CS 8392 OBJECT ORIENTED PROGRAMMING

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, ArrayLists – Strings

Question: 1 a) What is Inheritance?(2)
Or
1 b) Define Inheritance.(2)

Answer:

Inheritance in java is a mechanism in which one object acquires all the properties and behaviors of parent object. It is an important part of OPPs(Object Oriented programming system).

The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class). Inheritance represents the IS-A relationship, also known as *parent-child* relationship.

Question: 2 a) Why inheritance is used in java?(2)
Or
2 b) What is the use of inheritance in java?(2)

Answer:

- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

Question: 3 a) What are the terms used in inheritance?(2)
Or
3 b) List out the terms used in inheritance?(2)

Answer:

- **Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.

- **Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
- **Super Class/Parent Class:** Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
- **Reusability:** As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class.

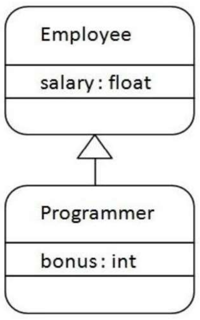
Question: 4 a) Write the syntax of Java inheritance with one example?(4)
Or
4 b) Give the syntax of java inheritance with one example?(4)

Answer:

```
class Subclass-name extends Superclass-name
{
        //methods and fields
}
```

The extends keyword indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality. In the terminology of Java, a class which is inherited is called parent or super class and the new class is called child or subclass.

Java Inheritance **Example**



As displayed in the above figure, Programmer is the subclass and Employee is the superclass. Relationship between two classes isProgrammer IS-A Employee.It means that Programmer is a type of Employee.

```
class Employee
{
                        float salary=40000;
 }
 class Programmer extends Employee
{
         int bonus=10000;
 public static void main(String args[])
        {
                Programmer p=new Programmer();
                System.out.println("Programmer salary is:"+p.salary);
                System.out.println("Bonus of Programmer is:"+p.bonus);
        }
}
```

Output

```
        Programmer salary is:40000.0
        Bonus of programmer is:10000
```

In the above example, Programmer object can access the field of own class as well as of Employee class i.e. code reusability.

Question: 5 a) Define inheritance. With diagrammatic illustrations and java programs illustrate the different types of inheritance with an example (13)

Or

5 b) Explain single, hierarchical and multi-level inheritance supported by java(13)

Answer:

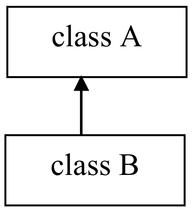
Inheritance in java is a mechanism in which one object acquires all the properties and behaviors of parent object

TYPES OF INHERITANCE IN JAVA

On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical. In java programming, multiple and hybrid inheritance is supported through interface only.

Single Inheritance

A class derived from one base class is known as Single Inheritance



Subclass B is derived from superclass A. class B acquires all the properties of class A

Syntax:

```
public class A
{
        ------
}
public class B extends A
{
        -----
}
```

File: TestInheritance.java

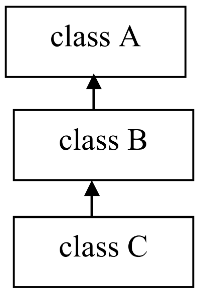
```
class Animal{
        void eat( )
           {
                System.out.println("eating...");
           }
        }
class Dog extends Animal{
        void bark( )
                {
```

```
                System.out.println("barking...");
            }
        }
class TestInheritance{
        public static void main(String args[])
        {
                Dog d=new Dog();
                d.bark();
                d.eat();
        }
}
```

Output: barking... eating...

Multilevel Inheritance

A class derived from other derived class is known as multilevel inheritance.



Syntax:

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

File: TestInheritance2.java

```
class Animal{
        void eat()
        {
                System.out.println("eating...");
        }
        }
class Dog extends Animal{
        void bark()
        {
                System.out.println("barking...");
        }
}
class BabyDog extends Dog{
        void weep()
        {
                System.out.println("weeping...");
        }
}
class TestInheritance2{
        public static void main(String args[])
        {
                BabyDog d=new BabyDog();
                d.weep();
                d.bark();
                d.eat();
        }
        }
```

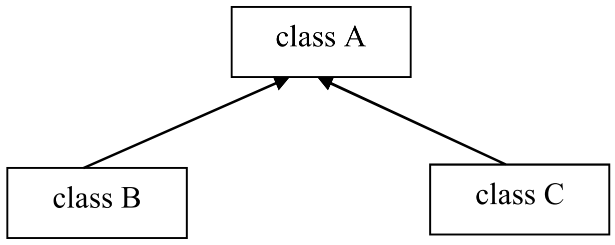
Output:

```
 weeping...
```

```
barking...
eating...
```

Hierarchical Inheritance

More than one subclasses are derived from one base class is known as Hierarchical Inheritance



Syntax:

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

File: TestInheritance3.java

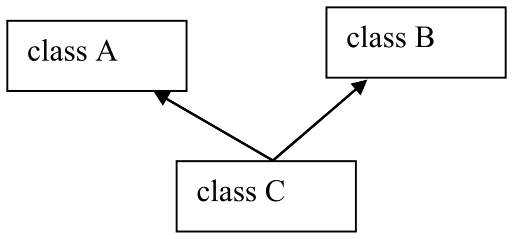
```
class Animal
{
        void eat()
        {
                System.out.println("eating...");
        }
}
```

```
class Dog extends Animal
{
        void bark()
        {
                System.out.println("barking...");
        }
}
class Cat extends Animal
{
        void meow()
        {
                System.out.println("meowing...");
        }
}
class TestInheritance3{
        public static void main(String args[])
        {
                Cat c=new Cat();
                c.meow();
                c.eat();
                //c.bark();//C.T.Error
        }
}
```

Output:

```
meowing...
eating...
```

Multiple Inheritance



When a class extends multiple classes i.e. known as multiple inheritance.

To reduce the complexity and simplify the language, multiple inheritance is not supported in java. Consider a scenario where A, B and C are three classes. The C class inherits A and B classes. If A and B classes have same method and you call it from child class object, there will be ambiguity to call method of A or B class.

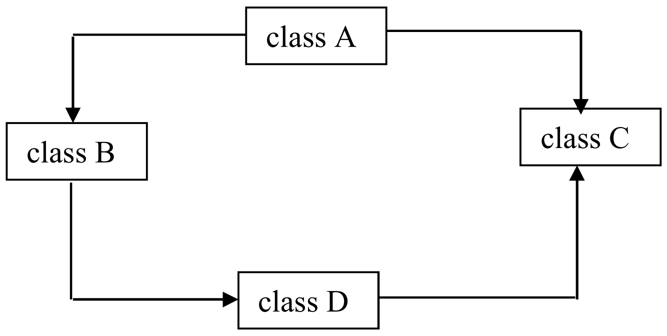
Since compile time errors are better than runtime errors, java renders compile time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error now.

Syntax:

```
public class A
{
        --------
}
public class B
{
        --------
}
public class C extends A,B
{
        ------
}// Java doesnot support multiple Inheritance
```

Hybrid Inheritance

Hybrid inheritance is a combination of Single and Multiple inheritance. A typical flow diagram would look like below. A hybrid inheritance can be achieved in the java in a same way as multiple inheritance Using interfaces. By using interfaces you can have multiple as well as hybrid inheritance in Java.



Example

```
public void dispA()
   {
      System.out.println("disp() method of ClassA");
   }
}
public class ClassB extends ClassA
{
   public void show()
   {
      System.out.println("show() method of ClassB");
   }
   public void dispB()
   {
      System.out.println("disp() method of ClassB");
   }
}
public class ClassC extends ClassA
{
   public void show()
   {
      System.out.println("show() method of ClassC");
   }
   public void dispC()
```

```
  {
     System.out.println("disp() method of ClassC");
  }
}
public class ClassD extends ClassB,ClassC
{
  public void dispD()
  {
     System.out.println("disp() method of ClassD");
  }
  public static void main(String args[])
  {
      ClassD d = new ClassD();
      d.dispD();
      d.show();//Confusion happens here which show method to call
  }
}
```

Output : Error!!

Question: 6 a) Summarize the concepts of super classes and sub classes(13)
Or
6 b) Explain in detail about super classes and sub classes(13)

Answer:

Super Keyword

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

There are three ways for using super keyword

i) Invoke the method of immediate parent class
ii) To access the class variables of immediate parent class
iii) To invoke the immediate parent class constructor

i)Invoke the variable

When both parent and child class have member with same name, we can use super keyword to access member of parent class.

Example:

```
class A
{
        int x=10;
}
class B extends A
{
        int x=20;
        void display()
        {
                System.out.println(Super.x)
        }
public static void main(String args[])
{
     B obj=new B();
     obj.display();
}
}
```

Output:

10

ii)Accessing class Methods

The super keyword can also be used to invoke parent class method. It should be used if subclass contains the same method as parent class (Method Overriding).

Example:

```
class A
{
        void display()
        {
                System.out.println("Class A");
        }
 }
class B
{
        void display()
        {
                System.out.println("classB");
        }
        void show()
        {
                super.display();
        }
public static void main(String args[])
{
        B obj=new B();
        obj.display();
        obj.show();
}
}
```

Output:

```
      class B
      class A
```

iii)Invoking the immediate parent class constructor

The super keyword can also be used to invoke the parent class constructor.

Syntax:

```
super();
```

super() if present, must always be the first statement executed inside a subclass constructor.

When we invoke a super() statement from within a subclass constructor, we are invoking the immediate super class constructor.

Example:

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

Output:

```
        Constructor of class A
        Constructor of class B
```

Question: 7 a) Discuss about protected members in Java with example.(8)
Or
7 b) Explain in detail about protected members with example.(8)

Answer:

Access Modifiers in java

There are two types of modifiers in java: access modifiers and non-access modifiers. The access modifiers in java specifies accessibility (scope) of a data member, method, constructor or class.

There are 4 types of java access modifiers:

- private
- default
- protected
- public

There are many non-access modifiers such as static, abstract, synchronized, native, volatile, transient etc.

private access modifier

The private access modifier is accessible only within class.

Default Access Modifier

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

Public Access Modifier

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

PROTECTED ACCESS MODIFIER

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

The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

Example of protected access modifier

In this example, we have created the two packages pack and mypack. The A class of pack package is public, so can be accessed from outside the package. But msg method of this package is declared as protected, so it can be accessed from outside the class only through inheritance.

```
//save by A.java
package pack;
public class A
{
        protected void msg()
        {
                System.out.println("Hello");}
        }
//save by B.java
package mypack;
import pack.*;
class B extends A
{
         public static void main(String args[])
         {
                 B obj = new B();
                 obj.msg();
         }
}  Output:Hello
```

Question: 8 a) Interpret with an example what is method overloading.(8)
Or
8 b) What is method overloading in java with an example.(8)

Answer:

METHOD OVERLOADING

If a class has multiple methods having same name but different in parameters, it is known as Method Overloading. If we have to perform only one operation, having same name of the methods increases the readability of the program.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int, int) for two parameters, and b(int, int, int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs.

So, we perform method overloading to figure out the program quickly.

Advantage of method overloading

Method *overloading* increases the readability of the program. In this example, we have created two methods, first add() method performs addition of two numbers and second add method performs addition of three numbers. In this example, we are creating static methods so that we don't need to create instance for calling methods.

```
class Adder
{
static int add(int a,int b)
{
        return a+b;
}
static int add(int a,int b,int c)
{
        return a+b+c;
}
}
class TestOverloading1
{
public static void main(String[] args)
{
        System.out.println(Adder.add(11,11));
        System.out.println(Adder.add(11,11,11));
}
}
```

Output:

```
22
33
```

Question: 9 a) Interpret with an example what is method overriding.(8)
Or
9 b) What is method overriding in java with an example.(8)

Answer:

METHOD OVERRIDING

In a class hierarchy, when a method in a subclass has the same name and type signature as a method in its superclass, then the method in the subclass is said to override the method in the superclass.

Usage of Java Method Overriding

Method overriding is used to provide specific implementation of a method that is already provided by its super class. Method overriding is used for runtime polymorphism

Example

```
class A
{
        int i,j;
        A(int a,int b)
        {
                i=a;
                j=b;
        }
        void show()
        {
                System.out.println(i+" "+j);
        }
}
class B extends A
{
        int k;
        B(int a,int b,int c)
        {
                super(a,b);
                k=c;
        }
        void show()
        {
                System.out.println(k);
        }
}
class Override
{
public static void main(String args[])
{
        B o=new B(1,2,3);
        o.show();
}
}
```

Output

```
          3
```

When show() is invoked on an object of type B, the version of show() defined within B is used. The version of show() inside B overrides the version declared in A.

To access the superclass version of an overridden method, use super.

```
class B extends A
{
            int k;
            B(int a,int b,int c)
            {
                super(a,b);
                k=c;
            }
            void show()
            {
                super.show();           //this calls A's show()
                System.out.println(k);
            }
}
```

Output:

```
            1 2
            3
```

Question: 10 a) Briefly discuss on Constuctors in subclass.(8)

Or

10 b)Explain in detail about constructors in subclass with an example.(8)

Answer:

A subclass can invoke the constructor method defined by the superclass. The syntax for the super will be

super();

or

super(parameter_list)

To invoke the constructor of superclass, super() or super(parameter_list) statement should appear in the first line of the subclass constructor.

Example:

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

```
        }
}
class B extends A
{
        int b;
        B(int i)
        {
          super(i);              // It must be on the first line of constructor of the subclass.
          this.b=i+5;
          System.out.println("b is initialised");
        }
        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 ConstrDemo
{
public static void main(String args[])
{
        B obj=new B(20);         //Note that the object of subclass is created only
        obj.show_a();
        obj.show_b();
}
}
```

Output:

```
a is initialised
b is initialized
The value of A=20
The value of B=25
The value of C=500
```

Question: 11 a) Describe in brief about object class and its methods in Java with a suitable Example.(8)

Or

11 b) Demonstrate the methods available in the object class(8)

Answer:

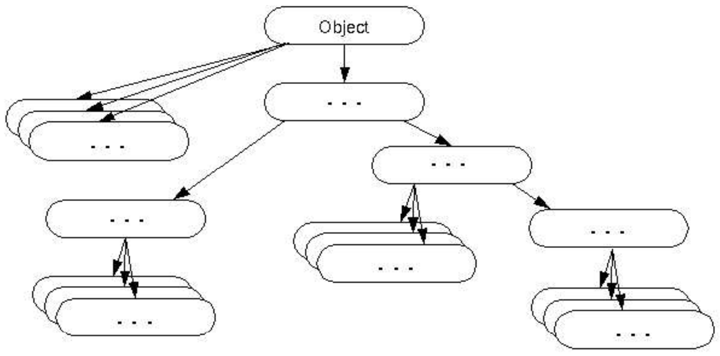
The Object class is the parent class of all the classes in java by default. It is the topmost class of java. The Object class is beneficial if you want to refer any object whose type you don't know. Notice that parent class reference variable can refer the child class object, known as upcasting.

Let's take an example, there is getObject() method that returns an object but it can be of any type like Employee,Student etc, we can use Object class reference to refer that object.

For example:

```
Object obj=getObject();
        //we don't know what object will be returned from this method
```

The Object class provides some common behaviors to all the objects such as object can be compared, object can be cloned, object can be notified etc.



Methods of Object class

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

Method	**Description**
public final Class getClass()	Obtains the class of an object at runtime
public int hashCode()	returns the hashcode number for this object.
public boolean equals(Object obj)	Determines whether one object is equal to another.
protected Object clone() throws CloneNotSupportedException	Creates a new object that is the same as the object being cloned
public String toString()	returns a string that describes the object.
public final void notify()	wakes up single thread, waiting on this object's monitor.
public final void notifyAll()	wakes up all the threads, waiting on this object's monitor.
public final void wait(long timeout)throws InterruptedException	causes the current thread to wait for the specified milliseconds, until another thread notifies (invokes notify() or notifyAll() method).
public final void wait(long timeout,int nanos)throws InterruptedException	causes the current thread to wait for the specified milliseconds and nanoseconds, until another thread notifies (invokes notify() or notifyAll() method).
public final void wait()throws InterruptedException	causes the current thread to wait, until another thread notifies (invokes notify() or notifyAll() method).

protected void finalize()throws Throwable	Called before an unused object is recycled

Example:

```
public class Test
{
public static void main(String args[])
{
        Test t=new Test();
        System.out.println(t.hashcode());
        t=null;
        System.gc();      //calling garbage Collector
        System.out.println("end");
}
protected void finalize()
{
        System.out.println("finalize method called");
}
}
```

Output:

```
        366712642
        end
        finalize method called
```

Question: 12 a)Write briefly on abstract classes and methods with an example(10)
Or
12 b) Discuss in detail about abstract classes and methods(10)

Answer:

Abstraction is a process of hiding the implementation details and showing only functionality to the user. Another way, it shows only important things to the user and hides the internal details for example sending sms, you just type the text and send the message. You don't know the internal processing about the message delivery.

Abstraction lets you focus on what the object does instead of how it does it.

Ways to achieve Abstraction

There are two ways to achieve abstraction in java

- Abstract class (0 to 100%)
- Interface (100%)

Abstract class in Java

A class that is declared as abstract is known as abstract class. It needs to be extended and its method implemented. It cannot be instantiated.

Syntax for abstract class

```
abstract class A
{
        ----
}
```

Abstract method

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

Syntax for abstract method:

```
abstract type name(parameter-list);
```

Example abstract method

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

Example of abstract class that has abstract method

```
// A Simple demonstration of abstract.
 abstract class A
{
        abstract void callme( );          // concrete methods are still allowed in abstract classes
         void callmetoo( )
         {
                System.out.println("This is a concrete method.");
        }
}
 class B extends A
{
            void callme( )
             {
                        System.out.println("B's implementation of callme.");
             }
}
 class AbstractDemo
{
             public static void main(String args[])
             {
             B b = new B( );
             b.callme( );
             b.callmetoo( );
             }
}
```

Output:

```
B's implementation of callme
This is a concrete method
```

Question: 13 a)Illustrate briefly about final methods and classes(13)
Or
13 b) Explain briefly on final keyword (13)

Answer:

The keyword final has three uses. First, it can be used to create the equivalent of a named constant. The other two uses of final apply to inheritance. Final keyword can be used along with variables, methods and classes.

1)final variable

2) final method

3) final class

1) Final variable

Final variables are constants. We cannot change the value of a final variable once it is initialized.

```
class Demo
{
        final int MAX_VALUE=99;
        void myMethod()
        {
                MAX_VALUE=101;
        }
        public static void main(String args[])
        {
                Demo obj=new  Demo();
                obj.myMethod();
        }
}
```

Output:

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        The final field Demo.MAX_VALUE cannot be assigned
```

We got a compilation error in the above program because we tried to change the value of a final variable “MAX_VALUE”.

2) Final method

To disallow a method from being overridden, specify final as a modifier at the start of its declaration A final method cannot be overridden. Which means even though a sub class can call the final method of parent class without any issues but it cannot override it.

Example:

```
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

3) Final Class :

Final class is a class that cannot be extended i.e. it cannot be inherited. e.g. int and float are final classes .

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

Question: 14 a)Explain simple interface, multiple interface and nested interface with example(13)
Or
14 b) Explain in detail about interface with examples.(13)
Or
14 c) .Describe what is meant by interface. How is interface declared and implemented in Java. Give example(13)

Answer:

An interface is similar to classes. It has static constants and abstract methods. The interface in java is a mechanism to achieve abstraction. There can be only abstract methods in the java interface not method body. It is used to achieve abstraction and multiple inheritance in Java. Any number of classes can implement an interface. Also, one class can implement any number of interfaces. Interfaces are designed to support dynamic method resolution at run time.

In other words, you can say that interfaces can have methods and variables but the methods declared in interface contain only method signature, not body. Java Interface also represents IS-A relationship. It cannot be instantiated just like abstract class.

Uses of interface

There are mainly three reasons to use interface. They are given below.

- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

Defining an interface

Interface is declared by using interface keyword. Methods that are declared in interface have no bodies. They end with a semicolon after the parameter list. Each method in an interface is also implicitly abstract, so the abstract keyword is not needed. A class that implement interface must implement all the methods declared in the interface. Variables can be declared inside of interface declarations. They are implicitly final and static. They must also be initialized. All methods and variables are implicitly public.

Syntax:

```
interface <interface_name>
{
     return-type method-name1(parameter-list);
    return-type method-name2(parameter-list);
    type final-varname1 = value;
    type final-varname2 = value;
    //------
    return-type method-nameN(parameter-list);
    type final-varnameN = value;
}
```

Example:

```
interface Animal
{
        public void eat();
        public void travel();
}
```

Implementing Interfaces

Once an interface has been defined, one or more classes can implement that interface. To implement an interface, include the implements clause in a class definition, and then create the methods defined by the interface. The methods that implement an interface must be declared public. Also, the type signature of the implementing method must match exactly the type signature specified in the interface definition.

Syntax:

```
        class classname [extends superclass] [implements interface [,interface...]]
        {
                // class-body
        }
```

Example:

```
public class Mammal implements Animal
{
        public void eat()
        {
                System.out.println("Mammal eats");
        }
        public void travel()
        {
                System.out.println("Mammal travels");
        }
}
```

Accessing Implementations Through Interface References

Any instance of any class that implements the declared interface can be referred to by such a variable

Example:

```
class Maindemo
{
        public static void main(String args[])
        {
                Mammal m = new Mammal();
                m.eat();
                m.travel();
        }
}
```

Example:

```
interface Animals
{
        void cat();
        void horse();
}
class A implements Animals
{
        public void cat()
        {
                System.out.println("Cat");
        }
        public void horse()
        {
                System.out.println("horse");
        }
}
class Maindemo
{
public static void main(String args[])
{
        A obj=new A();
        obj.cat();
        obj.horse();
}
}
```

Output:

```
        Cat
        horse
```

Multiple Interface

More than one interfaces can implement in a single class.

Syntax:

```
class classname implements interfacename1,interfacename2---
{
        -------
}
```

Example:

```
interface Animals
{
        void cat();
        void horse();
}
interface Birds
{
        void peacock();
        void parrot();
}
class A implements Animals,Birds
{
        public void cat()
        {
                System.out.println("eating");
        }
        public void horse()
        {
                System.out.println("sleeping");
```

```
          }
          public void peacock()
          {
                    System.out.println("dancing");
          }

          public void parrot()
          {
                    System.out.println("singing");
          }
}
class Maindemo
{
public static void main(String args[])
{
          A obj=new A()
          obj.cat();
          obj.horse();
          obj.peacock();
          obj.parrot();
}
}
```

Output:

```
          eating
          sleeping
          dancing
          singing
```

Nested Interface

An interface can be declared a member of a class or another interface. Such an interface is called a member interface or a nested interface. A nested interface can be declared as public, private, or protected.

Syntax:

```
          interface interfacename1
          {
          interface interfacename2
          {
                    methods;
                    variables;
          }
          }
```

Example

```
interface shape
{
        void draw();
        interface msg
        {
                void greet();
        }
}
class A implements shape.msg
{
        void draw()

        {
                System.out.println("Shape of interface");
        }
        void greet()
        {
                System.out.println("Welcome");
        }
}
class Maindemo
{
public static void main(String args[])
{
        A o=new A();
        o.draw();
        o.greet();
}
}
```

Output:

```
        Shape of interface
        Welcome
```

Question: 15 a) Express a Java program for extending interfaces(6)
Or
15 b) Explain how the interface is extended with example. (6)

Answer:

Extending interface

One interface can inherit another by use of the keyword extends. When a class implements an interface that inherits another interface, it must provide implementations for all methods defined within the interface inheritance chain.

Syntax:

```
        interface name1 extends interface name2
```

Example:

```
interface A
{
        void show();
}
interface B extends A
{
        void display();
}
class C implements B
{
        void show()
        {
                System.out.println("Java");
        }
        void display()
        {
                System.out.println("Programming");
        }
}
class Maindemo
{
public static void main(String args[])
{
        C obj=new C();
        obj.show();
        obj.display();
}
}
```

Output:

```
        Java
        Programming
```

Question: 16 a) Differentiate classes with interface in java.(6)
or
16 b) Compare Classes and interfaces in java.(6)

Answer:

Class	**Interface**
Variables can be instantiated and an object can be created	Variables cannot be instantiated and an object cannot be created
Class can contain concrete(with implementation) methods	The interface cannot contain concrete(with implementation) methods
The access specifiers used with classes are private, protected and public.	In Interface only one specifier is used- Public.
The members of a class can be constant or final	The members of interfaces are always declared as final
The class is denoted by a keyword class	The interface is denoted by a keyword interface
Methods are defined in class implementation	Methods are not defined

Question: 17 a) Explain the concept of Object cloning with an example(8)
or
17 b) Discuss object cloning in java(8)

Answer:

The object cloning is a way to create exact copy of an object. The clone() method of Object class is used to clone an 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.

The clone() method is defined in the Object class.

Syntax of the clone() method is as follows:

protected Object clone() throws CloneNotSupportedException

clone()

The clone() method saves the extra processing task for creating the exact copy of an object. If we perform it by using the new keyword, it will take a lot of processing time to be performed that is why we use object cloning.

Exception

CloneNotSupportedException − if the object's class does not support the Cloneable interface. Subclasses that override the clone method can also throw this exception to indicate that an instance cannot be cloned.

Types of Object Cloning

i)Shallow Cloning

ii)Deep Cloning

i)Shallow Cloning

In Java, when the cloning process is done by invoking the clone() method it is called Shallow Cloning. It is the default cloning process in Java where a shallow copy of the original object will be created with exact field. In case the original object has references to some other objects as fields, then only the references of that object will be cloned instead of new object creation. If you change the value of the cloned objects then it will be reflected in the original as well. Thus, shallow cloning is dependent on the original object.

ii)Deep Cloning

In this type of cloning, an exact copy of all the fields of the original object will be created. But in case, the original object has references to other objects as fields then a copy of those objects will also be created by calling the clone() method. This makes the cloned object independent of the original object and any changes made in any of the object won't be reflected on the other.

Example

```
class Employee implements Cloneable
{
        int id;
        String name;
        Employee()
        {
                id=11;
                name="Abu";
        }
protected Object clone() throws CloneNotSupportedException
{
        return super.clone();
}
public static void main(String args[])throws CloneNotSupportedException
{
        Employee e=new Employee();
        Employee e2=(Employee)e.clone();
        System.out.println(e.id+" "+e.name);
        System.out.println(e2.id+" "+e2.name);
        e2.id=34;
        System.out.println(e2.id);
        System.out.println(e.id);
}
}
```

Output:

```
        11 Abu
        11 Abu
        34
        11
```

Question: 18 a) Define inner classes. How to access object state using inner classes? Give an example.(13)
Or
18 b) Summarize in detail about inner class with its usefulness.(13)

Answer:

INNER CLASSES

Inner class are defined inside the body of another class (known as outer class). These classes can have access modifier or even can be marked as abstract and final. Inner classes have special relationship with outer class instances. This relationship allows them to have access to outer class members including private members too.

Inner classes can be defined in four different following ways as mentioned below:

1)Nested inner class
2)Method-local inner class
3)Anonymous inner class
4)Static nested class

1) Nested Inner class

Java inner class or nested class is a class which is declared inside the class or interface. We use inner classes to logically group classes and interfaces in one place so that it can be more readable and maintainable. Additionally, it can access all the members of outer class including private data members and methods.

Syntax of Inner class

```
class Java_Outer_class
{
    //code
    class Java_Inner_class
    {
         //code
    }
}
```

Advantage of java inner classes

There are basically three advantages of inner classes in java. They are as follows:

1) Nested classes represent a special type of relationship that is it can access all the members (data members and methods) of outer class including private.

2) Nested classes are used to develop more readable and maintainable code because it logically group classes and interfaces in one place only.

3) Code Optimization: It requires less code to write.

Example:

```
class Outer
{
        class Inner
        {
                public void show()
                {
                        System.out.println("Inner class method");
                }
        }
}
class Maindemo
{
public static void main(String args[])
{
Outer.Inner in=new Outer().new Inner();
in.show();
}
}
```

Output:

```
        Inner class method
```

2) Method–Local inner classes

A method local inner class is defined within a method of the enclosing class. If you want to use inner class ,you must instantiate the inner class in the same method, but after the class definition code. Only two modifiers are allowed for method-local inner class which are abstract and final. The inner class can use the local variables of the method (in which it is present), only if they are marked final.

```
class Outer
{
        void outerMethod()
        {
                System.out.println("outer method");
                class Inner         // inner class defined inside a method of outer class
                {
                        void innerMethod()
                        {
                                System.out.println("inner method");
                        }
                }                   //close inner class method
                Inner y=new Inner();
                y.innerMethod();
        }       // close inner class definition
}       //close Top level class method
class Demo
{
public static void main(String args[])
{
        Outer x=new Outer();
        x.outerMethod();
}
}
```

Output:

```
        Outer method
        Inner method
```

3) Anonymous Inner Classes

It is a type of inner class which has no name. It can be instantiated only once. It is usually declared inside a method or a code block ,a curly braces ending with semicolon. It is accessible only at the point where it is defined. It does not have a constructor simply because it does not have a name. It cannot be static

Example:

```
class Pizza
{
        public void eat()
        {
                System.out.println("pizza");
        }
}
public class Food
{
public static void main(String args[])
{
        Pizza p = new Pizza()
        {
                public void eat()
                {
                        System.out.println("anonymous pizza");
                }
        };
p.eat();
}
}
```

Output:

```
        Anonymous pizza
```

4) Static Nested Classes

A static nested classes are the inner classes marked with static modifier. Because this is static in nature so this type of inner class doesn't share any special kind of relationship with an instance of outer class. A static nested class cannot access non static members of outer class.

Example:

```
public class Outer
{
        static class Inner
        {
                public void show()
                {
                        System.out.println("java");
                }
        }
public static void main(String args[])
{
        Outer.Inner obj=new Outer.Inner();
        obj.show();
}
}
```

Output:

```
        Java
```

Question: 19 a) Develop a program to perform string operations using ArrayList (8)
or
19 b) Explain the concept of ArrayList with example(8)

ARRAY LISTS

ArrayList is a part of collection framework. It is present in java.util package. ArrayList is a variable-length array of object references. That is, an ArrayList can dynamically increase or decrease in size. Array lists are created with an initial size. When this size is exceeded, the collection is automatically enlarged. When objects are removed, the array can be shrunk.

The issue with arrays is that they are of fixed length so if it is full you cannot add any more elements to it, likewise if there are number of elements gets removed from it the memory consumption would be the same as it doesn't shrink. On the other ArrayList can dynamically grow and shrink after addition and removal of elements. Apart from these benefits ArrayList class enables us to use predefined methods of it which makes our task easy.

ArrayList has the constructors

constructor	**Description**
ArrayList()	It is used to build an empty array list.
ArrayList(Collection c)	It is used to build an array list that is initialized with the elements of the collection c.
ArrayList(int capacity)	It is used to build an array list that has the specified initial capacity.

Methods of ArrayList class

1) **add(Object o):** This method adds an object o to the arraylist.

```
obj.add("hello");
```

This statement would add a string hello in the arraylist at last position.

2) **add(int index, Object o):** It adds the object o to the array list at the given index.

```
obj.add(2, "bye");
```

It will add the string bye to the 2nd index (3rd position as the array list starts with index 0) of array list.

3) **remove(Object o):** Removes the object o from the ArrayList.

```
obj.remove("Chaitanya");
```

This statement will remove the string "Chaitanya" from the ArrayList.

4) **remove(int index):** Removes element from a given index.

```
obj.remove(3);
```

It would remove the element of index 3 (4th element of the list – List starts with o).

5) **set(int index, Object o):** Used for updating an element. It replaces the element present at the specified index with the object o.

```
obj.set(2, "Tom");
```

It would replace the 3rd element (index =2 is 3rd element) with the value Tom.

6) **int indexOf(Object o):** Gives the index of the object o. If the element is not found in the list then this method returns the value -1.

```
int pos = obj.indexOf("Tom");
```

This would give the index (position) of the string Tom in the list.

7) **Object get(int index):** It returns the object of list which is present at the specified index.

```
String str= obj.get(2);
```

Function get would return the string stored at 3rd position (index 2) and would be assigned to the string “str”. We have stored the returned value in string variable because in our example we have defined the ArrayList is of String type. If you are having integer array list then the returned value should be stored in an integer variable.

8) **int size():** It gives the size of the ArrayList – Number of elements of the list.

```
int numberofitems = obj.size();
```

9) **boolean contains(Object o):** It checks whether the given object o is present in the array list if its there then it returns true else it returns false.

```
obj.contains("Steve");
```

It would return true if the string “Steve” is present in the list else we would get false.

10) **clear():** It is used for removing all the elements of the array list in one go.

```
obj.clear();
```

ArrayList Example in Java

```
import java.util.*;
public class ArrayListExample
{
        public static void main(String args[])
        {
                /*Creation of ArrayList: I'm going to add String
                 *elements so I made it of string type */
                ArrayList<String> obj = new ArrayList<String>();
                /*This is how elements should be added to the array list*/
                 obj.add("Ajeet");
                 obj.add("Harry");
                 obj.add("Chaitanya");
```

```
                        obj.add("Steve");
                        obj.add("Anuj");
                        /* Displaying array list elements */
                        System.out.println("Currently the array list has following elements:"+obj);
                        obj.add(0, "Rahul");          //  Add element at the given index
                        obj.add(1, "Justin");
                         obj.remove("Chaitanya");  //Remove elements from array list like this
                        obj.remove("Harry");
                        System.out.println("Current array list is:"+obj);
                        obj.remove(1);                // Remove element from the given index
                        System.out.println("Current array list is:"+obj);
                }
        }
```

Output:

```
   Currently the array list has following elements:[Ajeet, Harry, Chaitanya, Steve, Anuj]
   Current array list is:[Rahul, Justin, Ajeet, Steve, Anuj]
   Current array list is:[Rahul, Ajeet, Steve, Anuj]
```

Question: 20 a) Define String? How Strings are handled in Java? Explain with code. (13)
or
21 b) Analyse with an example, how string objects are created. How it can be modified?(13)

Answer:

String is a sequence of characters. String is an object that represents a sequence of characters. The java.lang.String class is used to create string object. In java, string is basically an object that represents sequence of char values. An array of characters works same as java string.

For example: java string class provides a lot of methods to perform operations on string such as compare(), concat(), equals(), split(), length(), replace(), substring() etc. The java.lang.String class implements *Serializable*, *Comparable* and *CharSequence* interfaces.

The string objects can be created using two ways.

1. By String literal

2. By new Keyword

1) By String literal:

Java String literal is created by using double quotes. For Example:

String s="welcome”

2) By new keyword

String s=new String(“Welcome”);

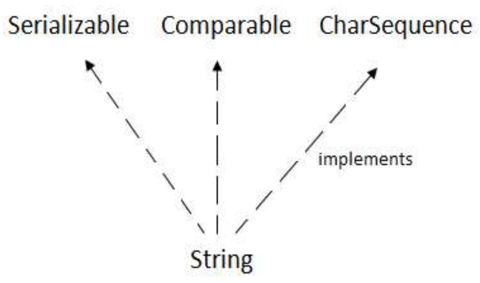
The java String is immutable i.e. it cannot be changed. Whenever we change any string, a new instance is created.

For example:

```
public static void main(String args[])
{
        String s1="java";//creating string by java string literal
        char ch[]={'s','t','r','i','n','g','s'};
        String s2=new String(ch);//converting char array to string
        String s3=new String("example");//creating java string by new keyword
        System.out.println(s1);
        System.out.println(s2);
        System.out.println(s3);
}
```

Output

```
java
strings
example
```



Java String class methods

The java.lang.String class provides a lot of methods to work on string. By the help of these methods, we can perform operations on string such as trimming, concatenating, converting, comparing, replacing strings etc.

Java String is a powerful concept because everything is treated as a string if you submit any form in window based, web based or mobile application.

i) charAt(int index)

charAt(int index) function returns the character located at the specified index.

String str = "studytonight";

System.out.println(str.charAt(2));

Output: u

Index of a String starts from 0, hence str.charAt(2) means third character of the String str.

ii) boolean equalsIgnoreCase(String s)

equalsIgnoreCase(String s) determines the equality of two Strings, ignoring their case (upper or lower case doesn't matters with this function).

String str = "java";

System.out.println(str.equalsIgnoreCase("JAVA"));

Output:

true

iii) indexOf()

indexOf() function returns the index of first occurrence of a substring or a character.

indexOf()method has four forms:

- int indexOf(String str): It returns the index within this string of the first occurrence of the specified substring.
- int indexOf(int ch, int fromIndex): It returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.
- int indexOf(int ch): It returns the index within this string of the first occurrence of the specified character.
- int indexOf(String str, int fromIndex): It returns the index within this string of the first occurrence of the specified substring, starting at the specified index.

Example:

```
public class StudyTonight
{
        public static void main(String[] args)
        {
                String str="StudyTonight";
                System.out.println(str.indexOf('u'));   //3rd form
                System.out.println(str.indexOf('t', 3));    //2nd form
                String subString="Ton";
                System.out.println(str.indexOf(subString)); //1st form
                System.out.println(str.indexOf(subString,7));   //4th form
        }
}
```

-1 indicates that the substring/Character is not found in the given String.

```
2
11
5
-1
```

iv) int length()

length() function returns the number of characters in a String.

```
        String str = "Count me";
System.out.println(str.length());
```

Output:

```
      8
```

v)replace()

replace() method replaces occurances of character with a specified new character.

Syntax:

```
        replace(char old,char new)
        String str = "Change me";
System.out.println(str.replace('m','M'));
```

Output:

Change Me

vi) substring*()*

substring() method returns a part of the string. substring() method has two forms,

```
public String substring(int beginIndex);
public String substring(int beginIndex, int endIndex);
/*
   character of begin index is inclusive and character of end index is exclusive.
*/
```

The first argument represents the starting point of the subtring. If the substring() method is called with only one argument, the substring returned, will contain characters from specified starting point to the end of original string.

But, if the call to substring() method has two arguments, the second argument specify the end point of substring.

```
String str = "0123456789";
System.out.println(str.substring(4));
```

Output:

456789

System.out.println(str.substring(4,7));

Output:

456

vii) String toLowerCase*()*

toLowerCase() method returns string with all uppercase characters converted to lowercase.

```
String str = "ABCDEF";
System.out.println(str.toLowerCase());
```

Output:

abcdef

viii) String toUpperCase*()*

This method returns string with all lowercase character changed to uppercase.

String str = "abcdef";

```
System.out.println(str.toUpperCase());
```

Output:

```
ABCDEF
```

ix) String valueOf()

Overloaded version of valueOf() method is present in String class for all primitive data types and for type Object.

valueOf() function is used to convert primitive data types into Strings.

```
public class Example
{
        public static void main(String args[])
        {
                int num = 35;
                String s1 = String.valueOf(num);    //converting int to String
                System.out.println(s1+"IAmAString");
        }
}
```

Output

```
35I
AmAString
```

But for objects, valueOf() method calls toString() function.

x) toString()

toString() method returns the string representation of the object used to invoke this method.toString() is used to represent any Java Object into a meaningful string representation. It is declared in the Object class, hence can be overrided by any java class. (Object class is super class of all java classes.)

```
public class Car
{
        public static void main(String args[])
        {
                Car c = new Car();
                System.out.println(c);
```

```
            }
            public String toString()
            {
                        return "This is my car object";
            }
}
```

Output

```
                    This is my car object
```

Whenever we will try to print any object of class Car, its toString() function will be called. toString() can also be used with normal string objects.

```
      String str = "Hello World";
      System.out.println(str.toString());
```

Output

```
    Hello World
```

If we don't override the toString() method and directly print the object, then it would print the object id.

Example:

```
public class Car
{
            public static void main(String args[])
            {
                        Car c = new Car();
                        System.out.println(c);
            }
}
```

studytonight.Car@15db9742 (here studytonight is a user-defined package).

xi) toString() with Concatenation

Whenever we concatenate any other primitive data type, or object of other classes with a String object,toString() function or valueOf() function is called automatically to change the other object or primitive type into string, for successful concatenation.

```
                    int age = 10;
```

String str = "He is" + age + "years old.";

In above case 10 will be automatically converted into string for concatenation using valueOf() function.

xii) trim()

This method returns a string from which any leading and trailing whitespaces has been removed.

String str = "   hello ";

System.out.println(str.trim());

Output

Hello

15

xiii)*getChar()*

To extract more than one character at a time, getChars( ) method is used.

Syntax:

void getChars(int sourceStart, int sourceEnd, char target[ ], int targetStart)

String s="Internationalization";

s.getChar(5,12,c,0)

Output:

national