occurred event to it's handler.

Event Handling

- Listener:
  - It is also known as event handler.
  - Listener is responsible for generating response to an event.
  - From java implementation point of view the listener is also an object.
  - Listener waits until it receives an event.
  - Once the event is received, the listener process the event and then returns.

Steps in Event Handling

- The User clicks the button, and the event is generated.
- The object of concerned event class is created automatically and information about the source and the event get populated with in same object.
- Event object is forwarded to the method of registered listener class.
- The method is now gets executed and returns.

Java Adapter Classes

- Java adapter classes provide the default implementation of listener interfaces.
- The adapter classes are found in java.awt.event, java.awt.dnd and javax.swing.event packages.

Working with Color, and Font.

Working with Color:

- Java supports color in a portable, device independent fashion.
- The AWT color system allows you to specify any color you want.
- Color is encapsulated by the Color class. The most used constructors are:
  - Color (int red, int green, int blue)
  - Color (int rgbValue)
  - Color (float red, float green, float blue)

Working with Color

- The first constructor takes three integers that specify the color as a mix of red, green, and blue.
- These values must be between 0 and 255.
- Example: new Color (255, 100, 100); // light red
- The second color constructor takes a single integer that contains the mix of red, green, and blue packed into an integer.
- The integer is organized with red in bits 16 to 23, green in bits 8 to 15, and blue in bits 0 to 7
- Example: int newRed = (0xff000000 | (0xc0 << 16) | (0x00 << 8) | 0x00);

  Color darkRed = new Color(newRed);
- The final constructor, Color (float, float, float), takes three float values (between 0.0 and 1.0) that specifies the relative mix of red, green, and blue.

Setting the Current Graphics Color

- By default, graphics objects are drawn in the current foreground color.
- We can change this color by calling the Graphics method setColor() - void setColor (Color newColor);
- Here, newColor specifies the new drawing color.
- We can obtain the current color by calling getColor() - Color getColor();

Working with Fonts:

- The AWT supports multiple type fonts.
- Fonts are encapsulated by the Font class.

Creating and Selecting a Font

- To select a new font, you must first construct a Font object that describes that font.
- One Font constructor has this general form - Font(String fontName, int fontStyle, int pointSize);
  - Here, fontName specifies the name of the desired font.
- To use a font that you have created, you must select it using setFont(), which is defined by Component.
- It has this general form - void setFont(Font fontObj);
  - Here, fontObj is the object that contains the desired font.

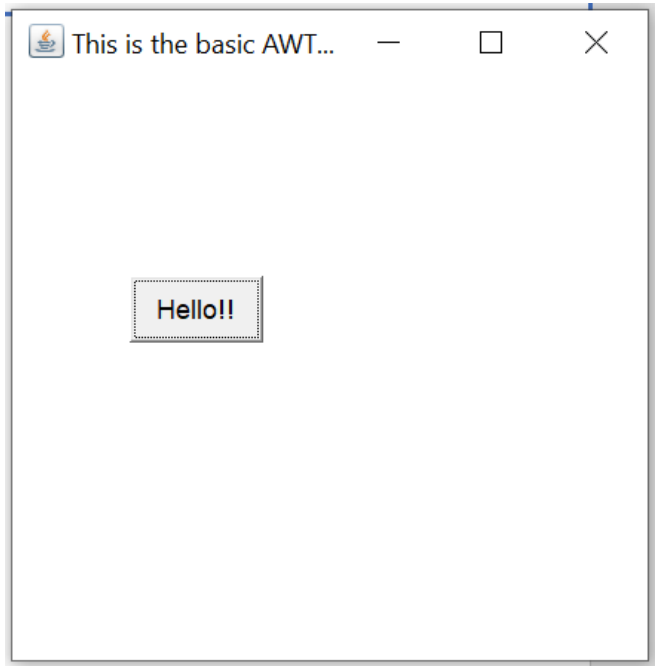
Java AWT Examples

To create simple AWT example, you need a frame. There are two ways to create a GUI using Frame in AWT.

1. By extending Frame class (inheritance)
2. By creating the object of Frame class (association)

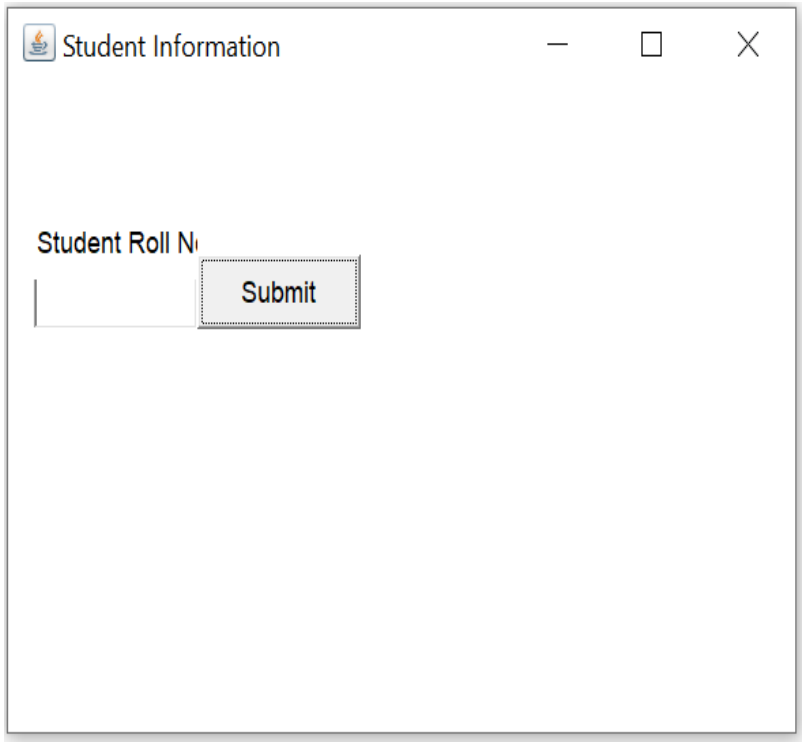
AWT Example by Inheritance


```
import java.awt.*;
public class AWTExample1 extends Frame {
    AWTExample1() {
        Button b = new Button("Hello!!");
        b.setBounds(60,120,60,30);
        add(b);
        setSize(300,300);
        setTitle("This is the basic AWT example");
        setLayout(null);
        setVisible(true);
    }
    public static void main(String args[]) {
        AWTExample1 f = new AWTExample1();
    }
}
```



AWT Example by Association


```
import java.awt.*;
class AWTExample2 {
    AWTExample2() {
        Frame f = new Frame();
        Label l = new Label("Student Roll No.:");
        Button b = new Button("Submit");
        TextField t = new TextField();
        l.setBounds(20, 80, 80, 30);
        t.setBounds(20, 100, 80, 30);
        b.setBounds(100, 100, 80, 30);
        f.add(b);
        f.add(l);
        f.add(t);
        f.setSize(400,300);
        f.setTitle("Student Information");
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[]) {
        AWTExample2 awt_obj = new AWTExample2();
    }
}
```



Introduction to Swing

- Swing is a Java Foundation Classes [JFC] library and an extension of the Abstract Window Toolkit [AWT].
- Swing offers much-improved functionality over AWT, new components, expanded components features, excellent event handling with drag and drop support.
- Swing has about four times the number of User Interface [UI] components as AWT and is part of the standard Java distribution.
- By today's application GUI requirements, AWT is a limited implementation, not quite capable of providing the components required for developing complex GUIs required in modern commercial applications.
- The AWT component set has quite a few bugs and really does take up a lot of system resources when compared to equivalent Swing resources.
- Netscape introduced its Internet Foundation Classes [IFC] library for use with Java.
- Its Classes became very popular with programmers creating GUIs for commercial applications.

Features of Swing

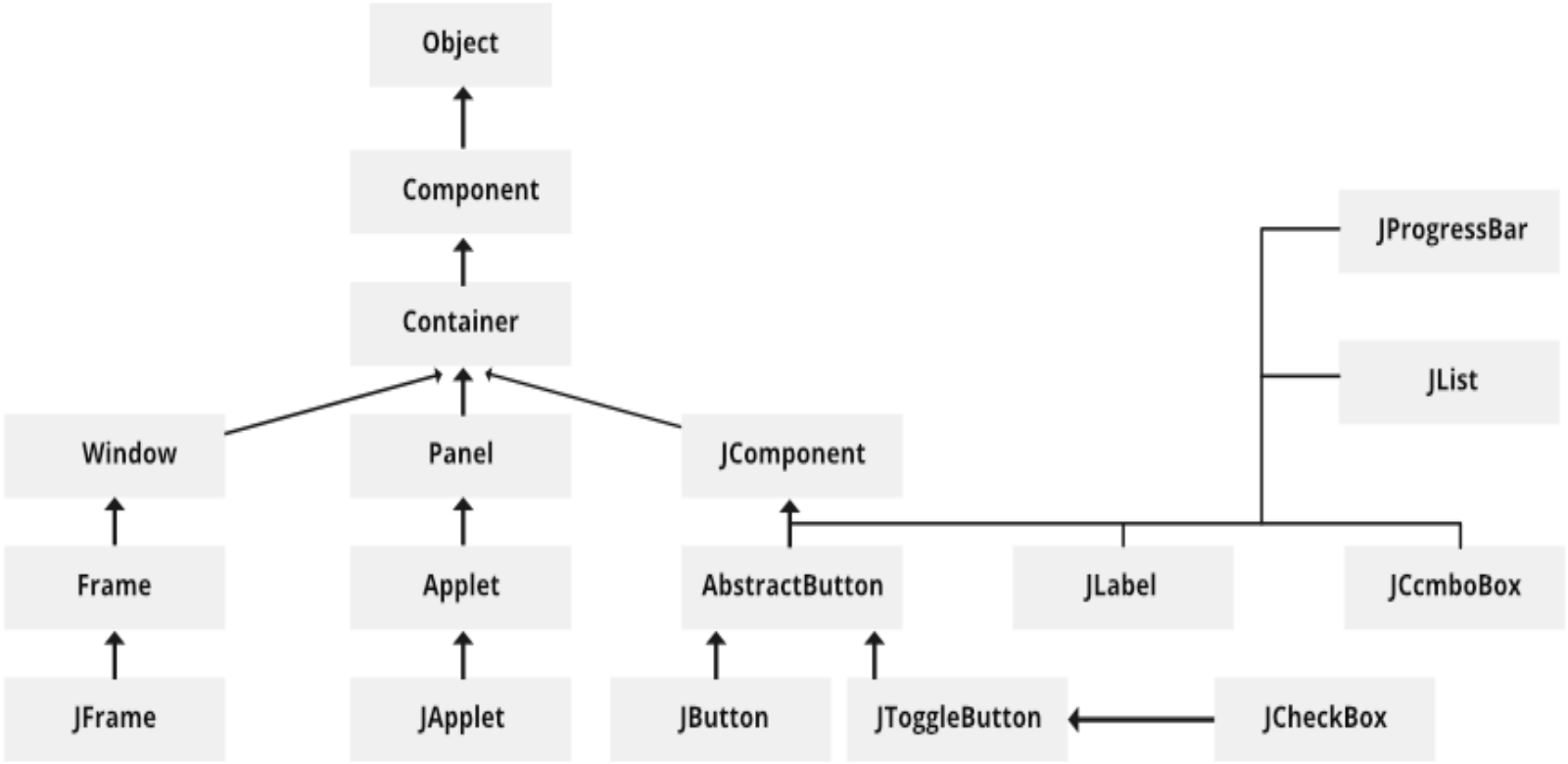
- Swing is a Set of API (API- Set of Classes and Interfaces).
- Swing is Provided to Design a Graphical User Interfaces.
- Swing is an Extension library to the AWT (Abstract Window Toolkit).
- Includes New and improved Components that have been enhancing the looks and Functionality of GUIs.
- Swing can be used to build (Develop) the Standalone swing GUI Apps, Servlets And Applets.
- It Employs model/view design architecture.
- Swing is more portable and more flexible than AWT, the Swing is built on top of the AWT.
- Swing is entirely written in Java.
- Java Swing Components are Platform-independent and the Swing Components are lightweight.
- Swing Supports Pluggable look and feels and Swing provides more powerful components such as tables, lists, Scrollpanes, Colourchooser, tabbedpane, etc.

Features of Swing Class

- Pluggable look and feel
- Uses MVC architecture
- Lightweight Components
- Platform Independent
- Advance features such as JTable, JTabbedPane, JScollPane etc
- Java is a platform-independent language and runs on any client machine, the GUI look and feel, owned and delivered by a platform specific O/S, simply does not affect an application's GUI constructed using Swing components.

- **Lightweight Components:** Starting with the JDK 1.1, its AWT supported lightweight component development. For a component to qualify as lightweight, it must not depend on any non-Java [O/s based) system classes. Swing components have their own view supported by Java's look and feel classes.
- **Pluggable Look and Feel:** This feature enables the user to switch the look and feel of Swing components without restarting an application. The Swing library supports components look and feel that remains the same across all platforms wherever the program runs. The Swing library provides an API that gives real flexibility in determining the look and feel of the GUI of an application

Swing Classes Hierarchy



https://www.geeksforgeeks.org/introduction-to-java-swing/

AWT vs Swing

Java AWT	Java Swing
AWT components are **heavyweight**.	Swing components are **lightweight**.
AWT **doesn't support pluggable look and feel**.	Swing **supports pluggable look and feel**.
AWT provides **less components** than Swing.	Swing provides **more powerful components** such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
AWT **doesn't follow MVC** (Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing **follows MVC**.

Java Swing Examples

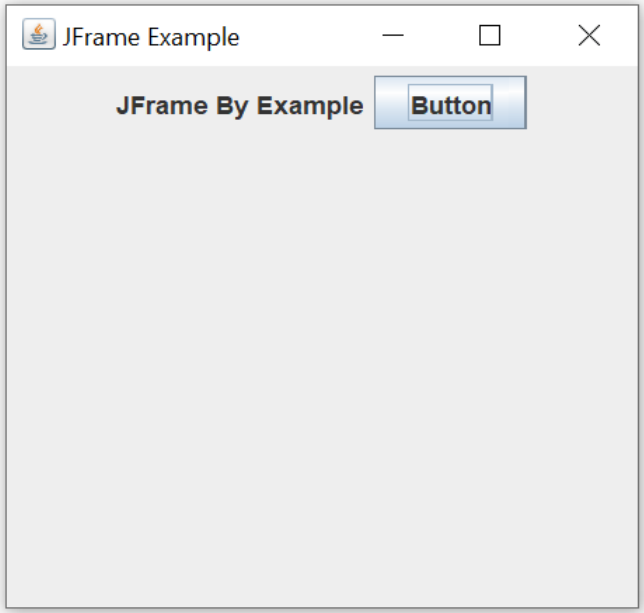
There are two ways to create a frame:

- By creating the object of Frame class (Association)
- By extending Frame class (Inheritance)

JFrame Example

Pune Institute of Computer Technology, Pune

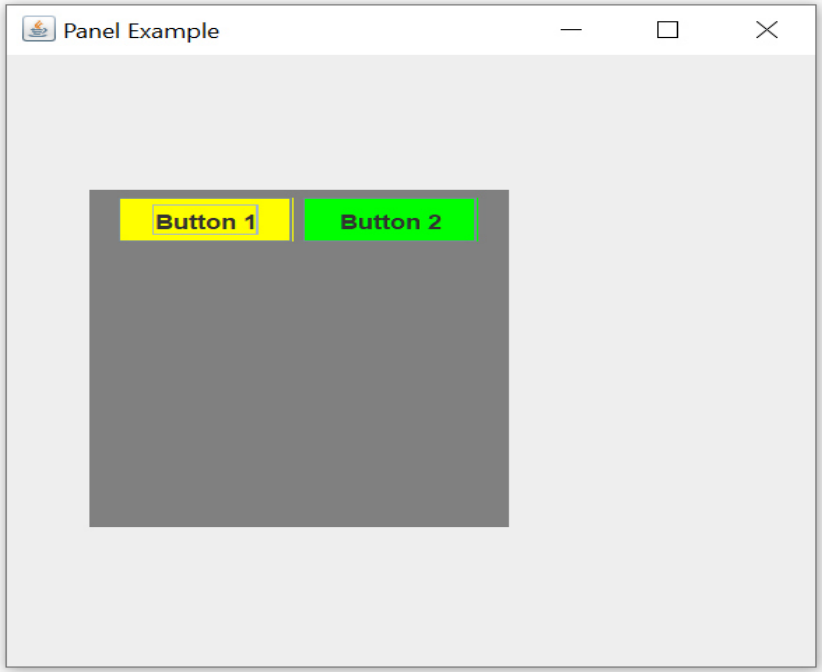
```
// A program to add a label and button in a frame

import java.awt.FlowLayout;
import javax.swing.*;

public class JFrameExample {
    public static void main(String s[]) {
        JFrame frame = new JFrame("JFrame Example");
        JPanel panel = new JPanel();
        panel.setLayout(new FlowLayout());
        JLabel label = new JLabel("JFrame By Example");
        JButton button = new JButton();
        button.setText("Button");
        panel.add(label);
        panel.add(button);
        frame.add(panel);
        frame.setSize(200, 300);
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```



JPanel Example

Pune Institute of Computer Technology, Pune

```
// A program to add panel to GUI

import java.awt.*;
import javax.swing.*;
public class PanelExample {
     PanelExample()  {
        JFrame f= new JFrame("Panel Example");
        JPanel panel=new JPanel();
        panel.setBounds(40,80,200,200);
        panel.setBackground(Color.gray);
        JButton b1=new JButton("Button 1");
        b1.setBounds(50,100,80,30);
        b1.setBackground(Color.yellow);
        JButton b2=new JButton("Button 2");
        b2.setBounds(100,100,80,30);
        b2.setBackground(Color.green);
        panel.add(b1);
        panel.add(b2);
        f.add(panel);
                f.setSize(400,400);
                f.setLayout(null);
                f.setVisible(true);
        }
        public static void main(String args[]) {
        new PanelExample();
    }
}
```

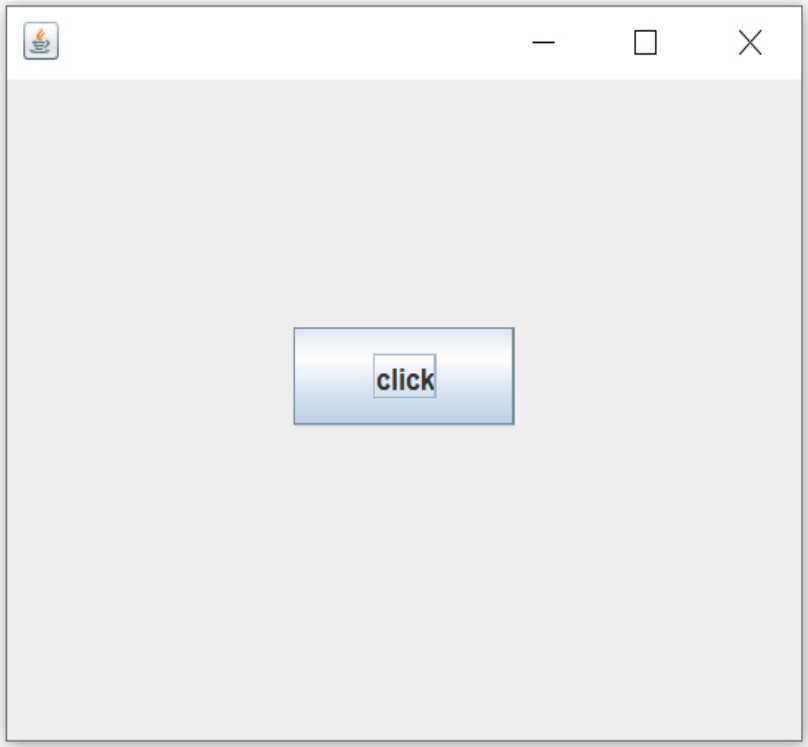


Swing Example


```java
//Create a Swing button

import javax.swing.*;
public class FirstSwingExample {
    public static void main(String[] args) {
        //creating instance of JFrame
        JFrame f=new JFrame();
                //creating instance of JButton
        JButton b=new JButton("click");
        //x axis, y axis, width, height
        b.setBounds(130,100,100, 40);
        f.add(b);//adding button in JFrame
        f.setSize(400,500);//400 width and 500 height
        f.setLayout(null);//using no layout managers
        f.setVisible(true);//making the frame visible
    }
}
```

JFrame and JButton in Constructor

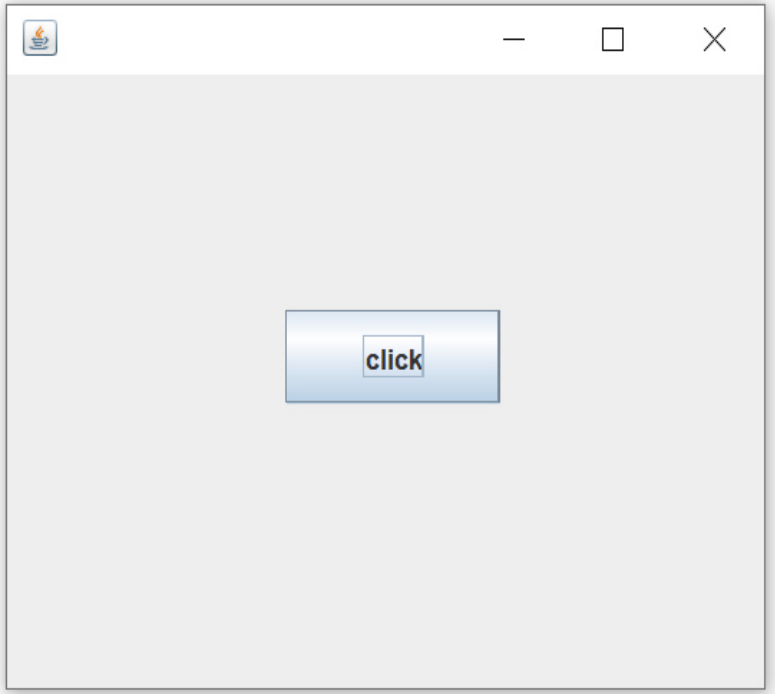

```
// Creating JFrame, JButton and method call inside the java constructor.

import javax.swing.*;
public class Simple {
    JFrame f;
    Simple(){
    f=new JFrame();
    JButton b=new JButton("click");
    b.setBounds(130,100,100, 40);
    f.add(b);
    f.setSize(400,500);
    f.setLayout(null);
    f.setVisible(true);
}
    public static void main(String[] args) {
        new Simple();
    }
}
```



Inheriting JFrame Class

Pune Institute of Computer Technology, Pune

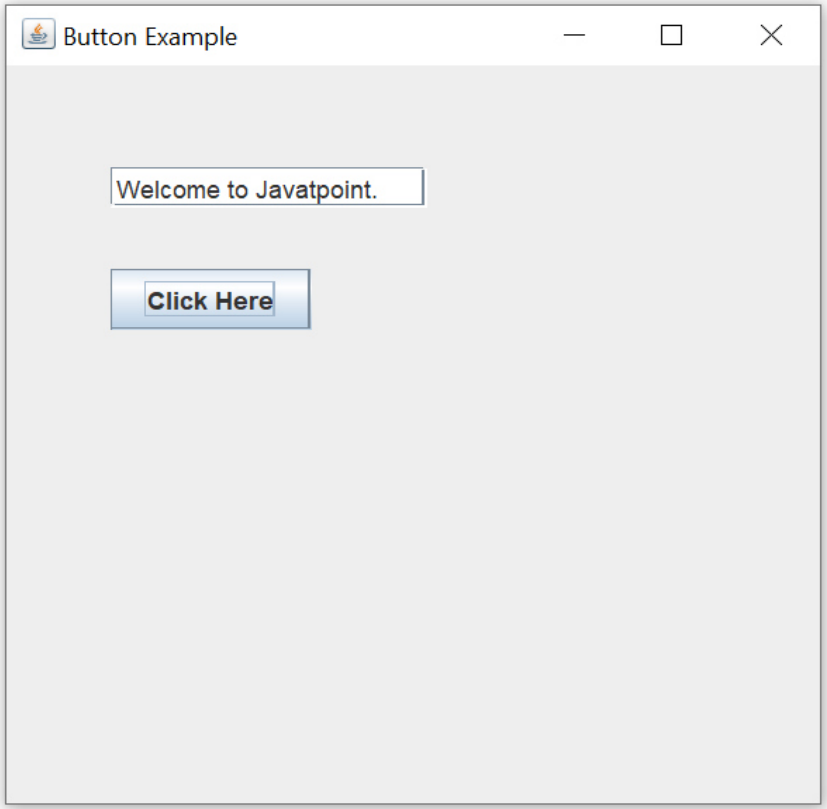
```
// Inherit the JFrame class, so there is no need to create the
// instance of JFrame class explicitly.

import javax.swing.*;
//inheriting JFrame
public class Simple2 extends JFrame{
    JFrame f;
    Simple2(){
        JButton b=new JButton("click");
        b.setBounds(130,100,100, 40);
        add(b);
        setSize(400,500);
           setLayout(null);
        setVisible(true);
    }
    public static void main(String[] args) {
        new Simple2();
    }
}
```



JButton Example

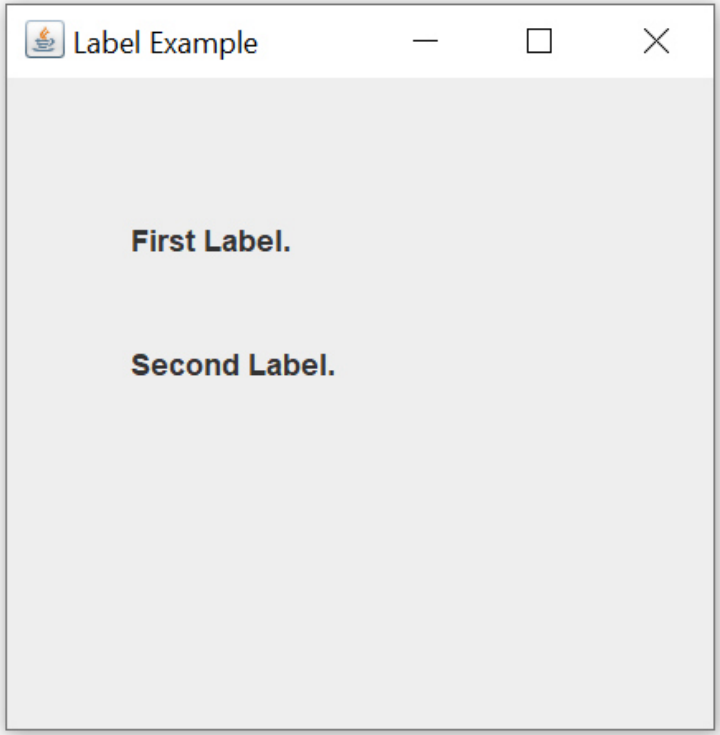
Pune Institute of Computer Technology, Pune

```
// Button with Action Listener
import java.awt.event.*;
import javax.swing.*;

public class ButtonExample {
public static void main(String[] args) {
    JFrame f=new JFrame("Button Example");
    final JTextField tf=new JTextField();
    tf.setBounds(50,50, 150,20);
    JButton b=new JButton("Click Here");
    b.setBounds(50,100,95,30);
    b.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e){
            tf.setText("Welcome to Javatpoint.");
        }
    });
    f.add(b);f.add(tf);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
}
```



JLabel Example


```
import javax.swing.*;
class LabelExample
{
public static void main(String args[]){
    JFrame f= new JFrame("Label Example");
    JLabel l1,l2;
    l1=new JLabel("First Label.");
    l1.setBounds(50,50, 100,30);
    l2=new JLabel("Second Label.");
    l2.setBounds(50,100, 100,30);
    f.add(l1); f.add(l2);
    f.setSize(300,300);
    f.setLayout(null);
    f.setVisible(true);
    }
}
```



Drawing Geometrical Shapes


```
import java.awt.*;
import java.awt.geom.*;
import javax.swing.*;

public class Shapes extends JPanel {
    public static void main(String[] args) {
        JFrame f = new JFrame("Experiment 11");
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.getContentPane().add(new Shapes());
        f.setSize(290, 325);
        f.setLocation(550, 25);
        f.setVisible(true);
    }
    public void paint(Graphics g) {
        g.setColor(Color.RED);
        g.drawLine(150, 50, 90, 50);
        Graphics2D g2 = (Graphics2D) g;
        g2.setColor(Color.GREEN);
        g2.drawRect(60, 100, 180, 60);
        Graphics2D g3 = (Graphics2D) g;
        g2.setColor(Color.BLUE);
        g3.drawOval(60, 200, 180, 60);
    }
}
```

File Handling

- In Java, file handling is done with File Class.
- The File Class is inside the **java.io** package. The File class can be used by creating an object of the class and then specifying the name of the file.
- File Handling is an integral part of any programming language as file handling enables us to store the output of any program in a file and allows us to perform certain operations on it.
- In simple words, file handling means reading and writing data to a file.

Streams in Java

- In Java, a sequence of data is known as a stream.
- This concept is used to perform I/O operations on a file.
- There are two types of streams :
  - Input Stream
  - Output Stream

Input Stream

- The Java InputStream class is the superclass of all input streams.
- The input stream is used to read data from numerous input devices like the keyboard, network, etc.
- InputStream is an abstract class, its subclasses are used to read data.
- There are several subclasses of the InputStream class, which are:
  - AudioInputStream
  - ByteArrayInputStream
  - FileInputStream
  - FilterInputStream
  - StringBufferInputStream
  - ObjectInputStream

Creating an InputStream

InputStream obj = new FileInputStream();

We can create an input stream from other subclasses as well as InputStream.

Methods of InputStream

Method	Description
read()	Reads one byte of data from the input stream.
read(byte[] array)()	Reads byte from the stream and stores that byte in the specified array.
mark()	It marks the position in the input stream until the data has been read.
available()	Returns the number of bytes available in the input stream.
markSupported()	It checks if the mark() method and the reset() method is supported in the stream.
reset()	Returns the control to the point where the mark was set inside the stream.
skips()	Skips and removes a particular number of bytes from the input stream.
close()	Closes the input stream.

Output Stream

- The output stream is used to write data to numerous output devices like the monitor, file, etc.
- OutputStream is an abstract superclass that represents an output stream.
- OutputStream is an abstract class, its subclasses are used to write data.
- There are several subclasses of the OutputStream class:
  - ByteArrayOutputStream
  - FileOutputStream
  - StringBufferOutputStream
  - ObjectOutputStream
  - DataOutputStream
  - PrintStream

Creating an OutputStream

OutputStream obj = new FileOutputStream();

We can create an output stream from other subclasses as well as OutputStream.

Methods of OutputStream

Method	Description
write()	Writes the specified byte to the output stream.
write(byte[] array)	Writes the bytes which are inside a specific array to the output stream.
close()	Closes the output stream.
flush()	Forces to write all the data present in an output stream to the destination.

Byte Stream

This stream is used to read or write byte data. The byte stream is again subdivided into two types which are as follows:

- Byte Input Stream: Used to read byte data from different devices.
- Byte Output Stream: Used to write byte data to different devices.

Character Stream

This stream is used to read or write character data. Character stream is again subdivided into 2 types which are as follows:

- Character Input Stream: Used to read character data from different devices.
- Character Output Stream: Used to write character data to different devices.

Java File Class Methods

Method Name	Description	Return Type
canRead()	It tests whether the file is readable or not.	Boolean
canWrite()	It tests whether the file is writable or not.	Boolean
createNewFile()	It creates an empty file.	Boolean
delete()	It deletes a file.	Boolean
exists()	It tests whether the file exists or not.	Boolean
length()	Returns the size of the file in bytes.	Long
getName()	Returns the name of the file.	String
list()	Returns an array of the files in the directory.	String[]
mkdir()	Creates a new directory.	Boolean
getAbsolutePath()	Returns the absolute pathname of the file.	String

File Operations in Java

The following are the several operations that can be performed on a file in Java :

- Create a File
- Read from a File
- Write to a File
- Delete a File

Create a File

- In order to create a file in Java, the createNewFile() method can be used.
- If the file is successfully created, it will return a Boolean value true and false if the file already exists.


```java
// Import the File class
import java.io.File;
// Import the IOException class to handle errors
import java.io.IOException;

public class Create_file {
    public static void main(String[] args)
    {

        try {
            File Obj = new File("myfile.txt");
            if (Obj.createNewFile()) {
                System.out.println("File created: "+ Obj.getName());
            }
            else {
                System.out.println("File already exists.");
            }
        }
        catch (IOException e) {
            System.out.println("An error has occurred.");
            e.printStackTrace();
        }
    }
}
```

Read from a File


```java
// Import the File class
import java.io.File;
// Import this class for handling errors
import java.io.FileNotFoundException;
// Import the Scanner class to read content from text files
import java.util.Scanner;

public class Read_file {
    public static void main(String[] args)
    {
        try {
            File Obj = new File("Exp6.txt");
            Scanner Reader = new Scanner(Obj);
            while (Reader.hasNextLine()) {
                String data = Reader.nextLine();
                System.out.println(data);
            }
            Reader.close();
        }
        catch (FileNotFoundException e) {
            System.out.println("An error has occurred.");
            e.printStackTrace();
        }
    }
}
```

Write to a File

- Use of FileWriter class along with its write() method in order to write some text to the file.

```
import java.io.FileWriter;   // Import the FileWriter class
import java.io.IOException;  // Import the IOException class for handling errors
public class Write_file {
    public static void main(String[] args)
    {
        try {
            FileWriter Writer
                = new FileWriter("Test.txt");
            Writer.write("Fundamentals of JAVA Programming");
            Writer.close();
            System.out.println("Successfully written.");
        }
        catch (IOException e) {
            System.out.println("An error has occurred.");
            e.printStackTrace();
        }
    }
}
```

Delete a File

- Use of delete() method in order to delete a file

```
// Import the File class
import java.io.File;
public class Delete_file {
    public static void main(String[] args)
    {
        File Obj = new File("Try.txt");
        if (Obj.delete()) {
            System.out.println("The deleted file is : " + Obj.getName());
        }
        else {
            System.out.println(
                "Failed in deleting the file.");
        }
    }
}
```

Random Access Files in Java

- The class *RandomAccessFile* is used for reading and writing to random access file.
- A random-access file behaves like a large array of bytes.
- There is a cursor implied to the array called file pointer, by moving the cursor to read write operations.
- If end-of-file is reached before the desired number of byte has been read than EOFException is thrown.
- It is a type of IOException.

Constructor

Constructor	Description
RandomAccessFile (File file, String mode)	Creates a random access file stream to read from, and optionally to write to, the file specified by the File argument.
RandomAccessFile (String name, String mode)	Creates a random access file stream to read from, and optionally to write to, a file with the specified name.

Methods

Modifier and Type	Method	Method
void	close()	It closes this random access file stream and releases any system resources associated with the stream.
FileChannel	getChannel()	It returns the unique FileChannel object associated with this file.
int	readInt()	It reads a signed 32-bit integer from this file.
String	readUTF()	It reads in a string from this file.
void	seek(long pos)	It sets the file-pointer offset, measured from the beginning of this file, at which the next read or write occurs.
void	writeDouble (double v)	It converts the double argument to a long using the doubleToLongBits method in class Double, and then writes that long value to the file as an eight-byte quantity, high byte first.
void	writeFloat(floa t v)	It converts the float argument to an int using the floatToIntBits method in class Float, and then writes that int value to the file as a four-byte quantity, high byte first.
void	write(int b)	It writes the specified byte to this file.
int	read()	It reads a byte of data from this file.
long	length()	It returns the length of this file.
void	seek(long pos)	It sets the file-pointer offset, measured from the beginning of this file, at which the next read or write occurs.

References

- E Balagurusamy, “Programming with JAVA”, Tata McGraw Hill, 6th Edition.
- Herbert Schildt, “Java: The complete reference”, Tata McGraw Hill, 7th Edition.
- https://www.tutorialspoint.com
- https://www.geeksforgeeks.org
- https://www.javatpoint.com/
- Online Resources