UNIT II INHERITANCE AND INTERFACES

Inheritance – Super classes- sub classes –Protected members – constructors in sub classes - the Object class – abstract classes and methods- final methods and classes – Interfaces – defining an interface, implementing interface, differences between classes and interfaces and extending interfaces - Object cloning -inner classes, Array Lists - Strings.

2.1 INHERITANCE

How do you implement multiple inheritances in Java? Explain. (4)
Explain the types of inheritance in java with examples. (13)

The properties of base class will be reused in derived class is called as inheritance. The old class is known as a base class or super class or parent class, and the new class is known as subclass or derived class or child class. By using **extends** keyword the properties of super class will be **reused** in sub class.

Types of Inheritance:

1. Single Inheritance
2. Multilevel Inheritance
3. Hierarchical Inheritance.
4. Multiple Inheritances. (Java does not support it, but achieved using interface)

Advantage of Inheritance

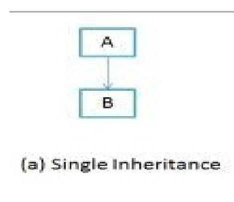
- *Reusability*
- *Extensibility*
- *Data hiding*
- *Overriding*

2.1.1 Single Inheritance

Single Inheritance has only one super class and one sub class. The below diagram shows that **A** is a super class and **B** is a sub class. Here the properties of class A will be reused in class B.

Syntax:

```
Class superclass
{
//body of the class
}
Class subclass extends superclass
{
//body of the class
}
```



(a) Single Inheritance

Example:

```
Class A
{
public void methodA()
{
System.out.println("Base class method");
}
}
Class B extends A
{
public void methodB()
{
System.out.println("Child class method");
}
}

Class Demo
{
public static void main(String args[])
{
B obj = new B();
obj.methodA(); //calling super class method
obj.methodB(); //calling Sub class method
}
}
```

OUTPUT:
Base class method
Child class method

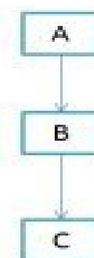
2.1.2 Multilevel Inheritance

How do you implement multilevel inheritances in Java? Explain. (4) (N/D 15)

When a class extends a class, which extends another class then this is called **multilevel inheritance**. For example class C extends class B and class B extends class A then this type of **inheritance** is known as **multilevel inheritance**.

Syntax:

```
class SuperClass
{
//body of the class
}
class SubClassOne extends SuperClass
{
//body of the class
}
class SubClassTwo extends SubClassOne
{
//body of the class
}
```



(d) Multilevel Inheritance

Example

```
class A
{
  public void methodA()
   {
    System.out.println("Class A method");
   }
}
class B extends A
{
  public void methodB()
  {
   System.out.println("class B method");
  }
}
class C extends B
{
  public void methodC()
  {
   System.out.println("class C method");
  }
}

class Demo {
  public static void main(String args[])
  {
   C obj = new C();
   obj.methodA();
   obj.methodB();
   obj.methodC();
  }
}
```

OUTPUT:
Class A Method
Class B Method
Class C Method

Example 2

```
import java.io.*;
import java.util.Scanner;

class StudentBasicInfo
{
int regno;
String name;
Scanner sc=new Scanner(System.in);
 void getmethodA()
{
System.out.println("Enter the Name: ");
name=sc.next();
System.out.println("Enter the Register Number: ");
regno= sc.nextInt();
}
void displaymethodA()
{
System.out.println("Name: " + name);
System.out.println("Register Number: " + regno);
}
}
class StudentHSCMarks extends StudentBasicInfo
{
int tamil, english, maths, biology, phy, chem;
void getmethodB()
{
getmethodA();
System.out.println("Enter the Marks: ");
tamil= sc.nextInt();
english= sc.nextInt();
maths= sc.nextInt();
biology= sc.nextInt();
phy= sc.nextInt();
```

```
chem= sc.nextInt();
}
}

class StudentAvg extends StudentHSCMarks
{
int total;
float avg;

void calculate()
{
total=tamil+english+maths+biology+phy+chem;
avg=total/12;
}

void displaymethodB()
{
displaymethodA();
System.out.println("Total: " +total);
System.out.println("Average: " + avg);
}
}

class StudDemo
{
public static void main(String args[])throws IOException
{
StudentAvg obj = new StudentAvg();
obj.getmethodB();
obj.calculate();
obj.displaymethodB();
}
}
```

INPUT:

Enter the Name: Raj
Enter the Register Number: 12345
Enter the Marks:
120
120
120
120
120
120

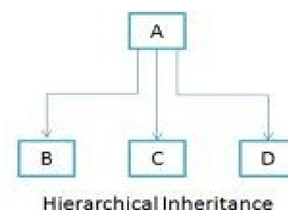
OUTPUT:
Name: Raj
Register Number: 12345
Total: 720
Average: 60.00

2.1.3 Hierarchical Inheritance

When more than one classes inherit a same class then this is called hierarchical inheritance. For example class B, C and D extends a same class A.

Example

```
class A
{
public void methodA()
{
System.out.println("method of Class A");
}
}
class B extends A
{
public void methodB()
{
System.out.println("method of Class B");
}
}
class C extends A
{
public void methodC()
{
System.out.println("method of Class C");
}
}
class D extends A
{
public void methodD()
{
System.out.println("method of Class D");
}
}
class Demo
{
public static void main(String args[])
{
B obj1 = new B();
C obj2 = new C();
D obj3 = new D();
obj1.methodA();
obj1.methodB();
obj2.methodC();
obj3.methodD();
}
}
```



Hierarchical Inheritance

Output:

```
method of Class A
method of Class B
method of Class C
method of Class D
```

2.2 CONSTRUCTORS IN SUB CLASSES

- A constructor invokes its superclass's constructor explicitly and if such explicit call to superclass's constructor is not given then compiler makes the call using super() as a first statement in constructor.
- 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.

Example:

```
class A
{
public A()
{
System.out.println("1. From Class A");
}
}
class B extends A
{
public B()
{
System.out.println("2. From Class B");
}
}
class C extends B
{
public C()
{
System.out.println("3. From Class C");
}
}
class Demo
{
public static void main(String args[])
{
C obj= new C();
}
}
```

2.2.1 Invoking Superclass Constructor

If a class is inheriting the properties of another class, the subclass automatically acquires the default constructor of the superclass. But if you want to call a parameterized constructor of the superclass, you need to use the super keyword as shown below.

super(values);

Example

```
class Superclass {
int age;
Superclass(int age) {
this.age = age;
  }
public void getAge() {
System.out.println("The age is: " +age);
  }
}
public class Subclass extends Superclass {
  Subclass(int age) {
    super(age);
  }
  public static void main(String argd[]) {
    Subclass s = new Subclass(24);
    s.getAge();
  }
}
```

2.2.2 Use of Keyword Super

The **super** keyword is similar to **this** keyword. Following are the scenarios where the super keyword is used.

- It is used to **differentiate the members** of superclass from the members of subclass, if they have same names.
- It is used to **invoke the superclass** constructor from subclass.

2.3 OBJECT CLASS IN JAVA

Object class is present in **java.lang** package. In java there is a special class named Object. If no inheritance is specified for the classes then all those classes are subclass of the Object class. In other words, Object is a superclass of all other classes by default. Hence

public clas A{….} is equals to ***public class A extends Object {….}***

Methods of Object class

The Object class provides many methods. They are as follows:

1. ***protected Object clone()*** This method creates and returns a copy of this object.

2. ***boolean equals(Object obj)*** This method indicates whether some other object is "equal to" this one.
3. ***protected void finalize()*** This method is called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
4. ***Class<?> getClass()*** This method returns the runtime class of this Object.
5. ***int hashCode()*** This method returns a hash code value for the object.
6. ***void notify()*** This method wakes up a single thread that is waiting on this object's monitor.
7. ***void notifyAll()*** This method wakes up all threads that are waiting on this object's monitor.
8. ***String toString()*** This method returns a string representation of the object.
9. ***void wait()*** This method causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object.
10. ***void wait(long timeout)*** This method causes the current thread to wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.
11. ***void wait(long timeout, int nanos)*** This method causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object, or some other thread interrupts the current thread, or a certain amount of real time has elapsed.

Example: Object Clone

```
import java.util.GregorianCalendar;
public class ObjectDemo {
   public static void main(String[] args
      GregorianCalendar cal = new GregorianCalendar();
      GregorianCalendar y = (GregorianCalendar) cal.clone();
      System.out.println("" + cal.getTime());
      System.out.println("" + y.getTime());
   }
}
```

```
Output
Mon Sep 17 04:51:41 EEST 2017
Mon Sep 17 04:51:41 EEST 2017
```

Example: *boolean equals(Object obj)*

```
public class ObjectDemo {
   public static void main(String[] args) {
      Integer x = new Integer(50);
      Float y = new Float(50f);
      System.out.println("" + x.equals(y));
      System.out.println("" + x.equals(50));
   }
}
```

```
Output
false
true
```

Example: *protected void finalize()*

```
import java.util.*;
public class ObjectDemo extends GregorianCalendar {
   public static void main(String[] args) {
      try {
         ObjectDemo cal = new ObjectDemo();
         System.out.println("" + cal.getTim
         System.out.println("Finalizing..."
         cal.finalize();
         System.out.println("Finalized.");
      } catch (Throwable ex) {
         ex.printStackTrace();
      }   }}
```

```
Output
Sat Sep 22 00:27:21 EEST 2012
Finalizing...
Finalized.
```

2.4 ABSTRACT CLASS & METHOD

A class that is declared with abstract keyword, is known as abstract class in java. It can have abstract and non-abstract methods (method with body).

Example abstract class

```
abstract class A{}
```

Abstract method

A method that is declared as abstract and does not have implementation is known as abstract method.

```
abstract void printStatus();//no body and abstract
```

Example of abstract class that has abstract method

In this example, Bike the abstract class that contains only one abstract method run. It implementation is provided by the Honda class.

```
abstract class Bike
{
      abstract void run();
}
class Honda4 extends Bike
{
void run()
{
System.out.println("running safely..");
}
public static void main(String args[])
{
Bike obj = new Honda4();
obj.run();
}
}
```

Points to Remember about abstract classes and abstract methods

- An ***abstract*** method must be present in an ***abstract*** class only. It should not be present in a non-abstract class.
- In all the non-abstract subclasses extended from an abstract superclass all the abstract methods must be implemented. An un-implemented abstract method in the subclass is not allowed.
- Abstract class cannot be instantiated using the new operator.
- A constructor of an abstract class can be defined and can be invoked by the subclasses.
- A class that contains abstract method must be ***abstract***, but the abstract class may not contain an abstract method.
- A subclass can be abstract but the super class can be concrete.

2.5 FINAL METHODS AND CLASSES

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1. variable
2. method
3. class

2.5.1 Java final variable

If you make any variable as final, you cannot change the value of final variable(It will be constant).

```
class Bike9{
final int speedlimit=90;
void run(){
speedlimit=400;
}
public static void main(String args[]){
Bike9 obj=new  Bike9();
obj.run();
}
}
```

Output:Compile Time Error

2.5.2 Java final method

If you make any method as final, you cannot override it.

```
class Bike{
final void run()
{
System.out.println("running");}
}
class Honda extends Bike
{
void run()
{
System.out.println("running safely with 100kmph");
}
public static void main(String args[]){
Honda honda= new Honda();
honda.run();
 }
}
```

Output:Compile Time Error

2.5.3 Java final class

If you make any class as final, you cannot extend it

```
final class Bike{ }
class Honda1 extends Bike{
void run()
{
System.out.println("running safely with 100kmph");
}
public static void main(String args[]){
Honda1 honda= new Honda1();
honda.run();
}
}
```

Output:Compile Time Error

2.6 INTERFACE

- An interface is a pure abstract class.
- An interface is similar to class, it have set of data members and methods.
- But methods are not implemented in interface; it must be implemented in class using ***implements*** keyword

Syntax for Interface Declaration:

```
interface interfacename
{
Type variable1=value;
Return_type method_name1(Arument list);
}
```

Syntax for Implementing Interface

```
class classname implements interfacename
{
//body of the class
}
```

Example Program-1 or Example for Multiple Inheritance

```
import java.io.*;
interface A
{
void display1();
}
interface B
{
void display2();
}
class C implements A,B
{
void display1()
{
System.out.println("This is from interface A ");
}
```

```
void display2()
{
System.out.println("This is from interface B ");
}
}
class Demo
{
public static void main(String args[])throws IOException
{
C obj = new C();
obj.display1();
obj.display1();
}}
```

2.6.1EXTENDING INTERFACES

An interface can extend another interface in the same way that a class can extend another class. The **extends** keyword is used to extend an interface, and the child interface inherits the methods of the parent interface.

Example:

```
interface Printable{
void print();
}
interface Showable extends Printable{
void show();
}
class TestInterface4 implements Showable{
public void print(){System.out.println("Hello");}
public void show(){System.out.println("Welcome");}

public static void main(String args[]){
TestInterface4 obj = new TestInterface4();
obj.print();
obj.show();
 }
}
```

2.6.2 DIFFERENCES BETWEEN CLASSES AND INTERFACES

Comparison	Class	Interface
Basic	A class is instantiated to create objects.	An interface can never be instantiated as the methods are unable to perform any action on invoking.
Keyword	class	interface
Access specifier	The members of a class can be private, public or protected.	The members of an interface are always public.
Methods	The methods of a class are defined to perform a specific action.	The methods in an interface are purely abstract.
Implement/Extend	A class can implement any number of interface and can extend only one class.	An interface can extend multiple interfaces but can not implement any interface.
Constructor	A class can have constructors to initialize the variables.	An interface can never have a constructor as there is hardly any variable to initialize.

2.6.3 Nested or Inner interfaces in Java

An interface i.e. declared within another interface or class is known as nested interface. The nested interfaces are used to group related interfaces so that they can be easy to maintain. The nested interface must be referred by the outer interface or class. It can't be accessed directly. We can only call the nested interface by using outer class or outer interface name followed by dot(.), followed by the interface name.

Points to remember for nested interfaces

There are given some points that should be remembered by the java programmer.

- Nested interface must be public if it is declared inside the interface but it can have any access modifier if declared within the class.
- Nested interfaces are declared static implicitely.

Example 1: Nested interface declared inside another interface

```
interface MyInterfaceA{
    void display();
    interface MyInterfaceB{
        void myMethod();
    }
}

class NestedInterfaceDemo1
    implements MyInterfaceA.MyInterfaceB{
     public void myMethod(){
         System.out.println("Nested interface method");
     }

     public static void main(String args[]){
         MyInterfaceA.MyInterfaceB obj=
                 new NestedInterfaceDemo1();
      obj.myMethod();
     }
}
```

Output:

Nested interface method

Example 2: Nested interface declared inside a class

```
class MyClass{
    interface MyInterfaceB{
        void myMethod();
    }
}

class NestedInterfaceDemo2 implements MyClass.MyInterfaceB{
     public void myMethod(){
         System.out.println("Nested interface method");
     }

     public static void main(String args[]){
        MyClass.MyInterfaceB obj=
               new NestedInterfaceDemo2();
        obj.myMethod();
     }
}
```

Output:

Nested interface method

2.7 OBJECT CLONING

Object cloning refers to creation of exact copy of an object. It creates a new instance of the class of current object and initializes all its fields with exactly the contents of the corresponding fields of this object. The **java.lang.Cloneable interface** must be implemented by the class whose object clone we want to create. If we don't implement Cloneable interface, clone() method generates **CloneNotSupportedException**

Advantage of Object cloning

- You don't need to write lengthy and repetitive codes. Just use an abstract class with a 4- or 5-line long clone() method.
- Clone() is the fastest way to copy array.

Disadvantage of Object cloning

- To use the Object.clone() method, we have to change a lot of syntaxes to our code, like implementing a Cloneable interface, defining the clone() method and handling CloneNotSupportedException, and finally, calling Object.clone() etc.
- We have to implement cloneable interface while it doesn?t have any methods in it. We just have to use it to tell the JVM that we can perform clone() on our object.
- Object.clone() is protected, so we have to provide our own clone() and indirectly call Object.clone() from it.
- Object.clone() doesn?t invoke any constructor so we don?t have any control over object construction.
- If you want to write a clone method in a child class then all of its superclasses should define the clone() method in them or inherit it from another parent class. Otherwise, the super.clone() chain will fail.
- Object.clone() supports only shallow copying but we will need to override it if we need deep cloning.

```
class Student18 implements Cloneable{
    int rollno;
    String name;
    Student18(int rollno,String name){
    this.rollno=rollno;
    this.name=name;
    }
    public Object clone()throws CloneNotSupportedException{
    return super.clone();
    }
    public static void main(String args[]){
    try{
    Student18 s1=new Student18(101,"amit");
    Student18 s2=(Student18)s1.clone();
    System.out.println(s1.rollno+" "+s1.name);
    System.out.println(s2.rollno+" "+s2.name);
    }catch(CloneNotSupportedException c){}
    }  }
```

```
Output:
101 amit
101 amit
```

Shallow Copy	Deep Copy
Cloned Object and original object are not 100% disjoint.	Cloned Object and original object are 100% disjoint.
Any changes made to cloned object will be reflected in original object or vice versa.	Any changes made to cloned object will not be reflected in original object or vice versa.
Default version of clone method creates the shallow copy of an object.	To create the deep copy of an object, you have to override clone method.
Shallow copy is preferred if an object has only primitive fields.	Deep copy is preferred if an object has references to other objects as fields.
Shallow copy is fast and also less expensive	Deep copy is slow and very expensive.

2.8 INNER CLASSES (NON-STATIC NESTED CLASSES)

Inner classes are of three types depending on how and where you define them. They are

- Inner Class
- Method-local Inner Class
- Anonymous Inner Class

2.8.1 INNER CLASS

Creating an inner class is quite simple. You just need to write a class within a class. Unlike a class, an inner class can be private and once you declare an inner class private, it cannot be accessed from an object outside the class.

Example

```
class Outer_Demo {
int num;
private class Inner_Demo {
public void print() {
System.out.println("This is an inner class");
}
}
void display_Inner() {
Inner_Demo inner = new Inner_Demo();
inner.print();
}
}
public class My_class {
public static void main(String args[]) {
Outer_Demo outer = new Outer_Demo();
outer.display_Inner();
}
}
```

Output : `This is an inner class.`

2.8.2 Method-local Inner Class

In Java, we can write a class within a method and this will be a local type. A method-local inner class can be instantiated only within the method where the inner class is defined.

Example

```
public class Outerclass {
void my_Method() {
int num = 23;
class MethodInner_Demo {
public void print() {
System.out.println("This is method inner class "+num);
}
}
MethodInner_Demo inner = new MethodInner_Demo();
inner.print();
}
public static void main(String args[]) {
Outerclass outer = new Outerclass();
outer.my_Method();
}}
```

Output This is method inner class 23

2.8.3 Anonymous Inner Class

An inner class declared without a class name is known as an **anonymous inner class**. In case of anonymous inner classes, we declare and instantiate them at the same time. Generally, they are used whenever you need to override the method of a class or an interface. The syntax of an anonymous inner class is as follows −

Syntax

```
AnonymousInner an_inner = new AnonymousInner() {
   public void my_method() {
      ........
   }
};
```

Example

```
abstract class AnonymousInner {
public abstract void mymethod();
}
public class Outer_class {
public static void main(String args[]) {
AnonymousInner inner = new AnonymousInner()
{
public void mymethod() {
System.out.println("This is an example of anonymous inner class");
}};
inner.mymethod();
}}
```

2.8.4 Static Nested Class

A static inner class is a nested class which is a static member of the outer class. It can be accessed without instantiating the outer class, using other static members. Just like static members, a static nested class does not have access to the instance variables and methods of the outer class. The syntax of static nested class is as follows −

Example

```
public class Outer {
static class Nested_Demo {
public void my_method() {
System.out.println("This is my nested class");
}
}
public static void main(String args[]) {
Outer.Nested_Demo nested = new Outer.Nested_Demo();
nested.my_method();
}
}
```

Output `This is my nested class`

2.9 ARRAY LISTS

Java ArrayList class uses a dynamic array for storing the elements. It inherits AbstractList class and implements List interface.

Constructors of Java ArrayList

- ArrayList( )
- ArrayList(Collection c)
- ArrayList(int capacity)

Methods of Java ArrayList

1. void add(int index, Object element)
2. boolean add(Object o)
3. boolean addAll(Collection c)
4. boolean addAll(int index, Collection c)
5. void clear()
6. Object clone()
7. boolean contains(Object o)
8. void ensureCapacity(int minCapacity)
9. Object get(int index)
10. int indexOf(Object o)
11. int lastIndexOf(Object o).
12. Object remove(int index)
13. protected void removeRange(int fromIndex, int toIndex)
14. Object set(int index, Object element)
15. int size()
16. Object[] toArray()
17. Object[] toArray(Object[] a)
18. void trimToSize()

Example

```
import java.util.*;
class TestCollection1{
public static void main(String args[]){
ArrayList<String>list=new ArrayList<String>();
list.add("Ravi");//Adding object in arraylist
list.add("Vijay");
list.add("Ravi");
list.add(1,"Ajay");
ArrayList<String>al2=new ArrayList<String>();
al2.add("Sonoo");
al2.add("Hanumat");
al.addAll(al2); //adding second list in first list
Iterator itr=list.iterator();
while(itr.hasNext()){
System.out.println(itr.next());
for(String obj:al)
System.out.println(obj);
}
}
}
```

OUTPUT:

```
Ajay
Ravi
Vijay
Ravi
Sonoo
Hanumat
```

2.10 STRINGS

- Generally, string is a sequence of characters. But in java, string is an object that represents a sequence of characters. String class is used to create string object.
- Java String provides a lot of concepts that can be performed on a string such as compare, concat, equals, split, length, replace, compareTo, intern, substring etc
- Strings can be create using string class.
- Example: **String s1="Welcome";**
- Strings can also be created using constructors

  String s=new String("Welcome");

2.10.1 Java String class methods

The java.lang.String class provides many useful methods to perform operations on sequence of char values.
`

No.	Method	Description
1	char charAt(int index)	returns char value for the particular index
2	int length()	returns string length
3	String substring(int beginIndex)	returns substring for given begin index
4	String substring(int beginIndex, int endIndex)	returns substring for given begin index and end index
5	boolean equals(Object another)	checks the equality of string with object
6	boolean isEmpty()	checks if string is empty
7	String concat(String str)	concatinates specified string
8	String replace(char old, char new)	replaces all occurrences of specified char value
9	String replace(CharSequence old, CharSequence new)	replaces all occurrences of specified CharSequence
10	String trim()	returns trimmed string omitting leading and trailing spaces
11	String toLowerCase(String str)	Covert lowercase to uppercase.
12	String toUpperCase(String str)	Covert uppercase to lowercase.

Example:

```
import java.io.*;
import java.lang.String;
public class Demo
{
   public static void main(String[] args)
{
   String obj = " Hello Java";
   System.out.println(obj.toLowerCase());
   System.out.println(obj.toUpperCase());
   System.out.println(obj.length());
   System.out.println(obj.trim());
   System.out.println(obj.length());
   System.out.println(obj.charAt(7));
   System.out.println(obj.replace("Java", "World"));
   System.out.println(obj.substring(6, 10));
  }
```

OUTPUT:

```
C:\Program Files\Java\jdk1.5.0\bin>java Demo
 hello java
 HELLO JAVA
11
Hello Java
11
J
 Hello World
 Jav
```

}

2.10.2 STRING BUFFER CLASS

The **StringBuffer** is a class which is alternative to the String class. But **StringBuffer** class is more flexible to use than **String** class.That means, using **StringBuffer** we can insert some component to the existing string or modify the existing string but in case of **String** class once the string is defined then it remains fixed.

Java StringBuffer class methods

No.	Method	Description
1	append(String str)	Appends the String to the buffer
2	capacity()	It returns the capacity of the string buffer
3	insert(int offset, char ch)	It insert the character at the position specified by the offset
4	replace(int Start,int end, String str)	It replaces the character specified by the new string
5	delete(int start,int end)	It deletes the character from the string specified by the starting and ending index.
6	reverse()	The character sequence is reversed
7	length()	It returns the length of the string buffer
8	charAt(int index)	It returns a specific character from the sequence which is specified by the index.
9	setCharAt(int index, char ch)	The character specified by the index from the stringbuffer is set to ch
10	setLengthIint new_len)	It sets the length of the string buffer.

Example 1:

```
import java.io.*;
import java.lang.*;
class Demo
{
public static void main(String[] args)
{
StringBuffer obj = new StringBuffer("Hello ");
System.out.println(obj.length());
System.out.println(obj.append("Java"));
System.out.println(obj.length());
System.out.println(obj.delete(0,5));
System.out.println(obj.insert(0,"Hello"));
System.out.println(obj.charAt(7));
System.out.println(obj.replace(6,10,"World"));
System.out.println(obj.reverse());
}
}
```

OUTPUT:

```
C:\Program Files\Java\jdk1.5.0\bin>java Demo
 6
Hello Java
10
 Java
Hello Java
a
Hello World
dlroW olleH
```

Write a java program to create a student examination database system that prints the mark sheet of students. Input student name, marks in 6 subjects . This mark should be between 0 and 100

If the average of marks is >= 80 then print Grade 'A'.
If the average is <80 and >=60 then prints Grade 'B'.
If the average is <60 and >=40 then prints Grade 'C'
Else print Grade 'D'.

```
import java.util.Scanner;
public class JavaProgram
{
    public static void main(String args[])
    {
        int mark[] = new int[6];
        int i;
        float sum=0, avg;
        Scanner scan = new Scanner(System.in);

        System.out.print("Enter the Name: ");
        String Name = scan.nextLine();

        System.out.println("Enter Marks Obtained in 5 Subjects : ");
        for(i=0; i<6; i++)
        {
            mark[i] = scan.nextInt();
            sum = sum + mark[i];
        }

        avg = sum/6;

        System.out.println("Name: "+ Name);

        System.out.print("Your Grade is ");
        if(avg>80)
        {
            System.out.print("A");
        }
        else if(avg>60 && avg<=80)
        {
            System.out.print("B");
        }
        else if(avg>40 && avg<=60)
        {
            System.out.print("C");
        }
        else
        {
            System.out.print("D");
        }
    }
}
```

```
Enter the Name: Arun
Enter Marks Obtained in 5 Subjects :
50
60
70
80
90
95
Name: Arun
Your Grade is B
```

CS8391 OBJECT ORIENTED PROGRAMMING

Question bank

UNIT –II Part-A

1. What is meant by Inheritance Hierarchy? Give an example. N/D 2010
2. What are Inheritance Hierarchy? A/M 2011
3. Define Inheritance Hierarchy? N/D 2011
4. Define Inheritance. A/M 2012
5. What are the conditions to be satisfied while declaring abstract classes? A/M 2012
6. What is meant by abstract class? N/D 2014
7. What is abstract class? A/M 2015
8. Mention the purpose of finalize method. A/M 2012
9. Define finalize method. A/M 2013
10. What is the use of final keyword? N/D 2012
11. What is the use of final keyword? A/M 2013
12. What is the use of final keyword? A/M 2016

13. In Java what is the use of Interface? N/D 2010
14. What is Interface? N/D 2011
15. Define interface and write the syntax of the interface. N/D 2012
16. Define interface and write the syntax of interface. A/M 2016
17. How to object clone? A/M 2013
18. What is meant by object cloning? N/D 2013
19. What is the role of Clone() method in Java? A/M 2014
20. What is meant by object cloning? **N/D 2015**
21. What is object cloning? **A/M 2018**
22. Define dynamic binding**. N/D 2015**
23. What is dynamic binding? **A/M 2018**
24. Define static Inner class. **A/M 2011**
25. Define Inner class. Why would you want to do that. **A/M 2015**

2017 Regulation

1. What is object cloning? **N/D 2018**
2. What is class hierarchy? Give example. **N/D 2018**
3. State the conditions for method overriding in Java. **A/M 2019**
4. Write the syntax for importing packages in a Java source file and give an example. **A/M 2019**

PART-B

1. Define Polymorphism. [Marks 2] **N/D 2010**
2. Explain the concept of polymorphism with a example. (8) N/D 2011
3. What is meant by polymorphism? Discuss the types of polymorphism with suitable examples. (16) **N/D 2014**

4. Explain Inheritance and class hierarchy. [Marks 8] **N/D 2010**
5. State the design hints for Inheritance (8) **A/M 2011**
6. Explain the concept of inheritance with suitable example. (8) **N/D 2011**
7. Give elaborate discussion on inheritance. (16) A/M 2012
8. What is inheritance? Write a program for inheriting a class. (8) **A/M 2013**
9. Describe briefly about inheritance. (8) **A/M 2014**
10. What is class hierarchy? Explain its types with suitable example. (8) **A/M 2015**
11. Discuss the advantage of inheritance. (4) **A/M 2018**

12. Write briefly on Abstract classes with an example. [Marks 6] **N/D 2010**
13. Write in detail about abstract class. (8) **N/D 2012**
14. What is abstract class? Write a program for abstract class example. (8) **A/M 2013**
15. Explain the concept of abstract class with example. (8) **N/D 2013**
16. Explain briefly abstract class and method. (8) **A/M 2014**
17. What is abstract class? State the purpose of it. (8) **N/D 2015**
18. Write in detail about the following: i)Abstract classes (8) **A/M 2016**
19. Write short notes on abstract classes (8) **A/M 2018**

20. Describe the finalize method with an example (8) **A/M 2011**
21. Write note on final keyword. (4) **N/D 2013**
22. What is final keyword explain with an example. (8) **A/M 2015**
23. Describe the need to declare the method as final (4) **A/M 2018**
24. Explain dynamic binding and final keyword with an example. (16) **N/D 2012**
25. Explain Dynamic Binding and Final keyword with example. (16) **A/M 2016**

26. Explain the concept of dynamic binding with a suitable example. (8) **A/M 2018**

27. Differentiate method overloading and method overriding. Explain both with an example. **A/M 2012**
28. State the properties of interface. (8) **A/M 2011**
29. Write in detail about interface (8) **N/D 2012`**
30. What is meant by interface? How it is declared and implement in Java. Give example. (12) **N/D 2013**
31. Explain in detail about the term interface and list out its properties. (8) **A/M 2015**
32. Describe the concept of reflection and interface with example. (16) **N/D 2015**
33. Write in detail about the following:. ii)Interface (16) **A/M 2016**
34. Explain how interface are handled in java with suitable example. (8) **A/M 2018**

35. Explain the following with examples: The clone able interface & The property interface. **N/D 2010**
36. What is meant by object cloning? Explain with an example. (8) **N/D 2011**
37. Explain the concept of object cloning and inner classes with example. (16) **N/D 2014**

38. What is a static Inner class? Explain with example. [Marks 8] **N/D 2010**
39. Discuss in detail about inner class, with its usefulness. (8) **N/D 2011**
40. Define inner classes. How to access object state using inner classes? Give an example. (8) **A/M 2013**
41. Discuss the object and inner classes with example. (8) **N/D 2013**
42. Describe in detail about inner classes in Java. (8) **A/M 2014**
43. Illustrate the concept of inner class with example. (8). **N/D 2015**

44. Explain the following in Strings: Concatenation & Substrings. [Marks 4] **N/D 2010**
45. Explain any four methods available in string handling. [Marks 4] **N/D 2010**
46. Explain any four string operation in java with an example (8) **N/D 2011**
47. Explain string handling classes in java with example. (16) **A/M 2012**
48. Discuss about string. (8) **N/D 2012**
49. Explain java building string function with an example. (8) **A/M 2013**
50. Write a Java program to reverse the given number. (6) **N/D 2013**
51. String in Java (3) **A/M 2015 A/M 2016**

2017 Regulation

N/D 2018

1. Define Inheritance. With diagrammatic illustration and java programs illustrate the different types of inheritance with an example**. (13)**
2. Write a java program to create a student examination database system that prints the mark sheet of students. Input student name, marks in 6 subjects . This mark should be between 0 and 100
   If the average of marks is >= 80 then print Grade 'A'.
   If the average is <80 and >=60 then prints Grade 'B'.
   If the average is <60 and >=40 then prints Grade 'C'
   Else print Grade 'D'.

A/M 2019

3. Explain hierarchical and multi level inheritance supported by Java and demonstrate the execution order of constructors in these types. **(13)**
4. Explain simple interface and nested interface with example. **(7)**
5. Presented a detailed comparison between classes and interfaces. **(6)**