Unit I

1 Foundations of Object Oriented Programming

1.1 : Software Evolution

Q.1 What is software evolution ? Explain.

Ans. :

- Software evolution means changes taking place in software tools, techniques and practices.
- During software evolution process different programming approaches are adopted. Following Fig. Q.1.1 represents layers of computer software.

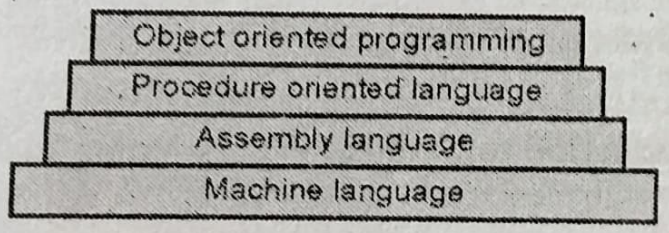


Fig. Q.1.1 Programming approaches and software evolution

- In early years Assembly language was the only programming language used by programmer. This is also called as low level programming language.
- In 1980, the practice of structured programming was followed by the invent of C. The C is also called as high level programming language.
- The object oriented programming approach eliminated the drawbacks of procedural programming.

The C++ and Java are popular object oriented programming languages.

1.2 : Introduction to Procedural Programming Technique

Q.2 What are the merits and demerits of procedural programming language ?

Ans. : Merits :

1. Simple to implement.
2. These languages have low memory utilization.

Demerits :

1. Large complex problems can not be implemented using this category of language.
2. Parallel programming is not possible in this language.
3. This language is less productive and at low level compared to other programming languages.

1.3 : Introduction to Modular Programming Technique

Q.3 Explain modular programming technique.

Ans. :

- Modular programming is the process of subdividing a computer program into separate sub-programs.
- A module is a separate software component. It can often be used in a variety of applications and functions with other components of the system.
- Similar functions are grouped in the same unit of programming code and separate functions are developed as separate units of code so that the code can be reused by other applications.

1.4 : Introduction to Object Oriented Programming Technique

Q.4 Give the difference between procedure oriented and object oriented programming.

Ans. :

Sr. No.	Procedural Programming Language	Object Oriented Programming Language (OOP)
1.	The procedural programming executes **series of procedures** sequentially.	In object oriented programming approach there is a **collection of objects**.
2.	This is a **top down** programming approach.	This is a **bottom up** programming approach.
3.	The major focus is on **procedures** or **functions.**	The main focus is on **objects.**
4.	Data reusability is **not possible.**	**Data reusability** is one of the important feature of OOP.
5.	Data hiding is **not possible.**	**Data hiding** can be done by making it private.
6.	It is **simple** to implement.	It is **complex** to implement.
7.	For example : C, Fortran, COBOL	For example : C++, JAVA

1.5 : Introduction to Generic Programming Technique

Q.5 Explain generic programming technique and given the merits and demerits of it.

Ans. :

- **Definition :** Generic programming is an approach in which algorithms are written in terms of type to be specified later. Types are then instantiated when needed for specific types provided as parameters.

- In the language like ADA, the generic programming approach is used.
- Similarly, in C++ the Standard Template Library (STL) allows us to adopt the general programming approach.

Merits

(1) The abstract code can be written using this approach, which can be used to serve any data type element.

(2) Generic programming paradigm is an approach to software decomposition.

Demerits

(1) The syntax is complicated.

(2) It is complex to implement.

(3) The extra instantiations generated by templates can also cause the difficulty for the debuggers to debug the program.

1.6 : Limitations of Procedural Programming and Need of OOP

Q.6 Explain the need of OOP.

Ans. : • Following are some drawbacks of OOP -

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

- To overcome these drawbacks the object oriented programming came into existence.

1.7 : Fundamentals of Object Oriented Programming

Q.7 State and explain various characteristics of OOP.

Ans. : • Various characteristics of object oriented programming are -

1. Object

- Object is an instance of a class.
- In C++ the class variables are called objects. Using objects we can access the member variable and member function of a class.
- Declaring objects -

The syntax for declaring object is –

```
Class_Name Object_Name;
```

- Example

```
Fruit f1;
```

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

2. Classes

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

```
class name_of_class
{
    private :
        variables declarations;
        function declarations;
    public :
        variable declarations;
        function declarations;
}; ← do not forget semicolon
```

- **Example**

```
class rectangle
```

```
{
    private :
        int len, br;
    public :
        void get_data ( ) ;
        void area( );
        void print_data ( );
};
```

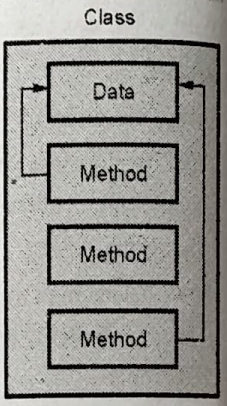


Fig. Q.7.1 : Concept of encapsulation

3. Data Encapsulation

- **Definition :** Encapsulation means binding of data and method together in a **single entity called class**.
- The data inside that class is accessible by the function in the same class. It is normally not accessible from the outside of the component.

4. Inheritance

- **Definition :** Inheritance is a property by which the new classes are created using the old classes. In other words the new classes can be developed using some of the properties of old classes.
- The old classes are referred as **base classes** and the new classes are referred as **derived classes.** That means the derived classes inherit the properties (data and functions) of base class.
- **Example :**

Here the **Shape** is a base class from which the **Circle, Line** and **Rectangle** are the derived classes. These classes inherit the functionality draw() and resize(). Similarly the Rectangle is a base class for the derived class Square. Along with the derived properties the derived class can have its own properties. For example the class Circle may have the function like backgrcolor() for defining the back ground color.

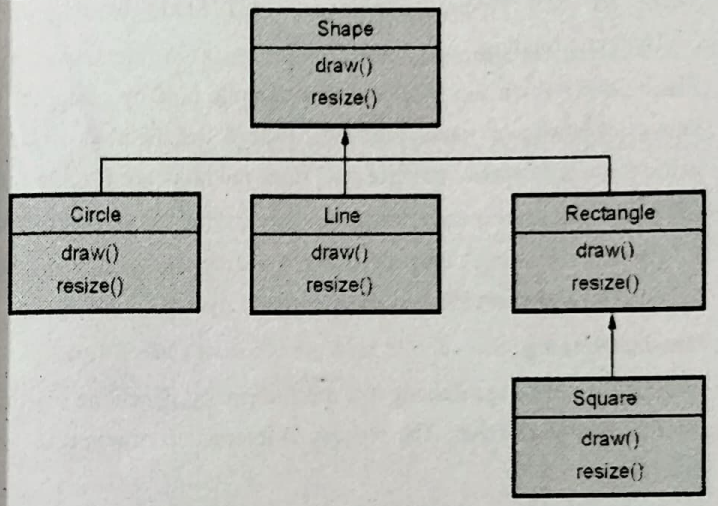


Fig. Q.7.2 : Hierarchical structure of inheritance

5. Polymorphism

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

6. Static and Dynamic Binding

- Connecting a method call to method body is called **binding**.

- There are two types of bindings - (1) Static binding and (2) Dynamic binding.
- The binding which can be resolved at compile time by compiler is known as **static or early binding**. In Java the methods written using keywords **static, private** and **final** methods are resolved at compile-time.
- In Dynamic binding compiler doesn't decide the method to be called. **Method overriding** is an example of dynamic binding.

7. **Message Passing**

- **Definition :** Message passing is a mechanism by which the objects interact with each other. The process of interaction of objects is as given below -
  1. Define the class with data members and member functions.
  2. Create the objects belonging to the class.
  3. Establish the communication between these objects using the methods.
- Object send information to each other using the methods. This process is called as **message passing**.

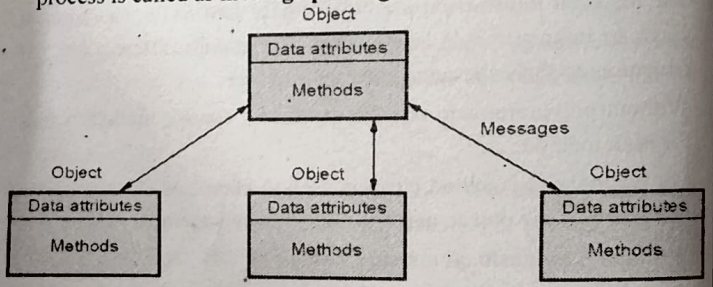


Fig. Q.7.3 : Methods and messages

END...

Unit II

2 Classes, Objects and Methods

2.1 : Creating a Class and Object

Q.1 Define a class item having data member code and price. Accept data for one object and display it.

Ans. :

MyClass.java

```
import java.io.*;
import java.lang.*;
import java.math.*;
public class MyClass
{
 int code;
 double price;
 public void display(int c,double p)
 {
  System.out.println();
  System.out.println("Code: "+c);
  System.out.println("Price: "+p);
 }
 public static void main(String args[])
 {
  MyClass obj1=new MyClass();
  obj1.display(101,76.50);
 }
}
```

Q.2 Write a program to create a class account having variable accno, accname and balance. Define deposite ( ) and withdraw ( ) methods. Create one object of class and perform the operation.

Ans. :

```
public class Account
{
    private double balance;
    public void initialize()
    {
     balance=1000;
    }
     public void deposit(double amount)
    {
     balance = balance + amount;
    }

    public void withdraw(double amount)
    {
     balance = balance - amount;
    }
    public double getBalance()
    {
     return balance;
    }
    public static void main(String args[])
    {
     Account ac=new Account();
     ac.initialize();
     System.out.println("Depositing some amount");
     ac.deposit(100);
     System.out.println("Total Balance is: ");
     ac.balance();
     System.out.println("Withdrawing some amount");
     ac.withdraw(200);
     System.out.println("Total Balance is: ");
     ac.balance();
    }
}
```


2.2 : Visibility / Access Modifiers and Encapsulation

Q.3 What is access modifiers ?

Ans. :

- Java has three commonly used access specifiers – public, private and protected. Java also defines default access level.
- The **protected** access specifier is used only during inheritance.
- **Public** allows classes, methods and data fields accessible from any class.
- **Private** allows classes, methods and data fields accessible only from within own class.
- If public or private access is not mentioned then that member uses default access which is just similar to public.

```
class AccessControl
{
    int a;
    public int b;
    private int c;

    void fun(int val)
    {
     c=val;
    }
    void display()
    {
     System.out.println("c = "+c);
    }
}
class test
{
    public static void main(String args[])
    {
     AccessControl obj=new AccessControl();
     obj.a=100; //Valid
     obj.b=200; //Valid
```

```
    obj.c=300; //Error: Private access
    obj.fun(300); //Valid
    obj.display(); //Valid
    }
}
```

2.3 : Defining and Calling Method

Q.4 Write a Java program for finding minimum of two numbers using function.

Ans. :

```
import java.util.*;
classMinValueDemo
{
    public static int MinValue(int x,int y)
    {
      if(x < y)
        return x;
      else
        return y;
    }
    public static void main(String args[])
    {
      Scanner input = new Scanner(System.in);
      System.out.println("Enter value of first number: ");
      int a = input.nextInt();
      System.out.println("Enter value of second number: ");
      int b = input.nextInt();
      int c = MinValue(a,b);
      System.out.println("The minimum value is: "+c);
    }
}
```

Method definition

Method call

Output

```
Enter value of first number:
10
Enter value of second number:
20
The minimum value is: 10
```


Q.5 Explain call by value and call by reference.

Ans. : • Parameter can be passed to the function by two ways -

1. Call by value
2. Call by reference

- In the **call by value** method the value of the actual argument is assigned to the formal parameter. If any change is made in the formal parameter in the subroutine definition then that change does not reflect the actual parameters.
- Following Java program shows the use of parameter passing by value -

Java Program

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

Output

```
The values of a and b before function call
a= 10
b= 20
The values of a and b after function call
a = 10
b= 20
```

2.4 : Reading Input from Keyboard

Q.6 How to read an integer value from keyboard ?

Ans. : Normally, when user types 123 on the console using the keyboard, it is interpreted as "123" that is string 123 and not the integer value 123. Therefore we need to convert the string to int value. Following are the steps to do so.

Step 1 : Create a Scanner using following code

```
Scanner scanner=new Scanner(System.in);
```

Step 2 : Prompt the user to type something

```
System.out.println("Enter Some integer value...");
```

Step 3 : Read the line of text entered by the user.

```
String input=scanner.nextLine();
```

Step 4 : Convert the string to int value. Use the **Integer** class to parse the string of characters into an integer.

```
int number = Integer.parseInt( input );
```

2.5 : The 'this' Keyword

Q.7 Write a short note on - this keyword.

Ans. :

- The **this** keyword is used by a method to refer the object which invoked that method.
- this keyword is used inside the method definition **to refer the current object**. That means **this** is a reference to the object which is used to invoke it.

- Following is a Java program in which the method uses the keyword **this.**
- **Example Program**

```
import java.io.*;
import java.lang.*;

public class thisDemo
{
 int a,b;
 public void Sum(int a,int b)
 {
  this.a=a;
  this.b=b;
 }
 public void display()
 {
  int c=a+b;
  System.out.println("a= "+a);
  System.out.println("b= "+b);
  System.out.println("Sum= "+c);
  System.out.println();
 }
 public static void main(String args[])
{
 thisDemo obj1=new thisDemo();
 thisDemo obj2=new thisDemo();
  obj1.Sum(10,20);
  obj1.display();
  obj2.Sum(40,50);
  obj2.display();
}
}
```

Output

```
a= 10
b= 20
Sum= 30

a= 40
b= 50
Sum= 90
```

2.6 : Method Overloading

Q.8 What is method overloading ?

Ans. :

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

For example :

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

That means, by overloading mechanism, we can handle **different number of parameters** or **different types** of parameter by having the same method name. Following Java program explain the concept overloading –

Java Program[OverloadingDemo.java]

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



```
  ans=num1+num2;
  System.out.println(ans);
}
public static void Sum(int num1,int num2,int num3)
{
  int ans;
  ans=num1+num2+num3;
  System.out.println(ans);
 }
}
```

Output

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

F:\test>java OverloadingDemo
Sum of two integers
30
Sum of two double numbers
30.9
Sum of three integers
60
```

2.7 : Using Object as a Parameters

Q.9 Can we pass object as parameter ? If yes, justify.

Ans. :

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

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

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

Java Program[ObjDemo.java]

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

 }
}
```

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

Method with object as parameter

2.8 : Returning Object

Q.10 How to return an object from a method ?

Ans. : • We can return an object from a method. The data type suc method is a class type.

Java Program[ObjRetDemo.java]

```
public class ObjRetDemo {
        int a;
        ObjRetDemo(int val)
        {
                a=val;
        }
        ObjRetDemo fun()
        {
```

Data type of this method is name of the class


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

}
}
```

2.9 : Array of Objects

Q.11 Write a Java program to demonstrate the use of array of objects.

Ans. :

```
import java.util.*;
class Student
{
        int rollno;
        String name;
        public void set_data()
        {
                Scanner input = new Scanner(System.in);
                System.out.println("Enter roll number: ");
                rollno = input.nextInt();
                System.out.println("Enter name: ");
                name = input.next();
        }
        public void display_data()
        {
                System.out.println(rollno+"\t"+name);
        }

class StudentInfo
```

Creating a class **Student**

```
{
    public static void main(String args[])
    {
        Student[] obj=new Student[10];
        System.out.println("How many records are there?");
        Scanner input = new Scanner(System.in);
        int n=input.nextInt();
        for(int i=0;i<n;i++)
        {
            obj[i]=new Student();
            obj[i].set_data();
        }
        System.out.println("The Student Records are ...");
        System.out.println("----------");
        System.out.println("RollNo\tName");
        System.out.println("----------");
        for(int i=0;i<n;i++)
        {
            obj[i].display_data();
        }
    }
}
```

Declaring an array of objects for Student class

Creating an array of objects for Student class

2.10 : Memory Allocation : 'new' and Memory Recovery : 'delete'

Q.12 Explain new and delete operators in detail.

Ans. : The memory can be allocated for an object using the keyword new. Using the keyword new we can allocate the memory for an object of a class.

Syntax

```
class_name object_name = new class_name()
```

For example

```
Test obj = new Test();
```

- When memory allocated by new keyword is no longer required, it is freed using delete operator.

- To deallocate dynamic memory, C++ provides an operator **delete** that is not found in Java. Although Java has a reserved name as keyword **delete.**
- **Java** doesn't require a delete operator because it has an **automatic garbage collector** that returns discarded data to the heap.
- C++ does not have a garbage collector so programmers must explicitly deallocate memory with delete.
- **Syntax**

```
delete object_name
```

- **Example**

```
delete obj;
```

2.11 : Static Data Members and Static Methods

Q.13 Explain about static variables and static methods in Java.

Ans. : The static members can be **static data member** or **static method.** The static members are those members which can be accessed **without using object.**

Following program illustrates the use of static data members and static methods.

Java Program[StaticProg.java]

```
/*****************************************************************
Program to introduce the use of the static method and static variables
*****************************************************************/
class StaticProg
{
 static int a=10;
 static void fun(int b)
 {

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

```
 }
}
class AnotherClass
{
 public static void main(String[] args)
 {
  System.out.println("a= "+StaticProg.a);
  StaticProg.fun(20);

 }
}
```

Output

```
a= 10
b= 20
```

2.12 : Forward Declaration

Q.14 What is forward declaration ? Is it used in Java ?

Ans. : • Forward declaration means declaration of class name, metho or variables prior to its implementation.

- Such declaration is required in C++ programming but this is not th case in Java.

2.13 : Class as Abstract Data Types (ADTs)

Q.15 What is abstract data type ? Can we represent a class a ADT ?

Ans. :

- Abstract Data type is a specification in which implementatio details are hidden. The ADT specifies what is to be done but hid how is to be done.
- An ADT does not specify how the data type is implemented. Thes implementation details are hidden from the user of the ADT an

protected from outside access. This concept is also referred to as **Encapsulation.**

- The class is an entity which contains the data members and member functions together as a single unit.
- Java class can be defined as follows -

```
class Customer {
      int ID;
      int Age;
      String Name;
      Customer() {
      }
      Customer(int ID) {
      }
      double withdraw_money() {
      ...
      ...
}
```

2.14 : Classes as Objects

Q.16 What is object class ?

Ans. : • In Java there is a special class named **Object.**

- Object is a superclass or parent class of all other classes by default.
- The object can be obtained using **getObject()** method.
- For example -

```
Object obj=getObject();
```

END...

Unit III

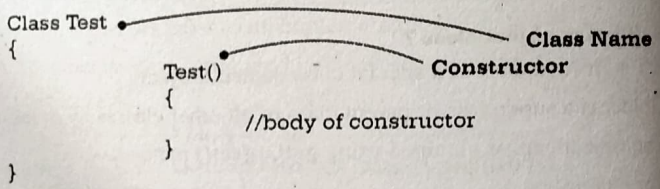
3 Constructors and Destructors

3.1 : Use of Constructor

Q.1 What is constructor ?

Ans. :

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



3.2 : Characteristics of Constructor

Q.2 Enlist any four characteristics of constructor.

Ans. :

1. Name of constructor must be the same as the name of the class for which it is being used.
2. The constructor must be declared in the **public** mode.
3. The constructor gets invoked automatically when an object gets created.

4. The constructor should not have any return type. Even a **void** data type should not be written for the constructor.

3.3 : Types of Constructor

Q.3 What are the types of constructor ? ☞ [SPPU : June-22, Marks 3]

Ans. : There are two types of constructor - (1) No argument constructor and (2) Parameterized constructor

Q.4 Explain the concept of parameterized constructor with an illustrative example. ☞ [SPPU : June-22, Marks 6]

Ans. : Parameterized constructor is a constructor in which the parameters are passed to the constructor function.

Example Program

```
class Person
{
   int id;
   String city;
   Person(int i,String n)
    {
      id=i;
      name=n;
    }
   void display()
   {
    System.out.print("ID: "+id);
    System.out.println(" Name: "+name);
   }
   public static void main(String args[])
   {
    Person p1=new Person(10,"Ankita");
    Person p2=new Person(20,"Prajka");
    p1.display();
    p2.display();
   }
}
```

Output

```
ID: 10 Name: Ankita
ID: 20 Name: Prajka
```

3.4 : Constructor Overloading

Q.5 What is constructor overloading ? Explain.

Ans. :

- Constructor overloading in Java allows to have more than o constructor inside one Class.
- Multiple constructor with different signatures is called **overloaded constructor.**
- Following is a simple Java program that illustrates the concept constructor overloading.
- **Example Program**

```
public class Rectangle2
{
    int height,width;
    double ht,wd;
    Rectangle2(int h,int w)  //constructor with two integer //values
    {
     height=h;
     width=w;
    }
    Rectangle2(double h,double w) //constructor with two
    //double values
    {
     ht=h;
     wd=w;
    }
    Rectangle2(int val)  //constructor with single integer value
    {
     height=val;
    }
    void area1()
    {
     System.out.println("Now, The function is called...");
     int result=height*width;
     System.out.println("The area is "+result);
    }
    void area2()
```



```
    {
     System.out.println("Now, The function is called...");
     double result=ht*wd;
     System.out.println("The area is "+result);
    }
    void area3()
    {
     System.out.println("Now, The function is called...");
     int result=height*height;
     System.out.println("The area is "+result);
    }
}
class OverLoadConstr
{
public static void main(String args[])
 {
    Rectangle2 obj1=new Rectangle2(11,20);
    obj1.area1(); //call to the method
    Rectangle2 obj2=new Rectangle2(11.33,20.22);
    obj2.area2(); //call to the method
    Rectangle2 obj3=new Rectangle2(10);
    obj3.area3(); //call to the method
 }
}
```

Output

```
Now, The function is called...
The area is 220
Now, The function is called...
The area is 229.09259999999998
Now, The function is called...
The area is 100
```

3.5 : Dynamic Initialization of Objects

Q.6 In Java, dynamic initialization of objects is possible. Justify.

Ans. : Dynamic initialization of object means initializing the data members of class while creating the object. That means when we provide initial values to the data members of the class during creation of the object – we dynamically initialize the objects.

Following program illustrates the initialization of two objects.

Java Program

```
class Addition
{
    private int a;
    private int b;
    public Addition(int x,int y)
    {
        a=x;
        b=y;
    }
    public void display()
    {
        System.out.println("a = "+a);
        System.out.println("b = "+b);
    }
}
class Main {
    public static void main(String[] args) {
        Addition add1=new Addition(10,20);
        System.out.println("Initialization of first object");
        add1.display(); //output : a = 10, b = 20
        System.out.println("Initialization of second object");
        Addition add2=new Addition(100,200);
        add2.display(); //output : a = 100, b = 200
    }
}
```

3.6 : Constructor with Default Arguments

Q.7 Can we pass default argument to a constructor in Java ? Explain how.

Ans. : The constructor can be defined by passing the default arguments to it.

However, in C++ we can implement constructor with default arguments. Consider following example which illustrates the use of default argument constructor.

C++ Program

```
#include <iostream>
using namespace std;

class Test
{
private:
    int a;
    int b;
    int c;
public :
    Test(int x=10,int y=20); //declaration of constructor with default arguments
    void display()
    {
      cout<<"\n a= "<<a;
        cout<<"\n b= "<<b;
        cout<<"\n c= "<<c;
    }
};
 Test::Test(int x,int y)
 {
   a=x;
   b=y;
   c=a+b;
 }
int main()
{
    class Test obj1,obj2(100);
    obj1.display();
    cout<<"\n";
    obj2.display();
    return 0;
}
```

3.7 : Symbolic Constants

Q.8 Write a short note on - Symbolic constants.

Ans. :

- There are two problems that are associated with the constants and those are -
  1. **Modifiability :** The constants may need to be modified during the program. For example : The value of pi may be 3.14 at one place in the program and it may be required as 3.14159 at some other place in the same program for the accuracy.
  2. **Understandability :** The purpose of each constant need to be remembered. For example : The constant INDEX may be required to store the index, the constant COUNT may keep track of some count or there may be some constant for SIZE.
- To overcome these problems there is a use of **symbolic constants**. The keyword **final** is used to declare the symbolic constants.
- The syntax for declaring the symbolic constants is -

```
final datatype variable_name=value
```

- For example -

```
final int INDEX=1;
final int SIZE=10;
```

3.8 : Garbage Collection : Destructor and Finalizers

Q.9 How garbage collection is done in Java ?

Ans. :

- Garbage collection is a method of automatic memory management.
- It works as follows -
  1. When an application needs some free space to allocate the nodes and if there is no free space available to allocate the memory for these objects then a system routine called **garbage collector** is called.

  2. This routine then searches the system for the free space. The free space is then made available for reuse.
- Java used **finalize()** method for garbage collection.

END...

Unit IV

4 Inheritance and Polymorphism

4.1 : Introduction and Need for Inheritance

Q.1 What is inheritace ?

☞ [SPPU : June-22, Marks 2]

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

Q.2 Explain the concept of base class and derived class.

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

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

Q.3 Explain the need for inheritance.

Ans. :

1) It helps in reduced code.
2) It makes use of reusability of code.

3) It enhances readability of code.
4) Execution of code is efficient.

4.2 : Types of Inheritance

Q.4 What are various types of inheritance ?

☞ [SPPU : June-22, Marks 2]

Ans. : 1. Single inheritance :

- In single inheritance there is one parent per derived class. This is the most common form of inheritance.

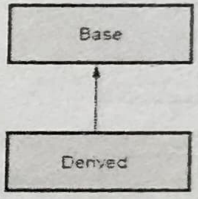


Fig. Q.4.1 Single inheritance

2. Multiple inheritance :

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

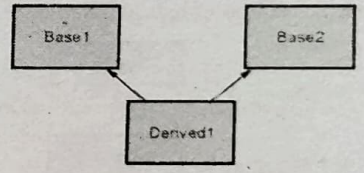


Fig. Q.4.2 Multiple inheritance

- Java **does not implement multiple inheritance** directly but it makes use of the concept called interfaces to implement the multiple inheritance.

3. Multilevel inheritance :

- When a derived class is derived from a base class which itself is a derived class then that type of inheritance is called multilevel inheritance.
- For example - If class A is a base class and class B is another class which is derived from A, similarly there is another class C being derived from class B then such a derivation leads to multilevel inheritance.

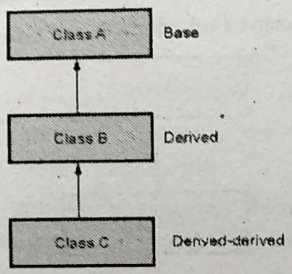


Fig. Q.4.3 Multilevel inheritance

4. Hybrid inheritance :

- When two or more types of inheritances are combined together then it forms the hybrid inheritance. The following Fig. Q.4.3 represents the typical scenario of hybrid inheritance.

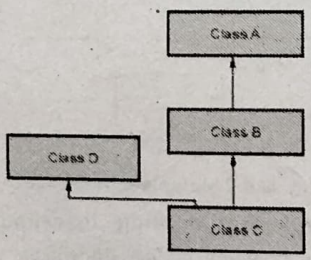


Fig. Q.4.4 Hybrid inheritance

4.3 : Implementation

Q.5 Write a Java program to implement single level inheritance.
OR How can you inherit a class in Java ?

[SPPU : June-22, Marks 5]

Ans. :

```
class A
{
        int a;
        void set_a(int i)
        {
                a=i;
        }
        void show_a()
        {
                System.out.println("The value of a= "+a);
        }
}
class B extends A  //extending  the base class A
{
        int b;
        void set_b(int i)
        {
                b=i;
        }
        void show_b()
        {
                System.out.println("The value of b= "+b);
        }
        void mul()
        {
                int c;
                c=a*b;
                System.out.println(" The value of c= "+c);
        }
}
class InheritDemo1
{
        public static void main(String args[])
        {
                A obj_A=new A();
                B obj_B=new B();
                obj_B.set_a(10);
```

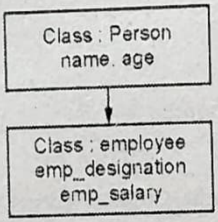
Note that object of class B is accessing method of class A

```
        obj_B.set_b(20);//10
        obj_B.show_a();//20
        obj_B.show_b();//200
        obj_B.mul();
    }
}
```

Output

```
The value of a= 10
The value of b= 20
The value of c= 200
```

Q.6 Write a program to implement following inheritance :



Ans. :

```
class Person
{
    String name;
    int age;
    void get_personInfo(int ag,String nm)
    {
        age=ag;
        name=nm;
    }
    void display_personInfo()
    {
        System.out.println("Name: "+name);
        System.out.println("age: "+age);
    }
}
class Employee extends Person
    //extending  the base class Person
{
    String emp_designation;
    float emp_salary;
```



```
    void get_employeeInfo(String designation,float salary)
    {
        emp_designation=designation;
        emp_salary=salary;
    }
    void display_empInfo()


    {
        System.out.println("Employee
        Designation:"+emp_designation);
        System.out.println("Employee Salary: "+emp_salary);
    }
}
class InheritDemo
{
    public static void main(String args[])
    {
        Employee e=new Employee();
        e.get_personInfo(25,"Ankita");
        e.display_personInfo();
        e.get_employeeInfo("Manager",10000);
        e.display_empInfo();
    }
}
```

Output

```
Name: Ankita
age: 27
Employee Designation: Manager
Employee Salary: 10000.0
```

Q.7 Write a Java code to implement multilevel inheritance

Ans. : Java Program[MultiLvlInherit.java]

```
class A
{
 int a;
 void set_a(int i)
 {
  a=i;
 }
```

```
 void show_a()
 {
  System.out.println("The value of a= "+a);
 }
}
class B extends A
{
 int b;
 void set_b(int i)
 {
  b=i;
 }
 void show_b()
 {
  System.out.println("The value of b= "+b);
 }
}
class C extends B
{
 int c;
 void set_c(int i)
 {
  c=i;
 }
 void show_c()
 {
  System.out.println("The value of c= "+c);
 }
 void mul()
 {
  int ans;
  ans=a*b*c;
  System.out.println(" The value of ans= "+ans);
 }
}
class MultiLvlInherit
{
 public static void main(String args[])
 {
  A obj_A=new A();
  B obj_B=new B();
```



```
  C obj_C=new C();
  obj_C.set_a(10);
  obj_C.set_b(20);
  obj_C.set_c(30);
  obj_C.show_a();//10
  obj_C.show_b();//20
  obj_C.show_c();//30
  obj_C.mul();//6000
 }
}
```

Output

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

4.4 : Benefits and Cost of Inheritance

Q.8 What are the benefits of inheritance ?

Ans. :

1. **Software reusability :** The child class inherits some behaviour of the parent class. Hence for the child class it is not necessary to write the code for the inherited behaviour.
2. **Code sharing :** The same class can be used by many applications or users. At other level the two or more classes developed by a single programmer may be inherited from a single parent class.
3. **Software components :** Using inheritance the reusable software components can be created. Due to creation of such software components new or novel applications can be created with actually no coding or little coding.
4. **Consistency of interface :** When various subclasses are derived from the same superclass then all these subclasses will have the same behaviour. Thus the similar objects have the similar interface and due to which the user will be presented with the collection of objects having similar behaviour.

5. **Rapid prototyping :** Rapid prototyping refers to creation of the basic system in efficient and quick manner. Many software components can be inherited from the parent class and interfaces. Due to which very small amount of code needs to be written to build a prototype system.
6. **Polymorphism and frameworks :** Using polymorphism the higher level components can be created by using different lower level parts that for the suitable application.
7. **Information hiding :** The software component can be reused by the programmer without bothering about its implementation details. This reduces the interconnected nature between the several software systems.

Q.9 What is the cost of inheritance ?

Ans. :

1. **Execution speed :** The inherited methods are often slower than the specialized code because the inherited method have to deal with arbitrary number of subclasses and interfaces. But this reduction in speed can be compensated by increase in speed of program development.
2. **Program size :** Due to use of library of software component the program size gets increased heavily. But due to this memory cost gets decreased and high quality, error free code can be produced.
3. **Message passing overhead :** Communication among the objects occurs using by passing the messages. It is observed that the message passing operation is more costly than the procedure invocation. But the message passing mechanism can be tolerated because of the benefits that are getting due to object oriented techniques.

   By eliminating the polymorphism and by using the dynamic methods or virtual methods the overhead of message passing mechanism can be reduced.

4. **Program complexity :** Due to overuse of inheritance the program complexity gets increased. In order to understand the control flow of the program, it is essential to scan the inheritance graph up and down for several times. Reading and scanning the inheritance graph in an up and down manner for understanding the program flow is called the **yo yo problem**.

4.5 : Constructors in Derived Classes

Q.10 Explain the concept of constructor chaining with suitable example.

Ans. : • Normally a superclass's constructor is called before the subclass's constructor. This is called **constructor chaining**.

- Thus the calling of constructor occurs from top to down.

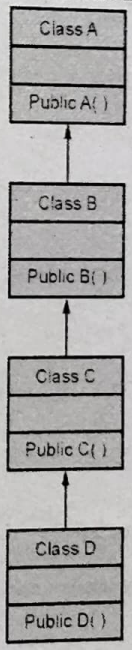


Fig. Q.10.1 Constructor call

- The constructor chaining can be illustrated by following Java program.

Java Program[MainClass.java]

```
class A
{
public A()    //constructor defined
      {
            System.out.println("1. Statement in class A");
      }
}
class B extends A //Derived child of A
{
      public B() //constructor defined
      {
      System.out.println("2. Statement in class B");
      }
}
class C extends B    //derived class of class B
{
      public C()     //constructor defined
      {
            System.out.println("3. Statement in class C");
      }
}
class D extends C  //derived class of class C
{
      public D()   //constructor defined
      {
            System.out.println("4. Statement in class D");
   }
}
class MainClass
{
   public static void main(String args[])
   {
      D obj=new D(); //calling constructor using derived class object
   }
}
```


Output

```
1. Statement in class A
2. Statement in class B
3. Statement in class C
4. Statement in class D
```

4.6 : Method Overriding

Q.11 Explain the concept of method overriding with an example.

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

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

Java Program[OverrideDemo.java]

```
class A
{
      int a=0;
      void fun(int i)
      {
            this.a=i;
      }
}
class B extends A
{
      int b;
      void fun(int i)
      {

            int c;
```

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

Output

```
value of a:15
value of b:20
The value of c= 300
```

Q.12 What is the difference between method overloading and method overriding.

Ans. :

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

4.7 : Abstract Classes

Q.13 What is abstract class ?

Ans. : Abstract class is a class which is declared with abstract keyword. It may contain both abstract and non-abstract methods (no-abstract method means the method with functional body).

For example -

```
abstract class A
{
abstract void fun1();
void fun2()
{
      System.out.println("A:In fun2");
}
}
```

This method is so abstract that it has no definition body.

4.8 : Interfaces

Q.14 What is interfaces ? Also give the difference between class and interface.

☞ [SPPU : June-22, Marks 2]

Ans. : An interface is defined as an abstract type used to specify the behavior of a class.

Difference between class and interface :

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

Class	Interface
The class is denoted by a keyword **class**.	The interface is denoted by a keyword **interface**.
The class contains data members and methods. But the methods are defined in class implementation. Thus class contains an executable code.	The interfaces may contain data members and methods but the methods are not defined. The interface serves as an outline for the class.

By creating an instance of a class the class members can be accessed.	You can not create an instance of an interface.
The class can use various access specifiers like public, private or protected.	The interface makes use of only public access specifier.
The members of a class can be constant or final.	The members of interfaces are always declared as final.

Q.15 Explain how to create an interface in Java with suitable example.

☞ **[SPPU : June-22, Marks 3]**

Ans. :

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

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

Do not compile this program. Simply save it.

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

Java Program[InterfaceDemo.java]

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

Defining the method declared in the interface


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

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

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

```
The value in the class A: 100
This is another method in class A
```

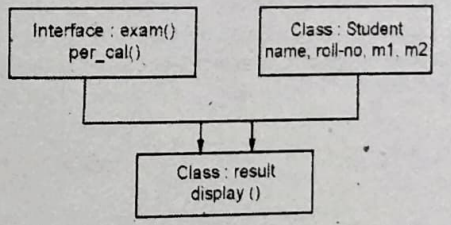
Q.16 Can we achieve multiple inheritance by using interface ? Illustrate with an example.

☞ **[SPPU : June-22, Marks 3]**

Ans. : Yes we can achieve multiple inheritance by using interface.

Example : Refer Q.17.

Q.17 Write a java program.



Ans. :

```
class Student
{
        String name;
        int roll_no;
        double m1,m2;
        Student(String name,int roll_no,double m1,double m2)
```

```
    {
        this.name=name;
        this.roll_no=roll_no;
        this.m1=m1;
        this.m2=m2;
    }
}
interface exam
{
    public void per_cal();
}
class result extends Student implements exam
{
    result(String nm,int roll,double m1,double m2)
    {
        super(nm,roll,m1,m2);

    }
    public void per_cal()
    {
        double percentage;
        percentage=((m1+m2)/200)*100;
        System.out.println("The percentage marks: "+percentage);
    }
    void display()
    {
        System.out.println("Name: "+name);
        System.out.println("Roll_No: "+roll_no);
        System.out.println("Marks m1= "+m1+" m2 = "+m2);
        per_cal();
    }
}
class Test
{
    public static void main(String args[])
    {
        result r=new result("Parth",10,90,95);
        r.display();
    }
}
```


Output

```
Name: Parth
Roll_No: 10
Marks m1= 90.0 m2 = 95.0
The percentage marks: 92.5
```

4.9 : Polymorphism

Q.18 What is polymorphism ?

Ans. : Definition : Polymorphism is a mechanism in which a single action can be performed in different ways. Here Poly means many and Morphs means forms. Hence polymorphism means many forms.

Q.19 Write a Java program to implement polymorphism

Ans. : Compile Time Polymorphism

- The polymorphism that is resolved during compiler time is known as compile time polymorphism.
- Method overloading is an example of compile time polymorphism.
- The method overloading is a technique in which we use the same function name for different purposes.
- The compile time polymorphism is also called as static polymorphism.
- **Example Program**

```
class calculate {
    int sum(int a,int b) {
        return a+b;
    }
    int sum(int a,int b,int c) {
        return a+b+c;
    }
}
class CompileTimePoly {
    public static void main(String args[]) {
        calculate obj = new calculate();
        System.out.println(obj.sum(10,20));
```

```
        System.out.println(obj.sum(10,20,30));
    }
}
```

Q.20 What is polymorphism ? Illustrate types of polymorphism. [SPPU : June-22, Marks 8]

Ans. : Polymorphism : Refer Q.18.

There are two types of polymorphism :

1. Compile time polymorphism
2. Run time polymorphism

1. **Compile time polymorphism :** Refer Q.19.
2. **Run time polymorphism :** The polymorphism which is resolved during run time is called runtime polymorphism.

- Method overriding is an example of run time polymorphism.
- The run time polymorphism is also called as dynamic method dispatch.
- In this technique, the call to method is resolved at run time.
- **Example Program**

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


Output

```
D:\test>javac RunPolyDemo.java
D:\test>java RunPolyDemo
Derived Method Called
```

4.10 : Mechanism for Software Reuse

Q.21 What is software reuse ? Enlist the advantages of software reuse.

Ans. : Definition of software reuse : Software reuse is the process of creating software systems from predefined software components.

Advantages of software reuse

1) Increase software productivity.
2) Shorten software development time.
3) Improve software system interoperability.
4) Develop software with fewer people.
5) Reduce software development and maintenance costs.
6) Produce more standardized software.

4.11 : Efficiency and Polymorphism

Q.22 Write short note on - efficiency and polymorphsim.

Ans. : • Polymorphism helps programmers reuse the code and classes once written, tested and implemented. They can be reused in many ways.

- Single variable name can be used to store variables of multiple data types.
- Polymorphism helps in reducing the coupling between different functionalities.
- One of the disadvantages of polymorphism is that developers find it difficult to implement polymorphism in codes.
- Run time polymorphism can lead to the performance issue as machine needs to decide which method or variable to invoke so it

basically degrades the performances as decisions are taken at run time.

- Polymorphism reduces the readability of the program. One needs to identify the runtime behavior of the program to identify actual execution time.

Q.23 Description : A bank maintains two kinds of accounts for customers. One is saving and other is current. Create a class account that store customer name, account number and type of account. Derived classes with more specific requirements and include necessary methods in order to achieve then following tasks :

i) Accept deposit from the customer and update the balance
ii) Display the balance.
iii) Compute and deposit interest
iv) Permit withdraw and update the balance.
v) Check minimum balance condition and display necessary notice.

☞ [SPPU : June-22, Marks 9]

Ans. : Implementation

```
import java.util.*;
class Account
{
    String cust_name;
    int acc_no;
    double balance;
    String type_of_account;
    Scanner sc;
    public Account()//Destructor
    {
        cust_name="";
        acc_no = 0;
        balance = 0;
        type_of_account ="";
    }
    public void input_data()
    {
        System.out.println("Enter name of Customer");
        cust_name = sc.next();
        System.out.println("Enter account number of Customer");
        acc_no = sc.nextInt();
```



```
        System.out.println("Enter balance: ");
        balance = sc.nextDouble();
        System.out.println("Enter type of account");
        type_of_account = sc.next();
    }
    public void display()
    {
        System.out.println("\n Name: "+cust_name);
        System.out.println("\n Account Number: "+acc_no);
        System.out.println("\nType of Account:
                            "+type_of+account);
        System.out.println("\n Balance Amount:"+balance);
    }
    public void withdraw()
    {
        double amount;
        System.out.println("\n Enter amount to be withdrawn: ");
        amount = sc.nextDouble();
        if( balance >= amount)
        {
            balance = balance - amount;
            System.out.println("Balance: "+balance);
        }
        else
            System.out.println("The amount cannot be drawn");

    }
    public void deposit()
    {
        double amount;
        System.out.println("\n Enter the amount to be deposited: ");
        amount = sc.nextDouble();
        balance = balance + amount;
        System.out.println("Balance: "+balance);
    }
}
class Current extends Account
{
    double servicecharge;
    String chequebook;
```

```
    int penalty;
    public void issueCheque()
    {
        System.out.println("The checkbook issued");
        servicecharge = 20;
        balance = balance - servicecharge;
    }
    public void checkBal()
    {
        System.out.println(balance);
    }
    public void imposePenalty()
    {
        if(balance<=500)
        {
            penalty = 50;
            balance = balance - penalty;
        }
        else
            System.out.println("No Penalty imposed");
    }
}
class Saving extends Account
{
    double interest;
    public void computeInterest()
    {
        int rate;
        rate = 7;
        interest = (amount*no_of_years*rate)/100;
    }
    public void depositInterest()
    {
        balance = balance + interest;
    }
}
class Demo
{
    public static void main(String args[])
    {
```



```
        Saving s = new Saving();
        Current c = new Current();
        s.input_data();
        s.display();
        s.withdraw();

        c.input_data();
        c.display();
        c.deposit();
    }
}
```

END...

Unit V

5 Exception Handling and Generic Programming

5.1 : Exception and Errors

Q.1 Define the terms - Exception and errors.

[SPPU : June-22, Marks 9]

Ans. : • **Exceptions : Exception** is an unusual condition that can occur in the program. This condition is caused due to run time error in the program.

- **Errors :** When any kind of serious problem occurs which could not be handled easily like **OutOfMemoryError** then an error is thrown.

Q.2 State and explain types of errors.

Ans. : • There are two types of errors -

1. **Compile time errors** 2. **Run time errors**

1. Compile time errors : • The errors that are detected by the Java compiler during the compile time are called the compile time errors.

- When compiler issues the errors it will not generate the .class file. Hence we must eliminate all the compile time errors first.
- The most commonly occurring compile time errors are -
  1. Missing semicolons.
  2. Wrong spelling of keywords and identifiers.
  3. Missing brackets of classes and methods.
  4. Use of undeclared variables.

2. Run time errors : • Sometimes the program is free from the compile time errors and creates a .class file. But it does not yield the results as per our expectations. In such situation we declare that the program has run time errors.

- The runtime errors are basically the logical errors that get caused due to the wrong logic.
- The most commonly occurring run time errors are -
  1. Accessing the array element which is out of the index.
  2. In an expression, divide by zero.
  3. Trying to store a value into an array of incompatible type.
  4. Passing the parameters that is not in the valid range.

5.2 : Built-in Exceptions and its Types

Q.3 Give the class hierarchy of exception class.

Ans. : Various exception classes get derived from **Throwable** class. This class hierarchy is given in the following figure.

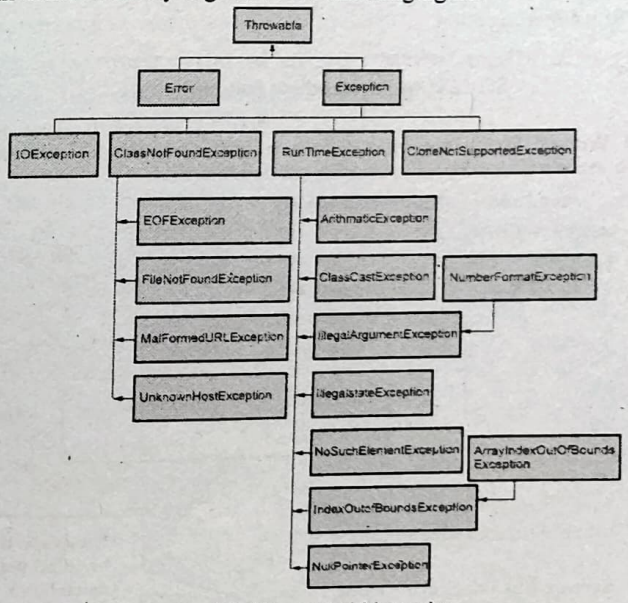


Fig. Q.3.1 Exception hierarchy

Q.4 Define checked and unchecked exceptions.

Ans. : • **Checked exception :** These types of exceptions need to be handled explicitly by the code itself either by using the **try-catch** block or by using **throws**. These exceptions are extended from the **java.lang.Exception** class.

- **For example :** IOException which should be handled using the try-catch block.
- **UnChecked exception :** These type of exceptions need not be handled explicitly. The Java Virtual Machine handles these types of exceptions. These exceptions are extended from **java.lang.RuntimeException** class.

  For example : ArrayIndexOutOfBounds, NullPointerException RunTimeException.

5.3 : Exception Handling Fundamentals

Q.5 Write a Java program to demonstrate the handling of divide by zero error using exception handling mechanism.

Ans. : Java Program[RunErrDemo.java]

```
class RunErrDemo
{
 public static void main(String[] args)
 {
 int a,b,c;
a=10;
b=0;
try
{
c=a/b;
}
catch(ArithmeticException e)
{
System.out.println("\n Divide by zero");
}
```

Exception occurs because the element is divided by 0.

Exception is handled using catch block


```
System.out.println("\n The value of a: "+a);
System.out.println("\n The value of b: "+b);
}
}
```

5.4 : Uncaught Exception

Q.6 Write short note on - uncaught exception.

Ans. :

- If there exists some code in the source program which may cause an exception and if the programmer does not handle this exception then java runtime system raises the exception.
- Following example illustrates how Java runtime system deals with an **uncaught exception**.
- When we use an index which is beyond the range of index then **ArrayIndexOutOfBoundsException** occurs. Following Java program illustrates it.

Java Program [ExceptionProg.java]

```
class ExceptionProg
{
    static void fun(int a[])
    {
    int c;
    c=a[0]/a[2];
      }
      public static void main(String args[])
      {
      int a[]={10,5};
      fun(a);
      }
}
```

5.5 : Multiple Catch Clauses

Q.7 Explain the details of multiple catch statement.

Ans. :

- It is not possible for the try block to throw a single exception always.
- There may be the situations in which different exceptions may get raised by a single try block statements and depending upon the type of exception thrown it must be caught.
- To handle such situation multiple catch blocks may exist for the single try block statements.
- The **syntax** for single try and multiple catch is -

```
try
{
  ...
  ...//exception occurs
}
catch(Exception_type e)
{
  ...//exception is handled here
}
  catch(Exception_type e)
{
  ...//exception is handled here
}
  catch(Exception_type e)
{
  ...//exception is handled here
}
```

Example

Java Program[MultipleCatchDemo.java]

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



```
  {
    int a[] = new int [3];

      try
    {
      for (int i = 1; i <=3; i++)
      {
      a[i] = i *i;
      }

      for (int i = 0; i <3; i++)
      {
        a[i] = i/i;
      }
    }
    catch (ArrayIndexOutOfBoundsException e)
    {
      System.out.println ("Array index is out of bounds");
    }
    catch (ArithmeticException e)
    {
      System.out.println ("Divide by zero error");
    }
  }
}
```

5.6 : Nested Try Statement

Q.8 What is nested try statement ? Explain.

Ans. :

- When there are chances of occurring multiple exceptions of different types by the same set of statements then such situation can be handled using the nested try statements.
- Following is an example of nested try-catch statements -

Java Program[NestedtryDemo.java]

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

```
    {
try
    {
        int a = Integer.parseInt (args [0]);
        int b = Integer.parseInt (args [1]);
        int ans = 0;

      try
      {
        ans = a / b;
        System.out.println("The result is "+ans);
      }
      catch (ArithmeticException e)
      {
        System.out.println("Divide by zero");
      }
    }
    catch (NumberFormatException e)
    {
      System.out.println ("Incorrect type of data");
    }
  }
}
```

Inner try - catch

Outer try - catch

5.7 : Using Throws and Throw

Q.9 Write a Java program to demonstrate the throws keyword.
Ans. :

Java Program

```
    /* This programs shows the exception handling mechanism
using throws
    */
    class ExceptionThrows
    {
    static void fun(int a,int b) throws ArithmeticException
    {
    int c;
     try
```



```
    {
     c=a/b;
    }
    catch(ArithmeticException e)
    {


      System.out.println("Caught exception: "+e);
      }
    }

    public static void main(String args[])
    {
     int a=5;
     fun(a,0);
    }
   }
```

Q.10 Write a Java program to demonstrate the throw keyword.
Ans. :

```
class ExceptionThrow
{
  static void fun(int a,int b)
  {
    int c;
    if(b==0)
     throw new ArithmeticException("Divide By Zero!!!");
   else
       c=a/b;
  }
  public static void main(String args[])
  {
    int a=5;
    fun(a,0);
  }
}
```

5.8 : User Defined Exception using Throw

Q.11 What are the steps to develop user defined exception ?

☞ [SPPU : June-22, Marks 5]

Ans. :

- We can throw our own exceptions using the keyword **throw**.
- The syntax for throwing out own exception is -

```
throw new Throwable's subclass
```

- Here the Throwable's subclass is actually a subclass derived from the **Exception** class.

Q.12 Write a program to input name and age of a person and throws an user define exception if entered age is negative.

Ans. :

```
import java.lang.Exception;
import java.util.Scanner;
class AgeException extends Exception
{
    AgeException(String msg)
    {
        super(msg);
    }
}
class MyExceptDemo3
{
    public static void main (String args [])
    {
        System.out.print("Enter your age: ");
        Scanner in=new Scanner(System.in);
        int age=in.nextInt();
        try
        {
          if(age<0)
            throw new AgeException("Negative Age!!!");
          else
```



```
            System.out.println("Correct Age");
        }
        catch (AgeException e)
        {
          System.out.println (e.getMessage());
        }
    }
}
```

Output

```
Enter your age: -10
Negative Age!!!
```

5.9 : What are Generics ?

Q.13 What is generic ? Explain its need.

Ans. :

- Generic is a mechanism for creating a general model in which generic methods and generic classes enable programmers to specify a single method (or a set of related methods) and single class (or a set of related classes) for performing the desired task.
- Following features show the importance of generics in Java.
  1. It saves the programmers burden of creating separate methods for handling data belonging to different data types.
  2. It allows the **code reusability**.
  3. **Compact code** can be created.

Q.14 Explain the concept of generic classes with example.

Ans. : • A generic class or interface contains one or more variables of generic data type. Following is a simple example which shows how to define the generic class.

```
public class Test<T>
{
```

```
    public Test(){val=null;}
    public Test(T val)
    {
        this.val=val;
    }
    public getVal()
    {
      return val;
    }
    public setVal()
    {
      val=newValue;
    }
    private T val;   //variable defined as of genric type
}
```

The concept of generic class or interface supports the idea of type-independency.

5.10 : Introduction to Language Specific Collection Interface

Q.15 Write a short note on - Collection framework.

Ans. :

- The standard data structures can be implemented in Java using some **library classes** and methods.
- These classes are present in the **java.util** package.
- The collection framework is comprized of **collection classes** and **collection interfaces.**
- Basically collection is a group of objects which are designed to perform certain task. These tasks are associated with alteration of data structures.
- The collection classes are the group of classes used to implement the collection interfaces.

- Various collection classes are

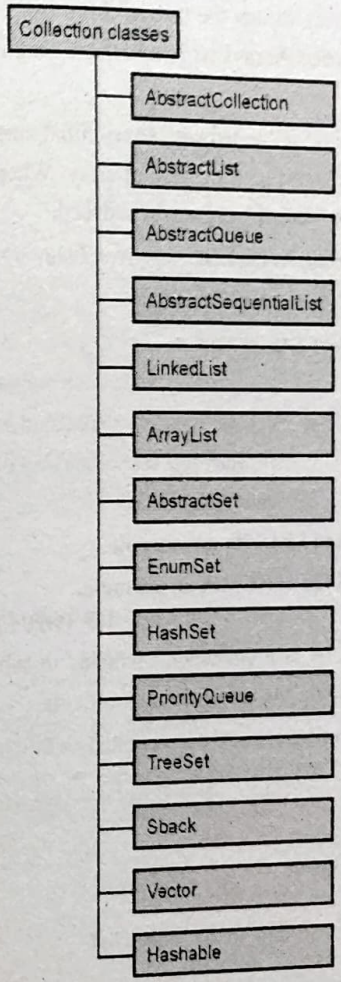


Fig. Q.15.1 Collection class hierarchy

Q.16 What is ArrayList ?

Ans. : • The **ArrayList** class implements the List interface. It is used to implement the dynamic array.

- The dynamic array means the size of array can be created as per requirement. Hence **ArrayList** is a variable length array of object references.
- Initially **ArrayList** is created with some initial size and then as per requirement we can extend the size of array. When the objects are removed then the size of array can be reduced.
- The syntax of using **ArrayList** is as given below -

```
ArrayList() ← Creates an empty list
ArrayList(Collection collection) ← Creates a list in which the
                                   collection elements are added
ArrayList(int c) ← Creates a list with specified capacity c, the
                   capacity represents the size of the
                   underlying array
```

Q.17 Explain Linked List with an example.

OR Explain linked list class with an example.

☞ **[SPPU : June-22, Marks 9]**

Ans. : • Linked List is a collection of nodes in which every node contains two fields Data and Next pointer fields.

- The link list can be graphically represented as follows -

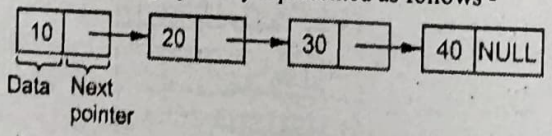


Fig. Q.17.1 Linked list

- The java.util package provides the collection class LinkedList in order to implement the List interface.
- The syntax of using this class is

```
Linked List() ← creates an empty linked list
Linked List(Collection collection) ← creates a linked list having
                                     the elements of collection
```


Programming Example : Following program makes use of various methods such as **add()**, **remove()**, **addFirst()**, **addLast()**, **removeFirst()**, **removeLast()** for manipulating the contents of linked list.

```
import java.io.*;
import java.util.*;
class LinkedListProg
{
  public static void main(String[] args)throws IOException
  {
    char ans='y',ch='y';
    int choice,val,position;
   String str;
    LinkedList obj=new LinkedList();
   Scanner s=new Scanner(System.in);
    do
    {
      System.out.println("\n\t\t Program for Implementing Linked List");
      System.out.print("\n1.Create\n2.Display \n3.Insert First\n4. Insert Last");
      System.out.print("\n5.Delete First\n6.Delete Last");
      System.out.print("\n7.Insert At any Position");
      System.out.println("\n8.Delete From any Position");
      System.out.println("Enter Your choice");
      choice=s.nextInt();
      switch(choice)
      {
        case 1:
          do
          {
            System.out.println("\n Enter the element to be inserted in the list");
            val=s.nextInt();
            obj.add(val);
            System.out.println(" Do u want to insert more elements?");
```

```
        str =s.next();
            ans=str.charAt(0);

          }while(ans=='y');
          break;
        case 2:
          System.out.println("\t The List elements are... "+obj);
          System.out.println("\t The size of linked list is... "+obj.size());
          break;
        case 3: System.out.println("\n Enter the element to be inserted in the list");
          val=s.nextInt();
          obj.addFirst(val);
          System.out.println("\n The element inserted!!!");
          break;
        case 4: System.out.println("\n Enter the element to be inserted in the list");
          val=s.nextInt();
          obj.addLast(val);
          System.out.println("\n The element inserted!!!");
          break;
        case 5: obj.removeFirst();
          System.out.println("\n The element deleted!!!");
          break;
        case 6: obj.removeLast();
          System.out.println("\n The element deleted!!!");
          break;
        case 7:System.out.println("\n Enter the element to be inserted in the list");
          val=s.nextInt();
          System.out.println("\n Enter the position at which the element is to be inserted");
          position=s.nextInt();
          obj.add(position,val);
          System.out.println("\n The element inserted!!!");
          break;
```



```
        case 8: System.out.println("\n Enter the position of element to be deleted");
          position=s.nextInt();
          obj.remove(position);
          System.out.println("\n The element deleted!!!");
          break;
      }
      System.out.println("\n Do u want to go to main menu?");
      str=s.next();
      ch=str.charAt(0);
    }while(ch=='y');
  }
}
```

Q.18 Give the difference between ArryaList and LinkedList collection.

Ans. :

Sr. No.	ArrayList	LinkedList
1.	ArrayList internally uses dynamic array to store the elements.	LinkedList internally uses doubly linked list to store the elements.
2.	It internally uses array. If any element is removed from the array, all the bits are shifted in memory.	It uses doubly linked list so no bit shifting is required in memory.
3.	Manipulation with ArrayList is slow	Manipulation with LinkedList is faster than ArrayList
4.	ArrayList class can act as a list only because it implements List only.	LinkedList class can act as a list and queue both because it implements List and Deque interfaces.
5.	ArrayList is better for storing and accessing data.	LinkedList is better for manipulating data.

Q.19 What is set interface ? Enlist some commonly used methods of set interface.

Ans. :

- The set interface is used to define the set of elements.
- It extends the collection interface.
- This interface defined unique elements. Hence if any duplicate elements is tried to insert in the set then the add() method returns false.
- Some commonly used functionalities in Set interface are -

Method	Description
add( )	Adds an object to the collection.
clear( )	Removes all objects from the collection.
contains( )	Returns true if a specified object is an element within the collection.
isEmpty( )	Returns true if the collection has no elements.
iterator( )	Returns an Iterator object for the collection, which may be used to retrieve an object.
remove( )	Removes a specified object from the collection.
size( )	Returns the number of elements in the collection.

END...

Unit VI

6 File Handling and Design Patterns

6.1 : Concept of Stream

Q.1 Define the term - stream. ☞ [SPPU : June-22, Marks 2]

Ans. :

- Stream is basically a channel on which the data flow from sender to receiver.
- An input object that reads the stream of data from a file is called **input stream.**
- The output object that writes the stream of data to a file is called **output stream.**

6.2 : Byte Stream and Character Stream Classes

Q.2 Explain byte stream classes in detail. ☞ [SPPU ; June-22, Marks 3]

Ans. :

- The byte stream is used for inputting or outputting the bytes.
- There are two super classes in byte stream and those are **InputStream** and **OutputStream** from which most of the other classes are derived.
- These classes define several important methods such as **read()** and **write()**.
- The input and output stream classes are as shown in Fig. 6.2.1 and Fig. Q.2.1.

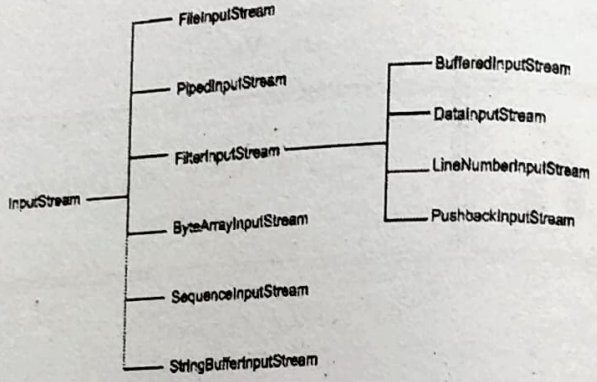


Fig. Q.2.1 InputStream classes

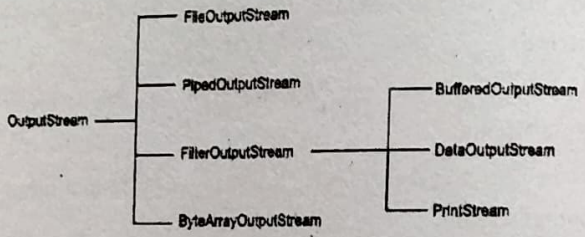


Fig. Q.2.2 OutputStream classes

Q.3 Explain character stream classes in detail.

☞ **[SPPU : June-22, Marks 4]**

Ans. :

- The character stream is used for inputting or outputting the characters.
- There are two superclasses in character stream and those are **Reader** and **Writer**.
- These classes handle the unicode character streams.
- The two important methods defined by the character stream classes are **read()** and **Write()** for reading and writing the characters of data.

- Various Reader and Writer classes are –

FileReader	FileWriter
PipeReader	PipeWriter
FilterReader	FilterWriter
BufferedReader	BufferedWriter
DataReader	DataWriter
LineNumberReader	LineNumberWriter
PushbackReader	PushbackWriter
ByteArrayReader	ByteArrayWriter
SequenceReader	SequenceWriter
StringBufferReader	StringBufferWriter

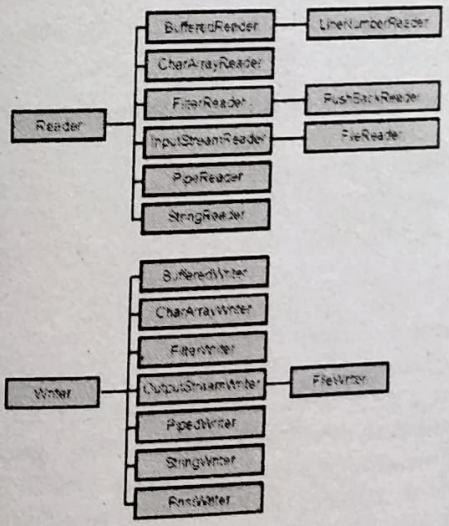


Fig. Q.3.1

6.3 : Using the File Class

Q.4 Write a constructor for file class.

Ans. :

- The file constructor is used in Java while handling the file I/O. These constructors are there in java.io.File. For example -

```
File f1=new File("input.dat");
```

Methods in class File :

1. **Public String getPath()** - This method returns a string that contains the path being used for the file.
2. **Public String getParent()** - This method returns a string that contains the name of the single directory which contains this file in the hierarchy.
3. **Public boolean exists()** - This method indicates whether or not a particular file exists where you expect it to be.
4. **Public boolean canWrite()** - This method indicates whether you have write access to this file.

Q.5 Write a Java program for obtaining the properties of file.

Ans. :

```
import java.io.File;
class FileProperties {
public static void main(String args[])
    {
        File FileObj=new File("test1/fun.txt");
        System.out.println("-------------------------------------------");
        System.out.println("File Properties are as follows...");
        System.out.println("-------------------------------------------");
        System.out.println("File Name: " + FileObj.getName());
        System.out.println("Path: " + FileObj.getPath());
        System.out.println("Abs Path: " + FileObj.getAbsolutePath());
        System.out.println("Parent: " + FileObj.getParent());
        System.out.println("This file exists: "+FileObj.exists());
```



```
        System.out.println(FileObj.canWrite() ?
        "is writeable" : "is not writeable");
        System.out.println(FileObj.canRead() ?
        "is readable" : "is not readable");
        System.out.println("is "+(FileObj.isDirectory() ? "" : " not "
        + " a directory"));
        System.out.println("It is a normal file: "+FileObj.isFile());
        System.out.println("This file is absolute:
        "+FileObj.isAbsolute());
        System.out.println("File last modified: " + new
        java.util.Date(FileObj.lastModified()));
        System.out.println("File size: " + FileObj.length() + " Bytes");
    }
}
```

6.4 : Input/Output Exceptions

Q.6 Describe the commonly used input and output exceptions.

Ans. :

I/O Exception Class	Purpose
IOException	If some sort of I/O related exception occurs then this class is used to handle it.
FileNotFoundException	This is the most commonly used exception during I/O operations using files. If the desired file for performing I/O is not found then this exception is used.
EOFException	If we come across end of file unexpectedly then this exception is raised.
InterruptedIOException	If I/O operation is interrupted , then this exception is used to handle the situation.

6.5 : Creation of Files

Q.7 How will you create a file using file stream class ? Explain with example.

Ans. :

- File stream object is an important entity which helps us to locate desired file using a filename. We need to create a file first and then initialize the file stream object.

Following sample code makes use **FileWriter** class

```
try{
FileWriter fwrt=new FileWriter("D:\\myfile.txt");
fwrt.write("Java Programming is Fun!!!");
fwrt.close();
}
catch(Exception e)
{
      System.out.println(e);
}
```

6.6 : Reading/Writing Character

Q.8 Write a program to copy contents of one file to another file using character stream class.

OR Write a java program to copy the content of the file "file1.txt" into new file"file2.txt". ☞ **[SPPU : June-22, Marks 8]**

Ans. :

```
import java.io.FileReader;
import java.io.FileWriter;
public class CopyFileProg
{
      public static void main(String args[])throws Exception{
            FileReader fr=new FileReader("D:\\file1.txt");
            FileWriter fwrt=new FileWriter("D:\\file2.txt");
            int i;
            while((i=fr.read())!=-1)
            {
```



```
                  fwrt.write((char)i);
            }
            System.out.println("The Data is copied from file1 to file2");
            fr.close();
            fwrt.close();
      }
}
```

Output

```
The Data is copied from file1 to file2
D:\MSBTE_JAVA _Programs> cd..
D:\> type file1.txt
Java is fun!!!
D:\> type file2.txt
Java is fun!!!
D:\>
```

6.7 : Reading/Writing Bytes

Q.9 Write a program to copy contents of one file to another. Using byte stream classes.

Ans. :

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
public class CopyFileProg1
{
      public static void main(String args[])throws Exception
      {
            FileInputStream fin=new FileInputStream("D:\\file1.txt");
            FileOutputStream fout=new
            FileOutputStream("D:\\file2.txt");
            int i;
            while((i=fin.read())!=-1)
            {
                  fout.write((char)i);
            }
            System.out.println("The Data is copied from file1 to file2");
            fin.close();
            fout.close();
      }
}
```

6.8 : Handling Primitive Data Types

Q.10 Write a Java program to write some data to a sample file and then read this sample file. Display the contents that you have read from the sample file.

Ans. :

```
import java.io.*;
class PrimitiveDemo
{
    public static void main(String args[]) throws IOException
    {
        //creating file and data stream object and
        initializing them for writing purpose
        File f=new File("d:\\Primfile.txt");
        FileOutputStream fos=new FileOutputStream(f);
        DataOutputStream dos=new DataOutputStream(fos);
        //Writing primitive data to the file
        System.out.println("\tWriting some data to the file ...");
        dos.writeInt(301);
        dos.writeChar('A');
        dos.writeBoolean(true);
        dos.writeDouble(123.45);
        //closing the output streams of
        filestream and datastream classes
        dos.close();
        fos.close();
        //initializing file and data stream
        objects for reading purpose
        FileInputStream fis=new FileInputStream(f);
        DataInputStream dis=new DataInputStream(fis);
        //Reading primitive data from the file
        System.out.println("\tReading some data from the file ...");
        System.out.println(dis.readInt());
        System.out.println(dis.readChar());
        System.out.println(dis.readBoolean());
        System.out.println(dis.readDouble());
```



```
        //closing the input streams of filestream
        and datastream classes
        dos.close();
        fos.close();
    }
}
```

6.9 : Concatenating and Buffer Files

Q.11 Write short note on – Concatenating and buffer files.

Ans. : Concatenation : This is the operation by which two or more files are saved in one file.

Buffer Files : In Java we can create a buffer to store temporary data that is read from and written to a stream and this process is known as input and output buffer operation.

Following Java program shows the concatenation of two files and displaying the result on Console.

```
import java.io.*;
class ConcatAndBuffer
{
  public static void main(String[] args)
  {
      try{
        FileInputStream f1 = new FileInputStream("d:\\File1.txt");
        FileInputStream f2 = new FileInputStream("d:\\File2.txt");
        SequenceInputStream resultFile = null;

        resultFile = new SequenceInputStream(f1,f2);

        BufferedInputStream inBuffer =
            new BufferedInputStream(resultFile);

        BufferedOutputStream outBuffer =
            new BufferedOutputStream(System.out);
```

```
        int ch;
        while((ch = inBuffer.read())!=1){
            outBuffer.write((char)ch);
        }
        inBuffer.close();
        outBuffer.close();
        f1.close();
        f2.close();
    }catch(Exception e) {}
  }
}
```

6.10 : Random Access Files

Q.12 What is random access file ?

Ans. :

- The **RandomAccessFile** class in the Java enables us to read or write to a specific location in the file at the file pointer.
- The **java.io.RandomAccessFile.seek(long pos)** method sets the filepointer offset, which is set to the beginning of this file, at which the next read or write occurs. The pos is the offset position, measured in bytes from the beginning of the file. This method does not return a value. The IOException is used while using this method.

Java Program[RandomAccessFileDemo.java]

```
import java.io.*;
public class RandomAccessFileDemo
{
  public static void main(String[] args)
  {
    try {
      RandomAccessFile raf = new
      RandomAccessFile("d://names.dat","rw");
      String names[] = new String[5];
```



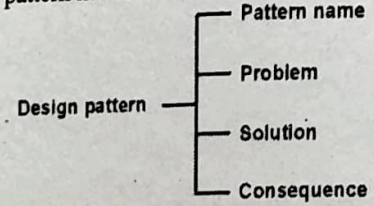
```
      names[0]      = "Archana";
      names[1]      = "Shilpa";
      names[2]      = "Sujata";
      names[3]      = "Soumya";
      names[4]      = "Shivraj";

      for (int i = 0; i < names.length; i++)
        {
        raf.writeUTF(names[i]);
      }
      raf.seek(raf.length());
      raf.writeUTF("Anuja");
      raf.seek(0);
      while (raf.getFilePointer() < raf.length())
        {
        System.out.println(raf.readUTF());
      }
       raf.seek(0);
     raf.writeUTF("Supriya");
      raf.seek(0);
      System.out.println("\t List After Modification...");
     while (raf.getFilePointer() < raf.length())
       {
       System.out.println(raf.readUTF());
     }
    }
    catch (FileNotFoundException e)
    {
      e.printStackTrace();
    }
    catch (IOException e)
    {
      e.printStackTrace();
    }

  }
}
```

6.11 : Introduction to Design Pattern

Q.13 What is design pattern ?

Ans. : Definition of design pattern : Design pattern is general reusable solution to commonly occurring problem within a given context in software design.

In general the pattern has four essential elements -



6.12 : Types of Design Patterns

Q.14 Explain the three types of design patterns.

Ans. :

(1) Creational Pattern

Creational design pattern are the design pattern that are used for object creation mechanism. This type of pattern is used in the situation when basic form of object creation could result in design problems or increase complexity of a code base.

The creational patterns show how to make the software design flexible.

There are well known design patterns that are part of creational pattern.

These are - Abstract factory, factory method, singleton pattern.

(2) Structural Pattern

- In software engineering, structural design patterns are design patterns that ease the design by identifying a simple way to realize relationships between entities.

- The structural design pattern serves as a blueprint for how different classes and objects are combined to form larger structures.
- Structural patterns are just similar to data structures.
- For example - Bridge, adapter, composite.

(3) Behavioral Pattern

- The behavioral design patterns are design patterns that identify common communication patterns between objects and realize these patterns. By doing so, these patterns increase flexibility in carrying out this communication.
- A behavioral pattern explains how objects interact.
- It describes how different objects and classes send messages to each other to make things happen and how the various tasks are divided among different objects.

6.13 : Adapter

Q.15 What is adapter pattern ? [SPPU : June-22, Marks 4]

Ans. :

Intent

- The adapter pattern allows the interface of existing class to be used from another interface.
- It is often used to make existing classes work with other without modifying their source code.
- The adapter helps two incompatible interfaces to work together.
- When one class relies upon a specific interface that is not implemented by another class, the adapter acts as a translator between the two types. Thus this pattern translates one interface for a class into another compatible interface.

Structure

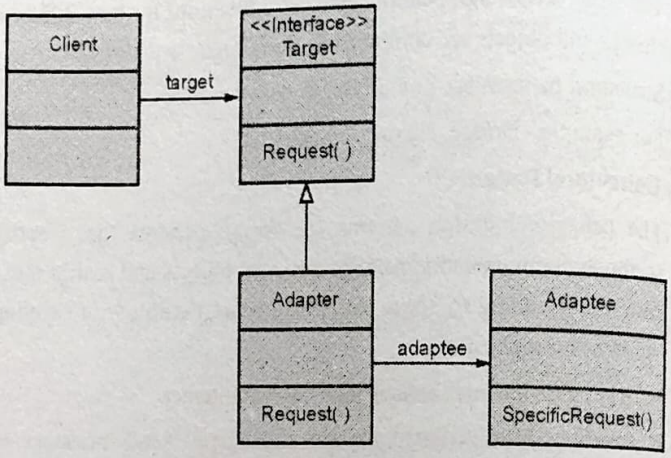


Fig. Q.15.1 : Representation

Application

Adapter pattern acts as a bridge between two compatible functionalities. The **bar code reader system** is a system in which the adapter pattern can be observed.

6.14 : Singleton

Q.16 What is singleton pattern ?

☞ **[SPPU : June-22, Marks 4]**

Ans. :

Intent

- The class has only one instance and it provides global point of access to it.

Motivation

- Singleton is a simplest pattern. This type of pattern belongs to creational design pattern.

- Singleton pattern is created in a situation in which **only one instance of a class** must be created and this instance can be used as a global point of access.
- Singleton work by having special method to get the instance of the desired object

Structure

Following diagram represents how to implement Singleton design pattern.

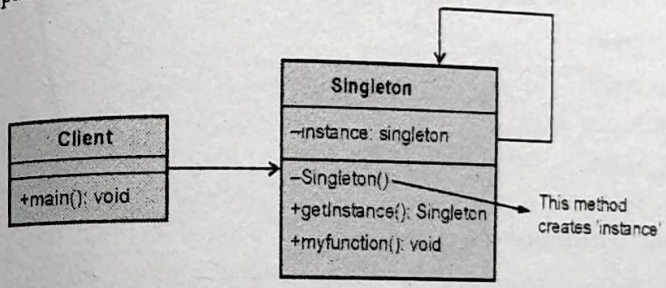


Fig. Q.16.1 : Representation of singleton

Application

Incometax Calculator is a typical example of singleton pattern. The client instantiates the instance for Incometax Calculator only once when needed.

6.15 : Iterator

Q.17 What is iterator pattern ? Give the merits and demerits of it.

OR What is adapter pattern ?

Ans. : The Iterator provides ways to access elements of an aggregate object sequentially without exposing the underlying structure of the object.

Merits

1. This design pattern accesses the contents of a collection without exposing its internal structure.

2. It supports multiple simultaneous traversals of a collection.
3. Using iterator uniform interface for traversing different collection is possible

Demerits

Iteration can be done only once. If you reach the end of the collection of elements, its done. If we need to iterate again we should get a new iterator.

Q.18 Description : Student management system using file handling. Use appropriate data members and methods.

i. Create

ii. Display

iii. Search

iv. Modify

v. Delete **[SPPU : June-22, Marks 9]**

Ans. : Implementation :

```
#include<iostream>
#include<iomanip>
#include<fstream>
#include<cstring>

using namespace std;
class Student_CLASS
{
        typedef struct STUDENT
        {
                char name[10];
                int Rollno;
        }Rec;
        Rec Records;
        public:
                void Create();
                void Display();
                void Update();
                void Delete();
                void Append();
```



```
                int Search();

};
void Student_CLASS::Create()
{
        char ch='y';
        fstream seqfile;
        seqfile.open("D:\\STUD.DAT",ios::in|ios::out|ios::binary);
        do
        {
                cout<<"\n Enter Name: ";
                cin>>Records.name;
                cout<<"\n Enter Rollno: ";
                cin>>Records.Rollno;
                //then write the record containing this data in the file
                seqfile.write((char*)&Records,sizeof(Records));
                cout<<"\nDo you want to add more records?";
                cin>>ch;
        }while(ch=='y');
        seqfile.close();
}
void Student_CLASS::Display()
{
        fstream seqfile;
        int n,m,i;
        seqfile.open("D:\\STUD.DAT",ios::in|ios::out|ios::binary);
        //positioning the pointer in the file at the beginning
        seqfile.seekg(0,ios::beg);
        cout<<"\n The Contents of file are ..."<<endl;
        //read the records sequentially
        while(seqfile.read((char *)&Records,sizeof(Records)))
        {
                if(Records.Rollno!=-1)
                {
                        cout<<"\nName: "<<Records.name;
                        cout<<"\nRollno: "<<Records.Rollno;
                        cout<<"\n";          }
}
int last_rec=seqfile.tellg();//last record position
//formula for computing total number of objects in the file
n=last_rec/(sizeof(Rec));
```

```
 seqfile.close();
}
void Student_CLASS::Update()
{
        int pos;
        cout<<"\n For updation,";
        fstream seqfile;
        seqfile.open("D:\\STUD.DAT",ios::in|ios::out|ios::binary);
        seqfile.seekg(0,ios::beg);
//obtaining the position of desired record in the file
        pos=Search();
        if(pos==-1)
        {
         cout<<"\n The record is not present in the file";
         return;
        }
//calculate the actual offset of the desired record in the file
        int offset=pos*sizeof(Rec);
        seqfile.seekp(offset);//seeking the desired record for modification
        cout<<"\n Enter the values for updation...";
        cout<<"\n Name: ";
        cin>>Records.name;
        cout<<"\n Rollno: ";
        cin>>Records.Rollno;
        seqfile.write((char*)&Records,sizeof(Records))<<flush;
        seqfile.seekg(0);
        seqfile.close();
        cout<<"\n The record is updated!!!";
}
void Student_CLASS::Delete()
{
        int id,pos;
        cout<<"\n For deletion,";
        fstream seqfile;
        seqfile.open("D:\\STUD.DAT",ios::in|ios::out|ios::binary);
        seqfile.seekg(0,ios::beg);//seeking for reading purpose
        pos=Search();//finding pos. for the record to be deleted
        if(pos==-1)
        {
         cout<<"\n The record is not present in the file";
```



```
         return;
        }
//calculate offset to locate the desired record in the file
        int offset=pos*sizeof(Rec);
        seqfile.seekp(offset);//seeking the desired record for deletion
        strcpy(Records.name,"");
        Records.Rollno=-1;
        seqfile.write((char*)&Records,sizeof(Records))<<flush;
        seqfile.seekg(0);
        seqfile.close();
        cout<<"\n The record is Deleted!!!";

}
void Student_CLASS::Append()
{
        fstream seqfile;
        seqfile.open("D:\\STUD.DAT",ios::ate|ios::in|ios::out|ios::binary);
        seqfile.seekg(0,ios::beg);
        int i=0;
        while(seqfile.read((char *)&Records,sizeof(Records)))
        {
                i++;//going through all the records
                        // for reaching at the end of the file
        }
        //instead of above while loop
        //we can also use seqfile.seekg(0,ios::end)
        //for reaching at the end of the file
        seqfile.clear();//turning off EOF flag
        cout<<"\n Enter the record for appending";
        cout<<"\nName: ";
        cin>>Records.name;
        cout<<"\nRollno: ";
        cin>>Records.Rollno;
        seqfile.write((char*)&Records,sizeof(Records));
        seqfile.seekg(0);//reposition to start(optional)
        seqfile.close();
        cout<<"\n The record is Appended!!!";
}
int Student_CLASS::Search()
{
        fstream seqfile;
```

```
    int id,pos;
    cout<<"\n Enter the Rollno for searching the record ";
    cin>>id;
    seqfile.open("D:\\STUD.DAT",ios::ate|ios::in|ios::out|ios::binary);
    seqfile.seekg(0,ios::beg);
    pos=-1;
    int i=0;
    while(seqfile.read((char *)&Records,sizeof(Records)))
    {
        if(id==Records.Rollno)
        {
            pos=i;
            break;
        }
        i++;
    }
    return pos;
}
int main()
{
    Student_CLASS List;
    char ans='y';
    int choice,key;
    do
    {
     cout<<"\n        Main Menu        "<<endl;
     cout<<"\n 1.Create";
     cout<<"\n 2.Display";
     cout<<"\n 3.Update";
     cout<<"\n 4.Delete";
     cout<<"\n 5.Append";
     cout<<"\n 6.Search";
     cout<<"\n 7.Exit";
     cout<<"\n Enter your choice ";
     cin>>choice;
     switch(choice)
     {
        case 1:List.Create();
               break;
        case 2:List.Display();
               break;
```



```
        case 3:List.Update();
               break;
        case 4:List.Delete();
               break;
        case 5:List.Append();
               break;
        case 6:key=List.Search();
               if(key<0)
                   cout<<"\n Record is not present in the file";
               else
                   cout<<"\n Record is present in the file";
               break;
        case 7:exit(0);
     }
     cout<<"\n\t Do you want to go back to Main Menu?";
     cin>>ans;
     }while(ans=='y');
     return 0;
}
```

Output

```
Main Menu
1.Create
2.Display
3.Update
4.Delete
5.Append
6.Search
7.Exit

Enter your choice 1
Enter Name: AAA
Enter Rollno: 10
Do you want to add more records?y
Enter Name: BBB
Enter Rollno: 20
Do you want to add more records?y
Enter Name: CCC
Enter Rollno: 30
Do you want to add more records?n
    Do you want to go back to Main Menu?y
      Main Menu
```

```
1.Create
2.Display
3.Update
4.Delete
5.Append
6.Search
7.Exit
Enter your choice 2
The Contents of file are ...
Name: AAA
Rollno: 10

Name: BBB
Rollno: 20

Name: CCC
Rollno: 30
      Do you want to go back to Main Menu?y
        Main Menu
1.Create
2.Display
3.Update
4.Delete
5.Append
6.Search
7.Exit
Enter your choice 3
For updation,
Enter the Rollno for searching the record 20

Enter the values for updation...
Name: XXX
Rollno: 11

The record is updated!!!
      Do you want to go back to Main Menu?y
        Main Menu
1.Create
2.Display
3.Update
```



```
4.Delete
5.Append
6.Search
7.Exit
Enter your choice 2
The Contents of file are ...
Name: AAA
Rollno: 10

Name: XXX
Rollno: 11

Name: CCC
Rollno: 30

      Do you want to go back to Main Menu?n
```

END...