GAIN MORE KNOWLEDGE
REACH GREATER HEIGHTS

PRESIDENCY COLLEGE
(AUTONOMOUS)
AFFILIATED TO BENGALURU CITY UNIVERSITY, APPROVED BY AICTE, DELHI & RECOGNISED BY THE GOVT. OF KARNATAKA
RE-ACCREDITED BY NAAC WITH 'A+' GRADE

UNIT – II: Inheritance and Polymorphism

Questions home1

1. Define and What are different forms of inheritance? Explain with example.10m
2. What is the method overriding? Explain 6m
3. Difference between Method overloading and Method Overriding 6m
4. What is the use of 'this' operator? 2m
5. What is the use of super keyword? 2m
6. What is dynamic binding or runtime polymorphism or dynamic method displatch. 6m
7. What is the difference between concrete class and abstract class. 6m
8. What is the importance of finalize() method. 6m
9. What is an Interface? Give example. Why to use interface??
10. Give the general form of interface with an example.
11. Explain with an example how a class implements an interface.
12. Multiple inheritance is not supported in case of class but it is supported in case of interface, Why?
13. WAP to calculate the area of circle using interface.
14. What is a package? What are the Advantage of package? What are the Types of package? Explain . 10m
15. Explain any 6 util packages. 6m
16. What is generic programming? 2m
17. What is instanceof Operator in java? 2m
18. What is casting objects in java? 2m

Inheritance in Java, Super and sub class, Overriding, Object class, Polymorphism, Dynamic binding, Generic programming, Casting objects, Instance of operator, Abstract class, Interface in java, Package in java, UTIL package, finalize method.

- *Inheritance is the process of acquiring the properties by the **sub class** (orderived class or child class) from the **super class** (or base class or parent class).*

- When a **child** class(newly defined abstraction) inherits(extends) its **parent** class(super class) , all the properties and methods of parent class becomes the member of child class.

- *In addition, child class(sub class) can add new data fields(properties) and behaviors(methods), and*

- It can override methods that are inherited from its parent class.

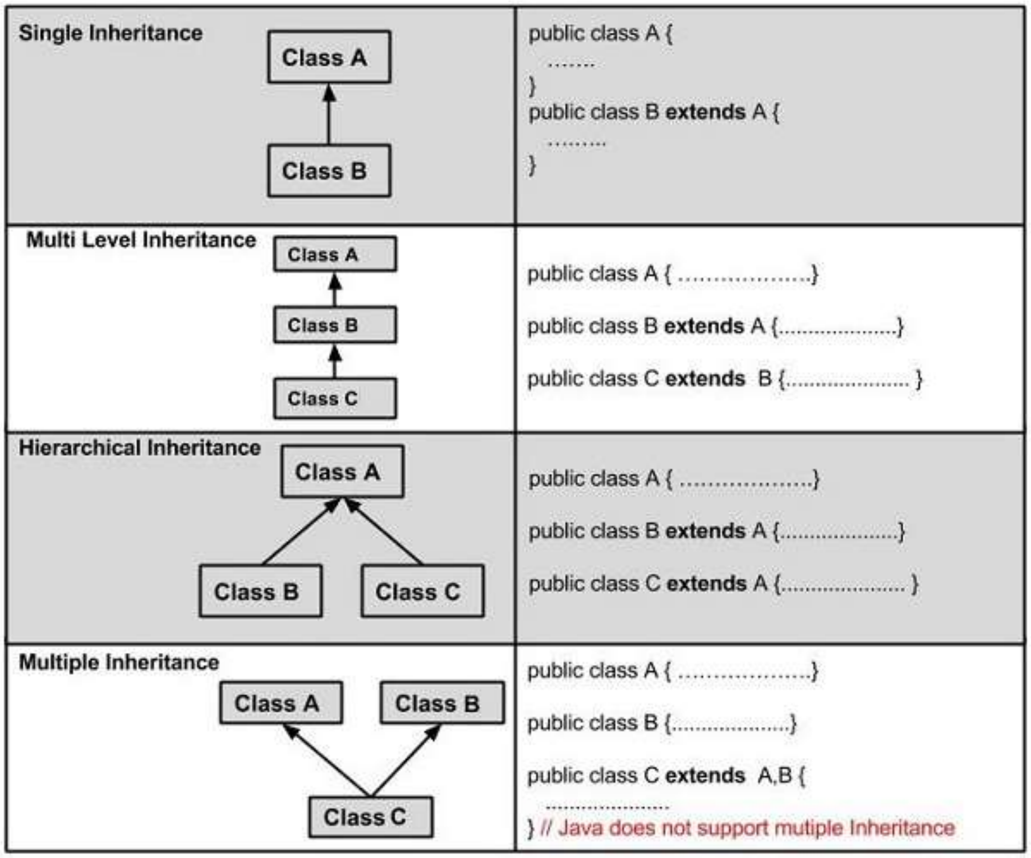
Inheritance Basics

The key word extends is used to define inheritance in Java.

Syntax:-

class subclass-name extends superclass-name

{ // body of the class
}


Class A
Class B
Class A
Class B
Class C
Class A
Class B
Class C
Class A
Class B
Class C

Single Inheritance

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

Multi Level Inheritance

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

Hierarchical Inheritance

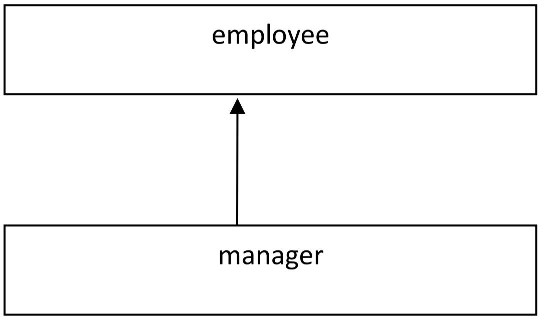
```
public class A { ...................}
public class B extends A {...................}
public class C extends A {.................... }
```

Multiple Inheritance

```
public class A { ...................}
public class B {...................}
public class C extends  A,B {
  ......................
} // Java does not support mutiple Inheritance
```

1. Single level or simple

In this type of inheritance, only one super class and one subclass are present.

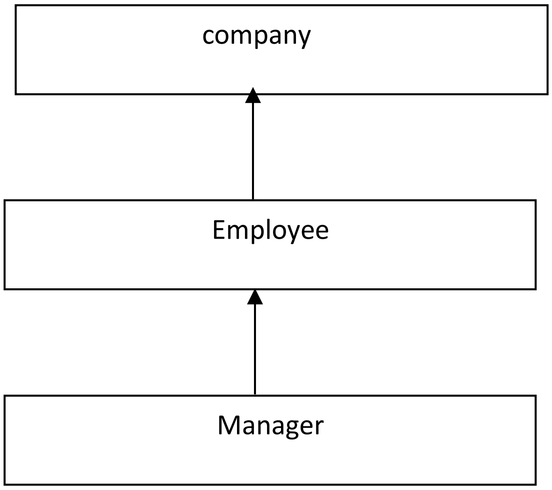


Employee class is the super class.
Manager class is sub class;

```
//code
class employee
{
  protected int a;
   -----------
    -----------
}
class manager extends emloyee
{
private
  -------------
----------------System.out.println(a)
}
```

Multilevel inheritance

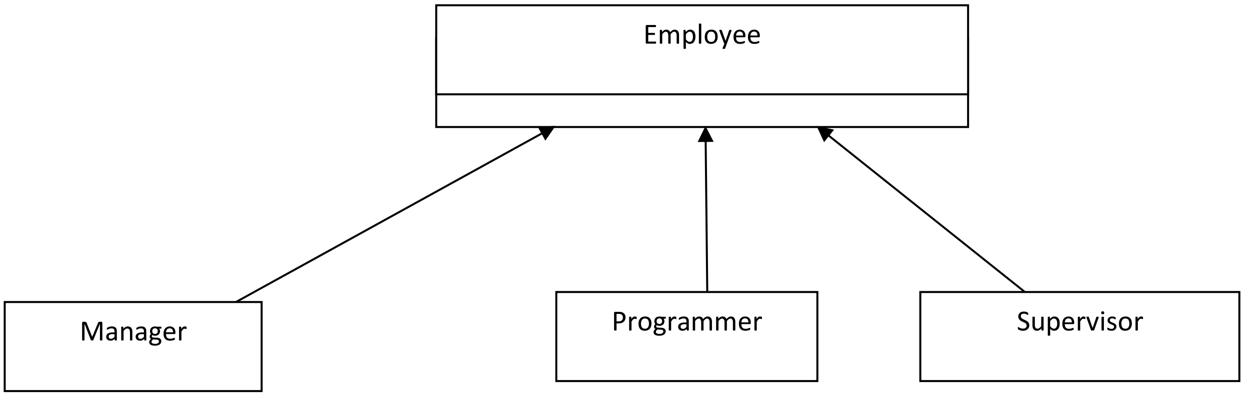
In this type, a subclass extends the features of a super class; This sub class might be extended to another subclass, so the current subclass becomes super class for another subclass. This can be extended to any level.



```
class company
{
    -----------------
   ------------------
}

class Employee extends company
{
     ----------------
   ----------------
}
class Manager extends Employee
{
 -------------------
----------------------
}
```

Hierarchical inheritance

A class can have two or more subclasses; in other words we can say, many subclasses can have a common super class.

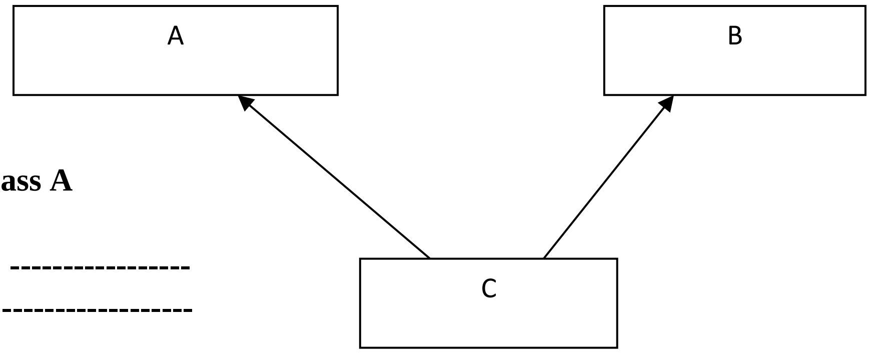


Manager, programmer and supervisor, all the three are employees; so on super class can have many subclasses. This is called Hierarchical inheritance.

```
class Employee
{
    -----------------
   ------------------
}

class Manager extends Employee
{
     ----------------
   ----------------
}
class programmer extends Employee
{
     ----------------
   ----------------
}
class Supervisor extends Employee
{
 -------------------
----------------------
}
```

Multiple inheritance

A subclass can have two or more super classes. For security reasons **java does not support multiple inheritance.** There is an indirect approach to achieve this. (through interfaces. You will learn in interface and packages chapter)

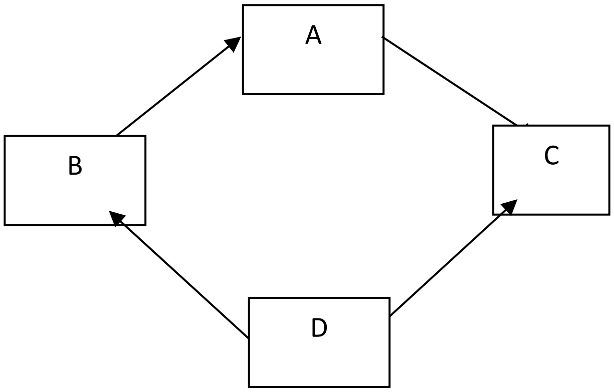


```
class A
{
    -----------------
   ------------------
}

class B
{
     ----------------
   ----------------
}
class C extends A, B //ERROR
{
 -------------------
----------------------
}
```

Note: A class can implement any number of interfaces but can extend only one class. Multiple inheritance is not supported **because it leads to deadly diamond problem**.

Hybrid Inheritance

The combination of hierarchical and multiple inheritance forms hybrid inheritance.



//Demonstration on inheritance

```
class A
{
        protected int a;
        void show1()
        {
                 System.out.println("value of a is " +a);
        }
}

class B extends A{
        void show2()
        {
                System.out.println("a="+a);
        }
}



class D  //it can't access protected members
{
  void show3()
  {
        System.out.println("value of a is " +a);
  }
}

class C
{
        public static void main(String [] args)
        {
                A a1 = new A();
                B b1 = new B();
                D d1 = new D();

                a1.a=10;
                  b1.a=20;
                 d1.a=30;

                a1.show1();
                b1.show2();
                d1.show3();
 }
```

Output: value of a is 10

```
a is 20
a is 30
```

this keyword

“this keyword is used inside the class methods to refer the current object”.

- It can be used to refer instance variable of current class
- It can be used to invoke or initiate current class constructor
- It can be passed as an argument in the method call
- It can be passed as argument in the constructor call
- It can be used to return the current class instance.

```
// Perfect Example to understand "this" keyword//
class thisdemo
{
        int x,y;
        void getdata(int x ,int y)
        {        this.x=x;
                this.y=y;
        }
        void putdata()
        {
                System.out.println("The value of x and y are "  + x +  ”and” +y);
        }
        public static void main(String args[])
        {
                int a,b;
                thisdemo    A  =    new thisdemo();
                A.getdata(2,3);
                A.putdata();

        }
}
```

```
Output:
The value of x and y are 2 and 3
```

Difference between this and super keyword

super’ is used for pointing the super class instance.	**‘this’** is used for pointing the current object.

super() is added in each class constructor automatically by compiler	**‘this’** is not added in each class constructor automatically by compiler.

super keyword

Super is a reference variable that is used to refer immediate parent class object.

Uses of Super keyword

1. Super() is used to invoke immediate parent class constructor.
2. Super() is used to invoke immediate parent class method.

```
//program with/without super keyword
class Vehicle
{
         int speed =50;
}
class Bike extends Vehicle
{
        int speed =100;

 void display()
 {
        System.out.println(speed); //will print speed of bike
         //System.out.println(super.speed); //will print speed of vehicle
 }

public static void main(String [] args)
{
   Bike b = new Bike();
   b.display();
}
} // end of  Bike class
```

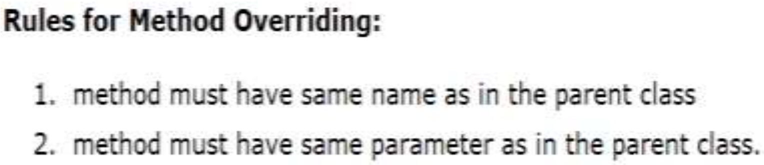
Output: 50   100

Method overriding in Java

- Having the same method in the subclass as declared in the parent class is known as method overriding.
- If a subclass provides a specific implementation of a method that is already provided by its super class, it is known as **method overriding**.

Advantage of 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



Rules for Method Overriding:

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

- In the following example, run() method is defined in the subclass which is same as parent class.
- But it has some specification. The name and parameter of the method is same and there is

IS-A relationship between the classes, so there is method overriding.

```
// PROGRAM TO DEMONSTRATE METHOD OVERRIDING
class Vehicle
{
        void run()
        {
                System.out.println("Vehicle is running");
        }
}
class Bike extends Vehicle
{
        void run()
         {
                System.out.println("Bike is running safely");
         }
        void TEST()
        {
                super.run();
                run();
         }

         public static void main(String [] args)
        {
                Bike obj = new Bike();
```

```
                    obj.TEST();
          }
}
```

OUTPUT

Vehicle is running

Bike is running safely

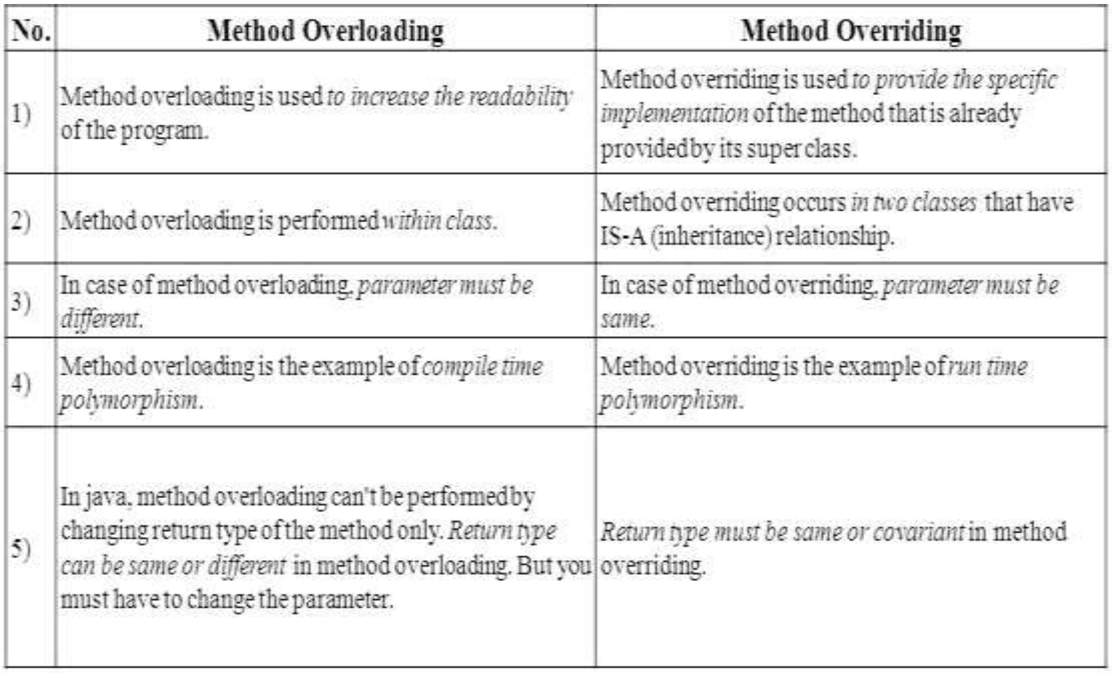
Can We override static method? 2M

No, static method cannot be overridden. It can be proved by run time polymorphism.

Why we cannot override static method? 2m

Because static method is bound with class whereas instance method is bound with object. Static belongs to class area and instance belongs to heap area.

Difference between Method overloading and Method Overriding 6m



No.	Method Overloading	Method Overriding
1)	Method overloading is used *to increase the readability* of the program.	Method overriding is used *to provide the specific implementation* of the method that is already provided by its super class.
2)	Method overloading is performed *within class.*	Method overriding occurs *in two classes* that have IS-A (inheritance) relationship.
3)	In case of method overloading, *parameter must be different.*	In case of method overriding, *parameter must be same.*
4)	Method overloading is the example of *compile time polymorphism.*	Method overriding is the example of *run time polymorphism.*
5)	In java, method overloading can't be performed by changing return type of the method only. *Return type can be same or different* in method overloading. But you must have to change the parameter.	*Return type must be same or covariant* in method overriding.

final keyword

The final keyword in java is used to restrict the user.

Final can be:

variable

method

class

1. final variable

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

```
/*there is a final variable speedlimit, we are going to change the value of this variable,
but we can't change because final variable once assigned a value can never be changed
*/

class Bike
{
            final int speedlimit =90; // final variable

          void run()
            {
                        speedlimit=400;     //error….
            }

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

}

//output: error -- cann't assign a value to final variable.......
```

2. final method

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

Example of final method

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

class Honda extends Bike
{

             void run()  // ERROR ........  run() is a final method
             {
                    System.out.println("Honda is running safely");
          }
  public static void main(String [] args)
  {
     Bike obj = new Bike();
     obj.run();
   }
}
```

error
void run() // ERROR .... run() is a final method

3) final class

If we make any class as final, we cannot extend it.

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

class Honda extends Bike  //ERROR…. Bike can't be extended
{
  public static void main(String [] args)
  {
     Honda obj = new Honda();
     obj.run();
   }
}
```

POLYMORPHISM

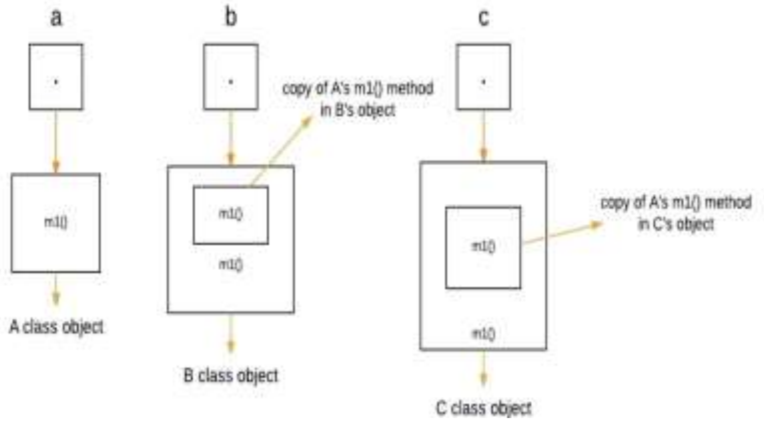
Dynamic method

DYNAMIC METHOD DISPATCH OR Runtime Polymorphism in Java

Method overriding is one of the ways in which Java supports **Runtime Polymorphism**. Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at run time, rather than compile time.

- When an overridden method is called through a superclass reference, Java determines which version(superclass/subclasses) of that method is to be executed based upon the type of the object being referred to at the time the call occurs. Thus, this determination is made at run time.

- At run-time, it depends on the type of the object being referred to (not the type of the reference variable) that determines which version of an overridden method will be executed
- A superclass reference variable can refer to a subclass object. This is also known as upcasting. Java uses this fact to resolve calls to overridden methods at run time.

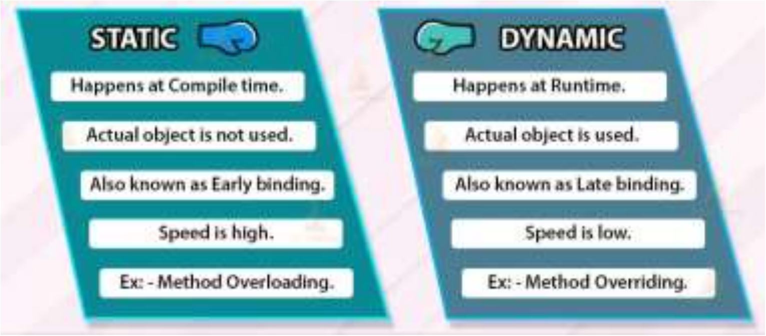

a
b
c
copy of A's m1() method in B's object
m1()
m1()
m1()
copy of A's m1() method in C's object
m1()
m1()
A class object
B class object
C class object

Advantages of Dynamic Method Dispatch

1. Dynamic method dispatch allow Java to support overriding of methods which is central for run-time polymorphism.
2. It allows a class to specify methods that will be common to all of its derivatives, while allowing subclasses to define the specific implementation of some or all of those methods.
3. It also allow subclasses to add its specific methods subclasses to define the specific implementation of some.

Static vs Dynamic binding

- Static binding is done during compile-time while dynamic binding is done during run-time.
- private, final and static methods and variables uses static binding and bonded by compiler while overridden methods are bonded during runtime based upon type of runtime object



STATIC	DYNAMIC
Happens at Compile time.	Happens at Runtime.
Actual object is not used.	Actual object is used.
Also known as Early binding.	Also known as Late binding.
Speed is high.	Speed is low.
Ex: - Method Overloading.	Ex: - Method Overriding.

Abstract class in Java

Sr. No.	Key	Abstract Class	Concrete Class
1	Supported Methods	Abstract class can have both an abstract as well as concrete methods.	A concrete class can only have concrete methods. Even a single abstract method makes the class abstract.
2	Instantiation	Abstract class can not be instantiated using new keyword.	Concrete class can be instantiated using new keyword.
3	Abstract method	Abstract class may or may not have abstract methods.	Concrete clas can not have an abstract method.
4	Final	Abstract class can not be declared as a final class.	Concrete class can be declared final.
5	Keyword	Abstract class declared using abstract keyword.	Concrete class is not having abstract keyword during declaration.
6	Inheritance	Abstract class can inherit another class using extends keyword and implement an **interface.**	Interface can inherit only an inteface.
7	Interface	Abstract class can not implement an interface alone. A child class is needed to be able to use the interface for instantiation.	Interface can be implemented easily.

Abstract class in Java

A class that is declared with abstract keyword, is known as **abstract class**. An abstract class is one which is containing some defined method and some undefined method. In java programming undefined methods are known as un-Implemented, or abstract method.

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)

Syntax to declare the abstract class

```
abstract class <class_name>
{
}
```

abstract method

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

Syntax to define the abstract method

```
abstract return_type <method_name>();     //no braces{}
```

Example of abstract class that have abstract method

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

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

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

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

Output:running safely..

The finalize( ) Method:

- It is possible to define a method that will be called just before an object's final destruction by the garbage collector.
- This method is called finalize( ), and it can be used to ensure that an object terminates cleanly.
- Clean-up activity means closing the resources associated with that object like Database Connection, Network Connection or we can say resource de-allocation.
- For example, you might use finalize( ) to make sure that an open file owned by that object is closed.
- Inside the finalize( ) method you will specify those actions that must be performed before an object is destroyed.

```
class Bye
{
  // Here overriding finalize method
            public void finalize()
            {
                        System.out.println("finalize method overriden");
            }

            void run()
            {
                        System.out.println("hellow");
            }

   public static void main(String[] args)
   {
      Bye m = new Bye();
      m.run();
      // Calling finalize method Explicitly.
     // m.finalize();
      //m.finalize();
      m = null;
      // Requesting JVM to call Garbage Collector method
      System.gc();
      System.out.println("Main Completes");
   }
}
```

Interface

1.What is an Interface? Give example. 2m

Give the general form of interface with an example. 4m 2016

How to declare an interface? 2m

An interface is a blueprint of a class. The interface is a mechanism to achieve abstraction in java. There can be only abstract methods in the interface. It is used to achieve **fully abstraction** and **multiple inheritace** in java.

```
interface <interface_name>
{

    // declare constant fields
    // declare abstract methods
}
```

Example:

```
interface Shape
{
    final static double pi=3.142;
   public void area(float r);
     //abstract method
}
```

Why to use interface??

- It is used to achieve fully abstraction.
- By interface, we can support the functionality of multiple inheritance.
- “A class cannot extend two classes but it can implement two interfaces”.

Interface is a 100 % pure **Abstract class** means it is a class which is 100 % abstract in nature i.e 100 % unknown what it does.

//multiple inheritance by interface

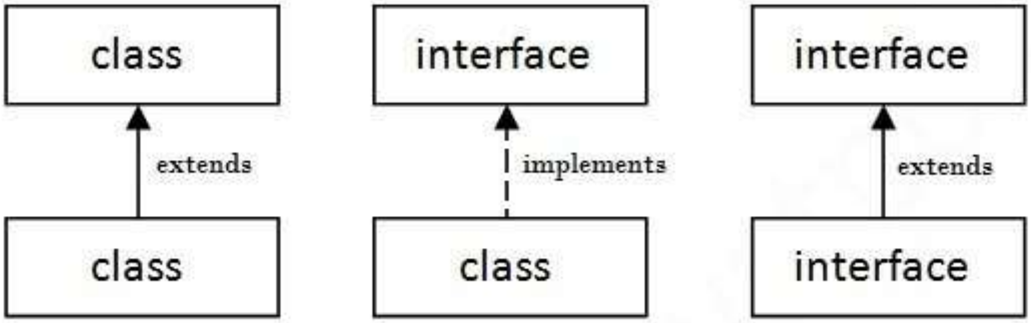
Explain with an example how a class implements an interface. 5m

```
 interface Father
{
  float ht=6.2f;
  void height();
}

interface Mother
{
  float ht=5.8f;
  void height();
}

class Child implements Father, Mother
{
  public void height()
  {
     float ht= (Father.ht + Mother.ht)/2;
```

```
    System.out.println("child's height is" + ht);
  }
}

class Multi
{
  public static void main(String [] arg)
  {
    Child ch = new Child();
    ch.height();
  }
}
```

output:

```
c:\>javac Multi.java
c:\>java Multi
Child's height is 6.0
```



Multiple inheritance is not supported in case of class but it is supported in case of interface, Why? 2m

because in class there is implementation provided in methods.
Whereas Interface has no implementation and it provides 100% abstraction.

//WAP to calculate the area of circle using interface.

```
interface Shape
{
   final static double pi=3.142;   //final static is optional

  public void area(float r);
}

class Circle implements Shape
```

```
{
  float rad;
  float area_circle;

  public void area(float r)
  {
    rad = r;
    area_circle = (float)pi * rad * rad;
    System.out.println("Area is " +area_circle);
  }

public static void main(String args[])
{
 Circle c = new Circle();
 c.area(5.5f);
 }
}
```

Write 5 differences between Abstract class and Interface. 5m

	Abstract class	Interface
1	It is collection of abstract method and concrete methods.	It is collection of abstract method.
2	It does not support multiple inheritance.	It support multiple inheritance.
3	Abstract class is preceded by abstract keyword.	It is preceded by Interface keyword.
4	Which may contain either variable or constants.	Which should contains only constants.
5	These class properties can be reused in other class using **extend** keyword.	These properties can be reused in any other class using **implements** keyword.
6	It can contain constructor.	It can not contain constructor

7	For the abstract class there is no restriction like initialization of variable at the time of variable declaration.	For the interface it should be compulsory to initialization of variable at the time of variable declaration.

What is the major difference between an interface and a class?

INTERFACE	CLASS
Interfaces cannot contain **implementation**.	Classes can contain implementation.
Interface cannot be **instantiated** (Objects cannot be created).	Classes can be instantiated (Objects can be created).
An interface is a **blueprint** of a class.	A *class* is the blueprint from which individual objects are created.
It can not contain constructor.	It can contain constructor.
All methods in an interface are abstract.	All methods in a class are not abstract.

What is a package? 2m

All classes in java belong to some package.

Package is a mechanism for organizing a group of related files in the same directory. It has the ability to reuse the classes and interfaces declared / defined in the package, again and again.

OR

“A package is a group of similar type of classes, interfaces and sub-packages”.

What are the Advantage of package? 5m

1. Application development time is less, because reuse the code
2. Application memory space is less (main memory)
3. Application execution time is less
4. Application performance is enhance (improve)
5. Redundancy (repetition) of code is minimized
6. Package provides access protection.
7. Package removes naming collision.

What are the Types of package? Explain . 5m

- **Predefined or built-in package**
- **User defined package**

Predefined or built-in package

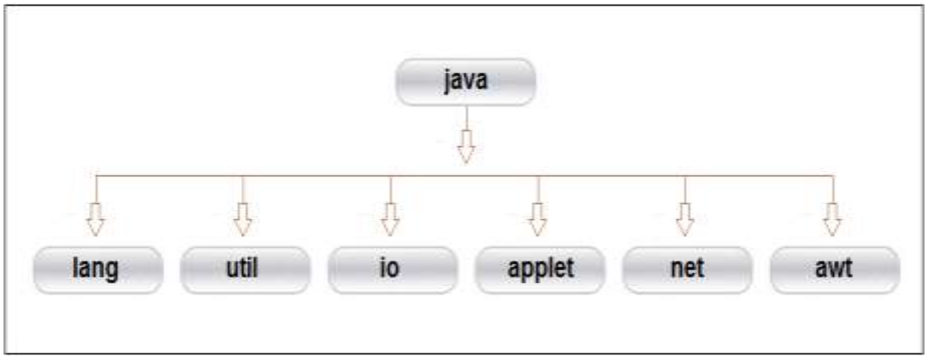
These are the package which are already designed by the Sun Microsystem and supply as a part of java API, every predefined package is collection of predefined classes, interfaces and sub-package.

There are many built-in packages such as **java, lang, awt, javax, swing, net, io, util, sql etc.**
example:
import java.io.*; - classes for input , output functions are bundled in this package
import java.lang ; - bundles the fundamental classes

User defined package

If any package is design by the user is known as user defined package. User defined package are those which are developed by java programmer and supply as a part of their project to deal with common requirement.

What are the frequently used API packages? 2m
Explain Java API packages. 5m



Java System Packages and Their Classes

java.lang	Language support classes. They include classes for primitive types, string, math functions,thread and exceptions.
java.util	Language utility classes such as vectors, hash tables, random numbers, etc.
java.io	Input/output support classes. They provide facilities for the input and output of data.
java.applet	Classes for creating and implementing applets.
java.net	Classes for networking. They include classes for communicating with local computers as well as with internet servers.
java.awt	Set of classes for implementing graphical user interface. They include classes for windows, buttons, lists, menus and so on.

Abstract window toolkit - awt

What are the naming conventions of packages? 2m

It is common practice to use **lowercased** names of packages to avoid any conflicts with the names of classes, interfaces.

Ex: **java.lang.Math.*;**

Creating a package:-

WRITE JAVA PROGRAM TO CALCULATE AREA OF SQUARE USING PACKAGE

Type the following code in notepad.

```
package figure;  //package declaration
 public class Square
{
   float side;
   float area;

  public Square (float a)   //constructor
  {
    side=a;
  }

  public void calarea()
  {
     area=side*side;
    System.out.println("Area of square " +area);
  }
}
```

Step1: CREATE A FOLDER IN	**D:\BCA1\figure\**
Step2: save Square.java in	**D:\BCA1\figure\Square.java**
Step3: Compile Square.java	**D:\BCA1\figure\javac Square.java**

Type the following code in notepad.

```
import figure.*;
 public class A
{
   public static void main(String str[])
  {
      Square S1 = new Square(20);
      S1.calarea();
   }
```

Step1: //Create file name called A.java and save in	D:\BCA\A.java
Step 2: Goto command prompt and compile	D:\BCA\javac A.java
Step3: To execute	D:\BCA\java A

How to create package in Java 5m

1. First create a directory within name of package.
2. Create a java file in newly created directory.
3. In this java file you must specify the package name with the help of package keyword.
4. Save this file with same name of public class. ...
5. Now you can use this package in your program.

Example: give above example.

Java.util Package in Java

Java.util Package

It contains the collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes (a string tokenizer, a random-number generator, and a bit array).

Following are the Important Classes in Java.util package :

1. AbstractCollection: This class provides a skeletal implementation of the Collection interface, to minimize the effort required to implement this interface.
2. AbstractList: This class provides a skeletal implementation of the List interface to minimize the effort required to implement this interface backed by a “random access” data store (such as an array).
3. AbstractQueue: This class provides skeletal implementations of some Queue operations.
4. AbstractSequentialList: This class provides a skeletal implementation of the List interface to minimize the effort required to implement this interface backed by a “sequential access” data store (such as a linked list).
5. ArrayDeque: Resizable-array implementation of the Deque interface.
6. ArrayList: Resizable-array implementation of the List interface.
7. Arrays: This class contains various methods for manipulating arrays (such as sorting and searching).
8. **Calendar:** The Calendar class is an abstract class that provides methods for converting between a specific instant in time and a set of calendar fields such as YEAR, MONTH, DAY_OF_MONTH, HOUR, and so on, and for manipulating the calendar fields, such as getting the date of the next week.
9. **Collections**: This class consists exclusively of static methods that operate on or return collections.
10. **Currency:** Represents a currency.
11. Date: The class Date represents a specific instant in time, with millisecond precision.
12. Random: An instance of this class is used to generate a stream of pseudorandom numbers.
13. Scanner: A simple text scanner which can parse primitive types and strings using regular expressions.
14. Stack: The Stack class represents a last-in-first-out (LIFO) stack of objects.
15. Timer: A facility for threads to schedule tasks for future execution in a background thread.
16. Vector: The Vector class implements a growable array of objects.

Generic Programming 2m

Java Generics is **a set of related methods or a set of similar types**. Generics allow types Integer, String, or even user-defined types to be passed as a parameter to classes, methods, or interfaces. Generics are mostly used by classes like HashSet or HashMap.

The Java Generics **allows us to create a single class, interface, and method that can be used with different types of data (objects)**. This helps us to reuse our code. Note: Generics does not work with primitive types (int , float , char , etc).

OR

It would be nice if we could write a single sort method that could sort the elements in an Integer array, a String array, or an array of any type that supports ordering.

Java **Generic** methods and generic classes enable programmers to specify, with a single method declaration, a set of related methods, or with a single class declaration, a set of related types, respectively.

Generics also provide compile-time type safety that allows programmers to catch invalid types at compile time.

What is instanceof Operator in java?

The instanceof operator in Java is used to check whether an object is an instance of a particular class or not.

Its syntax is

```
objectName instanceOf className;
```

Here, if objectName is an instance of className, the operator returns true. Otherwise, it returns false.

```
class Main {

  public static void main(String[] args) {

    // create a variable of string type
    String name = "WELCOME";

    // checks if name is instance of String
    boolean result1 = name instanceof String;
    System.out.println("name is an instance of String: " + result1);

    // create an object of Main
    Main obj = new Main();

    // checks if obj is an instance of Main
    boolean result2 = obj instanceof Main;
    System.out.println("obj is an instance of Main: " + result2);
  }
}
```

Casting objects 2m

Type Casting is **a feature in Java using which the form or type of a variable or object is cast into some other kind or Object**, and the process of conversion from one type to another is called Type Casting.

java object typecasting **one object reference can be type cast into another object reference**. The cast can be to its own class type or to one of its subclass or superclass types or interfaces. There are compile-time rules and runtime rules for casting in java.

End1