Core Java 170 Question JavaTpoint

1)What is difference between JDK,JRE and JVM?

Difference between JDK,JRE and JVM

Understanding the difference between JDK, JRE and JVM is important in Java. We will have brief overview of JVM here. If you want to gain the detailed knowledge of JVM, move to the next page. Firstly, let's see the basic differences between the JDK, JRE and JVM.

JVM

JVM (Java Virtual Machine) is an abstract machine.It is a specification that provides runtime environment in which java bytecode can be executed.

JVMs are available for many hardware and software platforms (i.e.JVM is plateform dependent).

The JVM performs four main tasks:

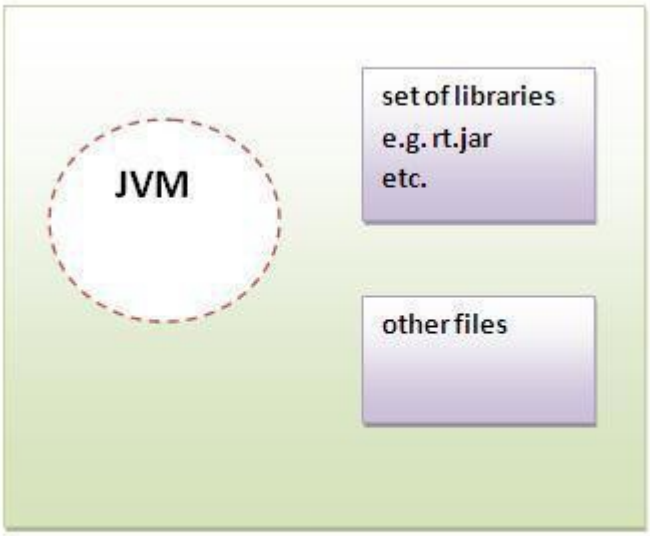
- Loads code
- Verifies code
- Executes code
- Provides runtime environment

JRE

JRE is an acronym for Java Runtime Environment.It is used to provide runtime environment.It is the implementation of JVM.It physically exists.It contains set of libraries + other files that JVM uses at runtime.

Implementation of JVMs are also actively released by other companies besides Sun

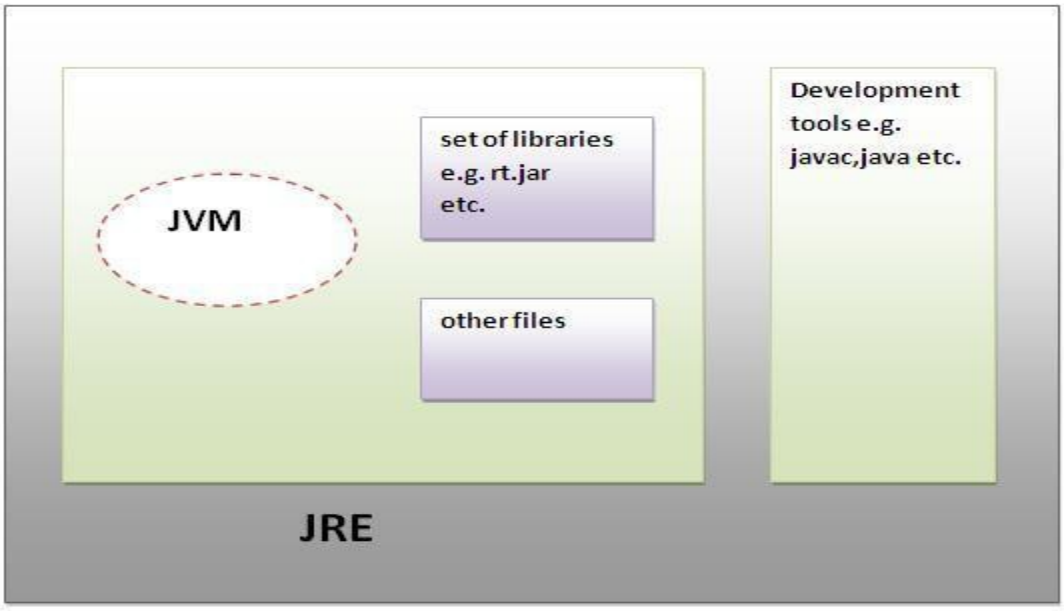
Micro Systems.



JRE

JDK

JDK is an acronym for Java Development Kit.It physically exists.It contains JRE + development tools.



JDK

2)How many types of memory areas are allocated by JVM?

JVM (Java Virtual Machine)

JVM (Java Virtual Machine) is an abstract machine.It is a specification that provides runtime environment in which java bytecode can be executed.

JVMs are available for many hardware and software platforms (i.e.JVM is plateform dependent).

The JVM performs four main tasks:

- Loads code
- Verifies code
- Executes code
- Provides runtime environment

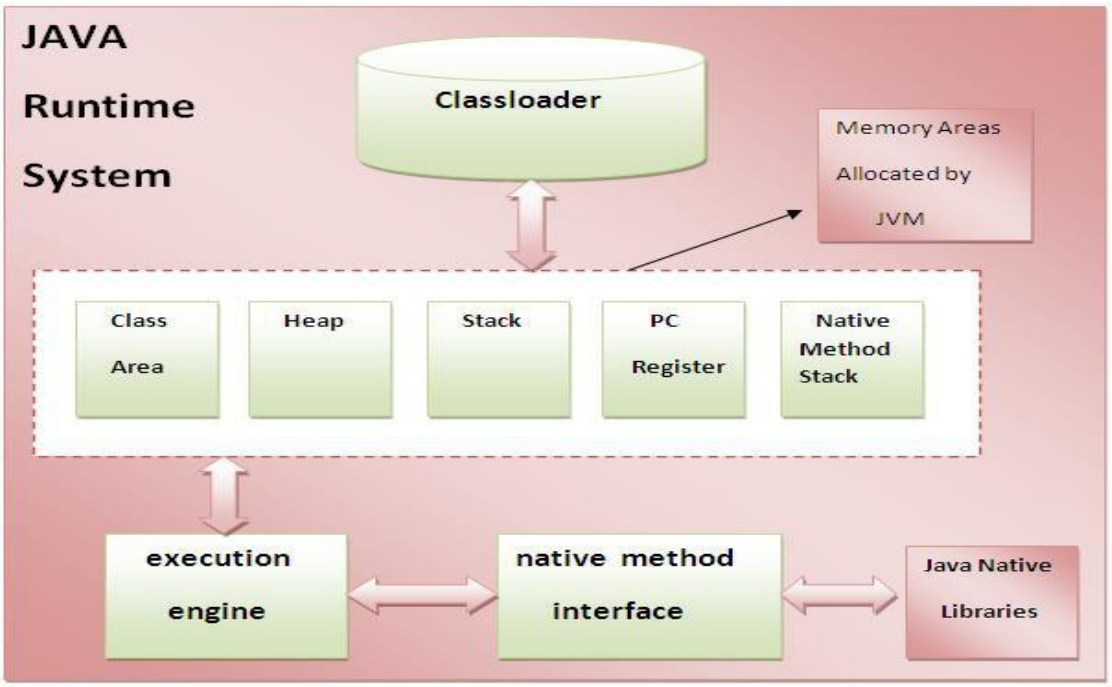
JVM provides definitions for the:

- Memory area

- Class file format
- Register set
- Garbage-collected heap
- Fatal error reporting etc.

Internal Architecture of JVM

Let's understand the internal architecture of JVM. It contains classloader, memory area, execution engine etc.



1) Classloader:

Classloader is a subsystem of JVM that is used to load class files.

2) Class(Method) Area:

Class(Method) Area stores per-class structures such as the runtime constant pool, field and method data, the code for methods.

3) Heap:

It is the runtime data area in which objects are allocated.

4) Stack:

Java Stack stores frames.It holds local variables and partial results, and plays a part in method invocation and return.

Each thread has a private JVM stack, created at the same time as thread.

A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.

5) Program Counter Regiser:

PC (program counter) register. It contains the address of the Java virtual machine instruction currently being executed.

6) Native Method Stack:

It contains all the native methods used in the application.

7) Execution Engine:

It contains:

1) A virtual processor

2) Interpreter: Read bytecode stream then execute the instructions.

3) Just-In-Time(JIT) compiler: It is used to improve the performance.JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation.Here the term ?compiler? refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.

3)What is JIT compiler?

Just-In-Time(JIT) compiler: It is used to improve the performance. JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation.Here the term “compiler” refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.

4)What is platform?

A platform is basically the hardware or software environment in which a program runs. There are two types of platforms software-based and hardware-based. Java provides software-based platform.

5)What is the main difference between Java platform and other platforms?

The Java platform differs from most other platforms in the sense that it's a software-based platform that runs on top of other hardware-based platforms.It has two components:

1. Runtime Environment
2. API(Application Programming Interface)

6)What gives Java its 'write once and run anywhere' nature?

The bytecode. Java is compiled to be a byte code which is the intermediate language between source code and machine code. This byte code is not platorm specific and hence can be fed to any platform.

7)What is classloader?

The classloader is a subsystem of JVM that is used to load classes and interfaces.There are many types of classloaders e.g. Bootstrap classloader, Extension classloader, System classloader, Plugin classloader etc.

8)Is Empty .java file name a valid source file name?

Yes, save your java file by .java only, compile it by **javac .java** and run by **java yourclassname** Let's take a simple example:

```
1. //save by .java only
2.
3. class A{
4. public static void main(String args[]){
5. System.out.println("Hello java");
6. }
7. }
8.
9. //compile by javac .java
10. //run by     java A
```

compile it by **javac .java**

run it by **java A**

9)Is delete,next,main,exit or null keyword in java?

No.

10)If I do not provide any arguments on the command line, then the String array of Main method will be empty or null?

It is empty. But not null.

11)What if I write static public void instead of public static void?

Program compiles and runs properly.

12)What is the default value of the local variables?

Ans:yes,that is current class instance (You cannot use return type yet it returns a value).

13)What is difference between object oriented programming language and object based programming language?

Object based programming languages follow all the features of OOPs except Inheritance.

Examples of object based programming languages are JavaScript, VBScript etc.

14)What will be the initial value of an object reference which is defined as an instance variable?

The object references are all initialized to null in Java.

15)What is constructor?

Constructor in Java

Constructor is a **special type of method** that is used to initialize the object.

Constructor is **invoked at the time of object creation**. It constructs the values i.e. provides data for the object that is why it is known as constructor.

1. Types of constructors
1. Default Constructor
2. Parameterized Constructor
2. Constructor Overloading
3. Does constructor return any value
4. Copying the values of one object into another initialization

Rules for creating constructor

There are basically two rules defined for the constructor.

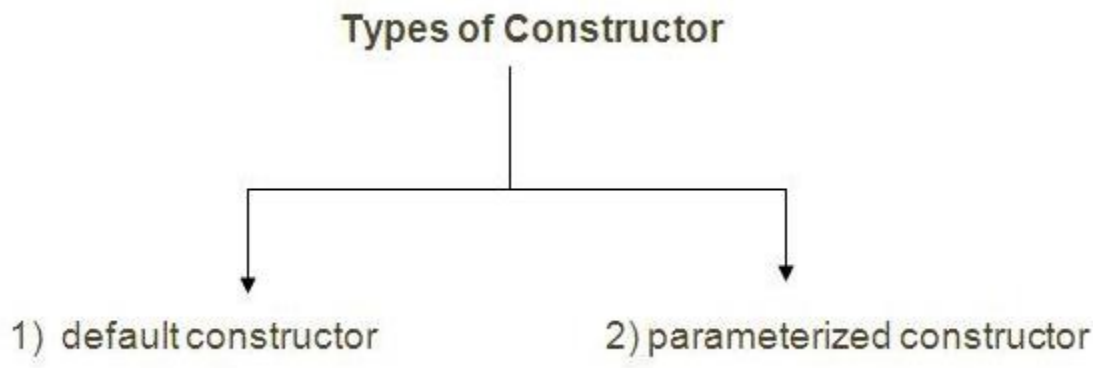
1.Constructor name must be same as its class name

2.Constructor must have no explicit return type

Types of constructors

There are two types of constructors:

1. default constructor (no-arg constructor)
2. parameterized constructor



---

1) Default Constructor

A constructor that have no parameter is known as default constructor.

Syntax of default constructor:

```
1. <class_name>(){}
```

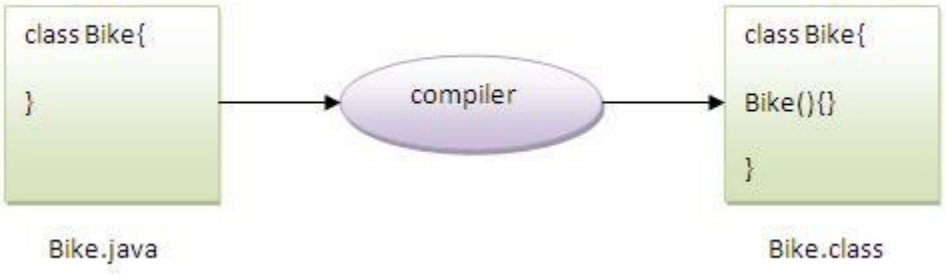
Example of default constructor

In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

```
1. class Bike{
2.
3. Bike(){System.out.println("Bike is created");}
4.
5. public static void main(String args[]){
6. Bike b=new Bike();
7. }
8. }
```

Output: Bike is created

Rule: If there is no constructor in a class, compiler automatically creates a default constructor.



Que)What is the purpose of default constructor?

Default constructor provides the default values to the object like 0, null etc. depending on the type.

Example of default constructor that displays the default values

```
1. class Student{
2. int id;
3. String name;
4.
5. void display(){System.out.println(id+" "+name);}
6.
7. public static void main(String args[]){
8. Student s1=new Student();
9. Student s2=new Student();
10.         s1.display();
11.         s2.display();
12.         }
13.         }
```

```
1. <strong>Output:</strong>0 null
2.       0 null
```

Explanation: In the above class,you are not creating any constructor so compiler provides you a default constructor.Here 0 and null values are provided by default constructor.

Parameterized constructor

A constructor that have parameters is known as parameterized constructor.

Why use parameterized constructor?

Parameterized constructor is used to provide different values to the distinct objects.

Example of parameterized constructor

In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

```
1. class Student{
2.    int id;
3.    String name;
4.
5.    Student(int i,String n){
6.    id = i;
7.    name = n;
8.    }
9.    void display(){System.out.println(id+" "+name);}
10.
11.         public static void main(String args[]){
12.         Student s1 = new Student(111,"Karan");
13.         Student s2 = new Student(222,"Aryan");
14.         s1.display();
15.         s2.display();
16.         }
17.       }
```

Output:111 Karan
222 Aryan

Constructor Overloading

Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists.The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.

Example of Constructor Overloading

```
1. class Student{
2.   int id;
3.   String name;
4.   int age;
5.   Student(int i,String n){
6.   id = i;
7.   name = n;
8.   }
9.   Student(int i,String n,int a){
10.        id = i;
11.        name = n;
12.        age=a;
13.        }
14.        void display(){System.out.println(id+" "+name+" "+age);}
15.
16.        public static void main(String args[]){
17.        Student s1 = new Student(111,"Karan");
18.        Student s2 = new Student(222,"Aryan",25);
19.        s1.display();
20.        s2.display();
21.        }
22.      }
```

Output:111 Karan 0
222 Aryan 25

What is the difference between constructor and method ?

There are many differences between constructors and methods. They are given

below.

Constructor	Method
Constructor is used to initialize the state of an object.	Method is used to expose behaviour of an object.
Constructor must not have return type.	Method must have return type.
Constructor is invoked implicitly.	Method is invoked explicitly.
The java compiler provides a default constructor if you don't have any constructor.	Method is not provided by compiler in any case.
Constructor name must be same as the class name.	Method name may or may not be same as class name.

Copying the values of one object to another like copy constructor in C++

There are many ways to copy the values of one object into another. They are:

- By constructor
- By assigning the values of one object into another
- By clone() method of Object class

In this example, we are going to copy the values of one object into another using constructor.

```
1. class Student{
2.    int id;
3.    String name;
4.    Student(int i,String n){
5.    id = i;
6.    name = n;
7.    }
8.
9.    Student(Student s){
10.           id = s.id;
```

```
11.         name =s.name;
12.         }
13.         void display(){System.out.println(id+" "+name);}
14.
15.         public static void main(String args[]){
16.         Student s1 = new Student(111,"Karan");
17.         Student s2 = new Student(s1);
18.         s1.display();
19.         s2.display();
20.         }
21.       }
```

Output:111 Karan
111 Karan

Copying the values of one object to another without constructor

We can copy the values of one object into another by assigning the objects values to another object. In this case, there is no need to create the constructor.

```
1. class Student{
2.   int id;
3.   String name;
4.   Student(int i,String n){
5.   id = i;
6.   name = n;
7.   }
8.   void display(){System.out.println(id+" "+name);}
9.
10.         public static void main(String args[]){
11.         Student s1 = new Student(111,"Karan");
12.         Student s2 = new Student();
13.         s2.id=s1.id;
14.         s2.name=s1.name;
15.         s1.display();
```

```
16.         s2.display();
17.         }
18.     }
```

Output:111 Karan
111 Karan

Que)Does constructor return any value?

Ans:yes,that is current class instance (You cannot use return type yet it returns a value).

Can constructor perform other tasks instead of initialization?

Yes, like object creation, starting a thread, calling method etc. You can perform any operation in the constructor as you perform in the method.

16)What is the purpose of default constructor?

Explained Above.

17)Does constructor return any value?

Explained Above.

18)Is constructor inherited?

No, constructor is not inherited.

19)Can you make a constructor final?

No, constructor can't be final.

20)What is static variable?

static keyword

The **static keyword** is used in java mainly for memory management. We may apply static keyword with variables, methods, blocks and nested class. The static keyword belongs to the class than instance of the class.

The static can be:

1.variable (also known as class

1. Static variable
2. Program of counter without static variable
3. Program of counter with static variable
4. Static method
5. Restrictions for static method
6. Why main method is static ?
7. Static block

method ?

variable)

2.method (also known as class method)

3.block

4.nested class

---

1) static variable

If you declare any variable as static, it is known static variable.

- The static variable can be used to refer the common property of all objects (that is not unique for each object) e.g. company name of employees,college name of students etc.
- The static variable gets memory only once in class area at the time of class loading.

Advantage of static variable

It makes your program memory efficient (i.e it saves memory).

Understanding problem without static variable

```
1. class Student{
2.    int rollno;
3.    String name;
4.    String college="ITS";
5. }
```

Suppose there are 500 students in my college, now all instance data members will get memory each time when object is created.All student have its unique rollno and name so instance data member is good.Here, college refers to the common property of all objects.If we make it static,this field will get memory only once.

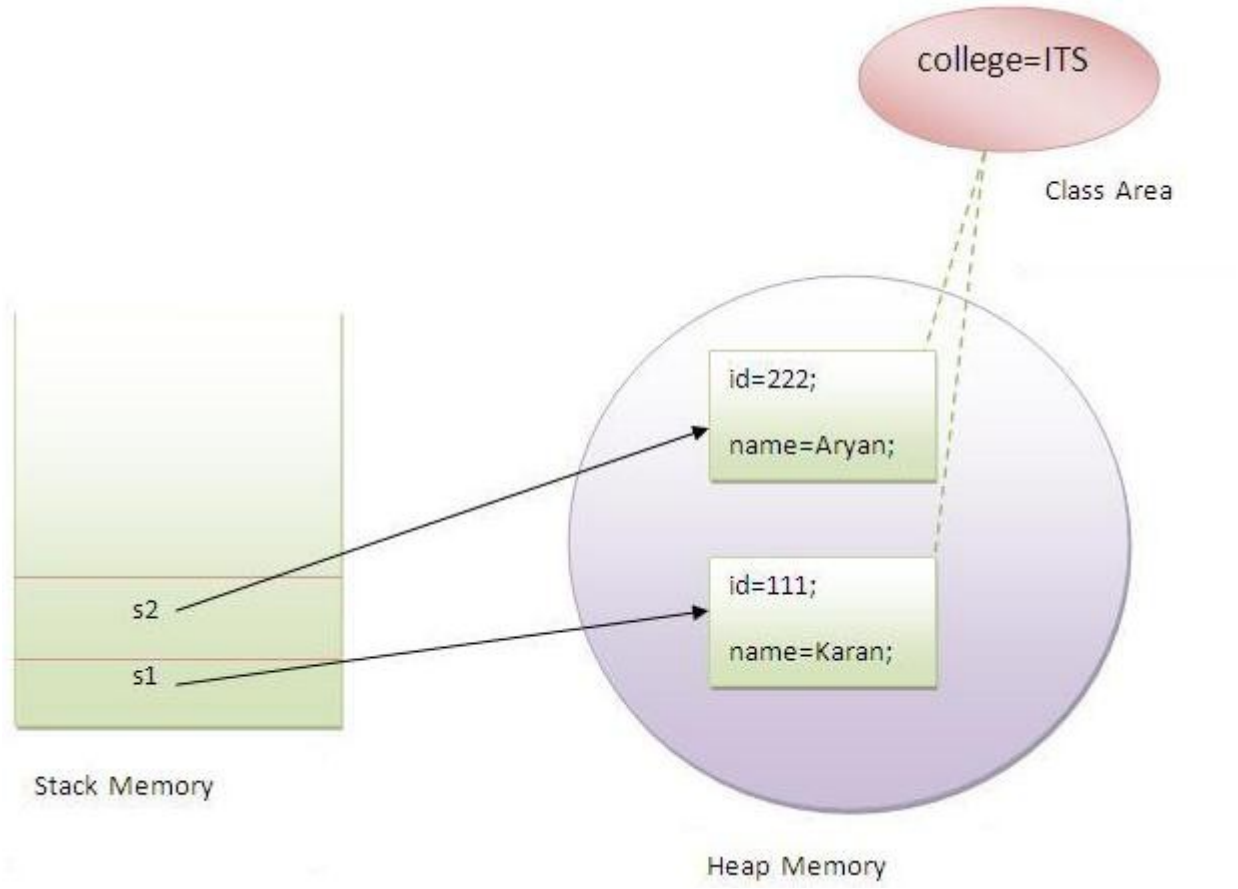
> *static property is shared to all objects.*

Example of static variable

```
1. //<b>Program of static variable</b>
2.
```

```
3. class Student{
4.   int rollno;
5.   String name;
6.   static String college ="ITS";
7.
8.   Student(int r,String n){
9.   rollno = r;
10.          name = n;
11.          }
12.          void display (){System.out.println(rollno+" "+name+" "+college);}
13.
14.          public static void main(String args[]){
15.          Student s1 = new Student (111,"Karan");
16.          Student s2 = new Student (222,"Aryan");
17.
18.          s1.display();
19.          s2.display();
20.          }
21.        }
```

Output:111 Karan ITS
222 Aryan ITS



Program of counter without static variable

In this example, we have created an instance variable named count which is incremented in the constructor. Since instance variable gets the memory at the time of object creation, each object will have the copy of the instance variable, if it is incremented, it won't reflect to other objects. So each objects will have the value 1 in the count variable.

```
1. class Counter{
2. int count=0;//will get memory when instance is created
3.
4. Counter(){
5. count++;
6. System.out.println(count);
7. }
8.
9. public static void main(String args[]){
```

```
10.
11.         Counter c1=new Counter();
12.         Counter c2=new Counter();
13.         Counter c3=new Counter();
14.
15.         }}
```

```
Output:1
       1
       1
```

Program of counter by static variable

As we have mentioned above, static variable will get the memory only once, if any object changes the value of the static variable, it will retain its value.

```
1. class Counter{
2. static int count=0;//will get memory only once and retain its value
3.
4. Counter(){
5. count++;
6. System.out.println(count);
7. }
8.
9. public static void main(String args[]){
10.
11.         Counter c1=new Counter();
12.         Counter c2=new Counter();
13.         Counter c3=new Counter();
14.
15.         }}
```

```
Output:1
       2
       3
```

21)What is static method?

If you apply static keyword with any method, it is known as static method.

- A static method belongs to the class rather than object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- static method can access static data member and can change the value of it.

Example of static method

```
1. //Program of changing the common property of all objects(static field).
2.
3. class Student{
4.     int rollno;
5.     String name;
6.     static String college = "ITS";
7.
8.     static void change(){
9.     college = "BBDIT";
10.         }
11.
12.         Student(int r, String n){
13.         rollno = r;
14.         name = n;
15.         }
16.
17.         void display (){System.out.println(rollno+" "+name+" "+college);}
18.
19.         public static void main(String args[]){
20.         Student.change();
21.
22.         Student s1 = new Student (111,"Karan");
23.         Student s2 = new Student (222,"Aryan");
24.         Student s3 = new Student (333,"Sonoo");
```

```
25.
26.         s1.display();
27.         s2.display();
28.         s3.display();
29.         }
30.       }
```

Output:111 Karan BBDIT
222 Aryan BBDIT
333 Sonoo BBDIT

Another example of static method that performs normal calculation

```
1. //Program to get cube of a given number by static method
2.
3. Class Calculate{
4.
5.  static int cube(int x){
6.  return x*x*x;
7.  }
8.
9.  public static void main(String args[]){
10.         int result=Calculate.cube(5);
11.         System.out.println(result);
12.         }
13.       }
```

Output:125

Restrictions for static method

There are two main restrictions for the static method. They are:

1. The static method can not use non static data member or call non-static method directly.

2. this and super cannot be used in static context.

```
1. class A{
2.  int a=40;//non static
3.
4.  public static void main(String args[]){
5.  System.out.println(a);
6.  }
7. }
```

Output: Compile Time Error

Que)why main method is static?

Ans) because object is not required to call static method if it were non-static method, jvm creats object first then call main() method that will lead the problem of extra memory allocation.

22)why main method is static?
Explained Above.

23)What is static block?
Is used to initialize the static data member.

- It is excuted before main method at the time of classloading.

Example of static block

```
1. class A{
2.
3.  static{System.out.println("static block is invoked");}
4.
5.  public static void main(String args[]){
6.  System.out.println("Hello main");
7.  }
8. }
```

Output:static block is invoked
Hello main

Que)Can we execute a program without main() method?

Ans)Yes, one of the way is static block but in previous version of JDK not in JDK 1.7.

```
1. class A{
2.  static{
3.  System.out.println("static block is invoked");
4.  System.exit(0);
5.  }
6. }
```

Output:static block is invoked (if not JDK7)

24)Can we execute a program without main() method?
Explained Above.

25)What if the static modifier is removed from the signature of the main method?

Program compiles. But at runtime throws an error "NoSuchMethodError".

26)What is difference between static (class) method and instance method?

1)A method i.e. declared as static is known as static method.	A method i.e. not declared as static is known as instance method.
2)Object is not required to call static method.	Object is required to call instance methods.
3)Non-static (instance) members cannot be accessed in static context (static method, static block and static nested class) directly.	static and non-static variables both can accessed in instance methods.
4)For example: public static int cube(int n){ return n*n*n;}	For example: public void msg(){...}.

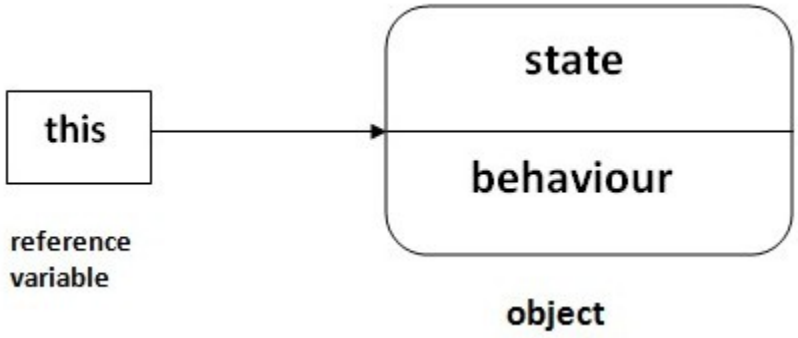
27)What is this in java?
There can be a lot of usage of **this keyword**. In java, this is a **reference variable** that refers to the current object.

Usage of this keyword

Here is given the 6 usage of this keyword.

1. this keyword can be used to refer current class instance variable.
2. this() can be used to invoke current class constructor.
3. this keyword can be used to invoke current class method (implicitly)
4. this can be passed as an argument in the method call.
5. this can be passed as argument in the constructor call.
6. this keyword can also be used to return the current class instance.

Suggestion:If you are beginner to java, lookup only two usage of this keyword.



---

1) The this keyword can be used to refer current class instance variable.

If there is ambiguity between the instance variable and parameter, this keyword resolves the problem of ambiguity.

Understanding the problem without this keyword

Let's understand the problem if we don't use this keyword by the example given below:

```
1. class student{
2.    int id;
3.    String name;
4.
5.    student(int id,String name){
6.    id = id;
7.    name = name;
8.    }
```

```
9.    void display(){System.out.println(id+" "+name);}
10.
11.        public static void main(String args[]){
12.        student s1 = new student(111,"Karan");
13.        student s2 = new student(321,"Aryan");
14.        s1.display();
15.        s2.display();
16.        }
17.      }
```

Output:0 null
0 null

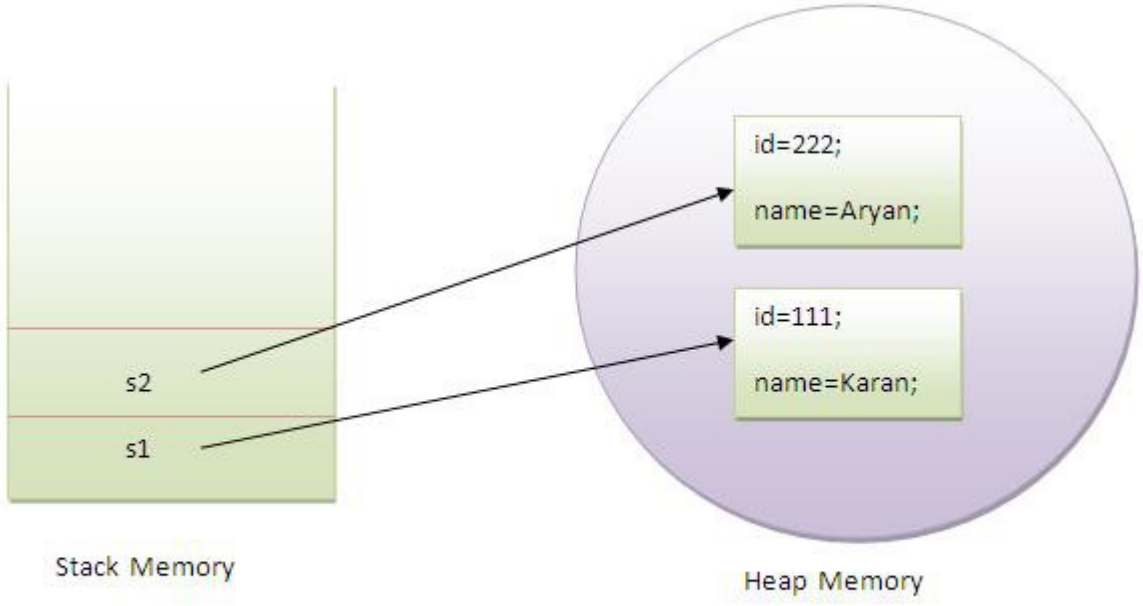
In the above example, parameter (formal arguments) and instance variables are same that is why we are using this keyword to distinguish between local variable and instance variable.

Solution of the above problem by this keyword

```
1. //example of this keyword
2.
3. class Student{
4.    int id;
5.    String name;
6.
7.    student(int id,String name){
8.    this.id = id;
9.    this.name = name;
10.        }
11.        void display(){System.out.println(id+" "+name);}
12.        public static void main(String args[]){
13.        Student s1 = new Student(111,"Karan");
14.        Student s2 = new Student(222,"Aryan");
15.        s1.display();
16.        s2.display();
```

```
17.     }
18.     }
```

Output:111 Karan
222 Aryan



If local variables(formal arguments) and instance variables are different, there is no need to use this keyword like in the following program:

Program where this keyword is not required

```
1. class Student{
2.   int id;
3.   String name;
4.
5.   student(int i,String n){
6.   id = i;
7.   name = n;
8.   }
9.   void display(){System.out.println(id+" "+name);}
10.      public static void main(String args[]){
```

```
        Student e1 = new Student(111,"karan");
        Student e2 = new Student(222,"Aryan");
        e1.display();
        e2.display();
    }
    }
```

Output:111 Karan
222 Aryan

---

2) this() can be used to invoked current class constructor.

The this() constructor call can be used to invoke the current class constructor (constructor chaining). This approach is better if you have many constructors in the class and want to reuse that constructor.

```
//Program of this() constructor call (constructor chaining)

class Student{
    int id;
    String name;
    Student (){System.out.println("default constructor is invoked");}

    Student(int id,String name){
    this ();//it is used to invoked current class constructor.
        this.id = id;
        this.name = name;
        }
        void display(){System.out.println(id+" "+name);}

        public static void main(String args[]){
        Student e1 = new Student(111,"karan");
        Student e2 = new Student(222,"Aryan");
        e1.display();
        e2.display();
```

```
20.         }
21.     }
```

Output:

```
default constructor is invoked
default constructor is invoked
111 Karan
222 Aryan
```

Where to use this() constructor call?

The this() constructor call should be used to reuse the constructor in the constructor. It maintains the chain between the constructors i.e. it is used for constructor chaining. Let's see the example given below that displays the actual use of this keyword.

```
1. class Student{
2.     int id;
3.     String name;
4.     String city;
5.
6.     Student(int id,String name){
7.     this.id = id;
8.     this.name = name;
9.     }
10.         Student(int id,String name,String city){
11.         this(id,name);//now no need to initialize id and name
12.         this.city=city;
13.         }
14.         void display(){System.out.println(id+" "+name+" "+city);}
15.
16.         public static void main(String args[]){
17.         Student e1 = new Student(111,"karan");
18.         Student e2 = new Student(222,"Aryan","delhi");
19.         e1.display();
20.         e2.display();
21.         }
```

```
22.     }
```

Output:111 Karan null
222 Aryan delhi

Rule: Call to this() must be the first statement in constructor.

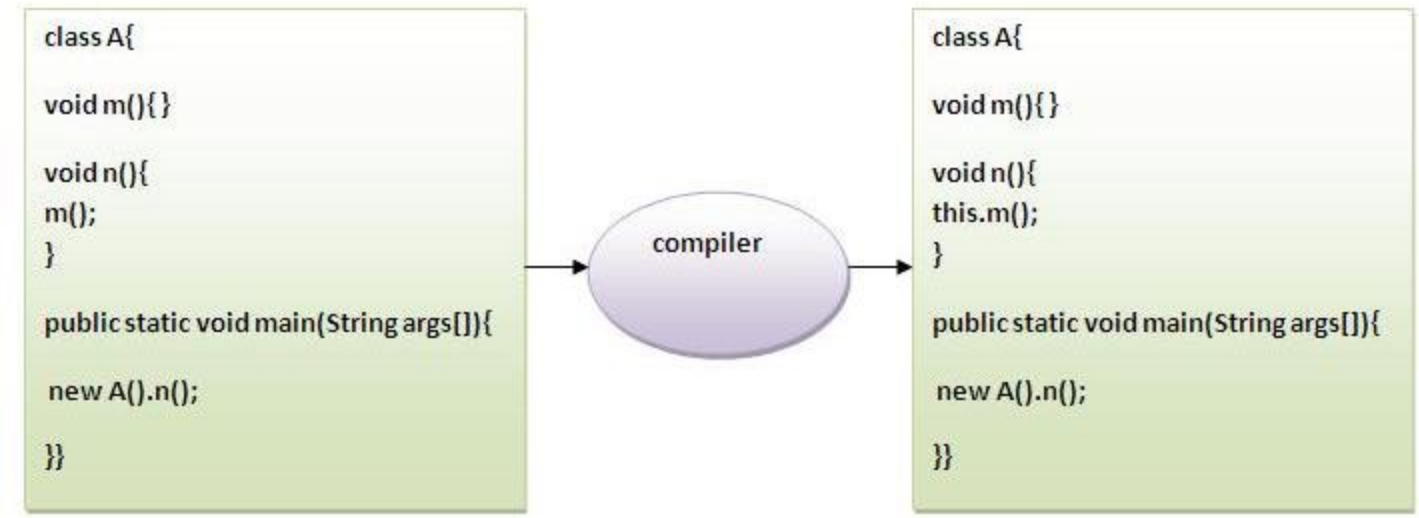
```
1. class Student{
2.   int id;
3.   String name;
4.   Student (){System.out.println("default constructor is invoked");}
5.
6.   Student(int id,String name){
7.   id = id;
8.   name = name;
9.   this ();//must be the first statement
10.       }
11.       void display(){System.out.println(id+" "+name);}
12.
13.       public static void main(String args[]){
14.       Student e1 = new Student(111,"karan");
15.       Student e2 = new Student(222,"Aryan");
16.       e1.display();
17.       e2.display();
18.       }
19.     }
```

Output: Compile Time Error

---

3)The this keyword can be used to invoke current class method (implicitly).

You may invoke the method of the current class by using the this keyword. If you don't use the this keyword, compiler automatically adds this keyword while invoking the method. Let's see the example




```
1. class S{
2.  void m(){
3.  System.out.println("method is invoked");
4.  }
5.  void n(){
6.  this.m();//no need because compiler does it for you.
7.  }
8.  void p(){
9.  n();//complier will add this to invoke n() method as this.n()
10.         }
11.         public static void main(String args[]){
12.         S s1 = new S();
13.         s1.p();
14.         }
15.       }
```

Output:method is invoked

4) this keyword can be passed as an argument in the method.

The this keyword can also be passed as an argument in the method. It is mainly used in the event handling. Let's see the example:

```
1. class S{
2.  void m(S obj){
3.  System.out.println("method is invoked");
4.  }
5.  void p(){
6.  m(this);
7.  }
8.
9.  public static void main(String args[]){
10.         S s1 = new S();
11.         s1.p();
12.         }
13.       }
```

Output:method is invoked

Application of this that can be passed as an argument:

In event handling (or) in a situation where we have to provide reference of a class to another one.

---

5) The this keyword can be passed as argument in the constructor call.

We can pass the this keyword in the constructor also. It is useful if we have to use one object in multiple classes. Let's see the example:

```
1. class B{
2.  A obj;
3.  B(A obj){
4.    this.obj=obj;
5.  }
6.  void display(){
7.    System.out.println(obj.data);//using data member of A class
8.  }
9. }
```

```
10.
11.     class A{
12.      int data=10;
13.      A(){
14.       B b=new B(this);
15.       b.display();
16.      }
17.      public static void main(String args[]){
18.       A a=new A();
19.      }
20.     }
```

Output:10

---

6) The this keyword can be used to return current class instance.

We can return the this keyword as an statement from the method. In such case, return type of the method must be the class type (non-primitive). Let's see the example:

Syntax of this that can be returned as a statement

```
1. return_type method_name(){
2. return this;
3. }
```

Example of this keyword that you return as a statement from the method

```
1. class A{
2. A getA(){
3. return this;
4. }
5. void msg(){System.out.println("Hello java");}
6. }
7.
8. class Test{
```

```
9.  public static void main(String args[]){
10.         new A().getA().msg();
11.         }
12.         }
```

Output: Hello java

Proving this keyword

Let's prove that this keyword refers to the current class instance variable. In this program, we are printing the reference variable and this, output of both variables are same.

```
1. class A{
2. void m(){
3. System.out.println(this);//prints same reference ID
4. }
5.
6. public static void main(String args[]){
7. A obj=new A();
8. System.out.println(obj);//prints the reference ID
9.
10.         obj.m();
11.         }
12.         }
```

Output:A@13d9c02
A@13d9c02

28)What is Inheritance?

Inheritance in Java

Inheritance is a mechanism in which one object acquires all the properties and behaviours of parent object.

1. Inheritance
2. Types of Inheritance

in case of class?

The idea behind inheritance is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you reuse (or inherit) methods and fields, and you add new methods and fields to adapt your new class to new

situations.

Inheritance represents the **IS-A relationship**.

Why use Inheritance?

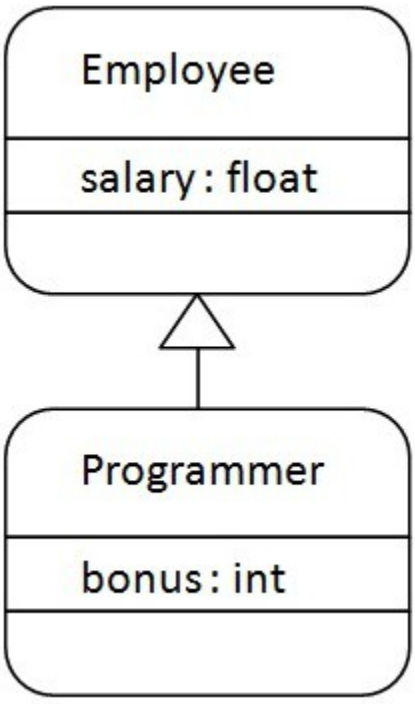
- For Method Overriding (So Runtime Polymorphism).
- For Code Reusability.

Syntax of Inheritance

```
1. class Subclass-name extends Superclass-name
2. {
3.   //methods and fields
4. }
```

The keyword extends indicates that you are making a new class that derives from an existing class. In the terminology of Java, a class that is inherited is called a superclass. The new class is called a subclass.

Understanding the simple example of inheritance



As displayed in the above figure, Programmer is the subclass and Employee is the superclass. Relationship between two classes is **Programmer IS-A Employee**.It means

that Programmer is a type of Employee.

```
1. class Employee{
2.  float salary=40000;
3. }
4.
5. class Programmer extends Employee{
6.  int bonus=10000;
7.
8.  public static void main(String args[]){
9.   Programmer p=new Programmer();
10.          System.out.println("Programmer salary is:"+p.salary);
11.          System.out.println("Bonus of Programmer is:"+p.bonus);
12.       }
13.       }
```

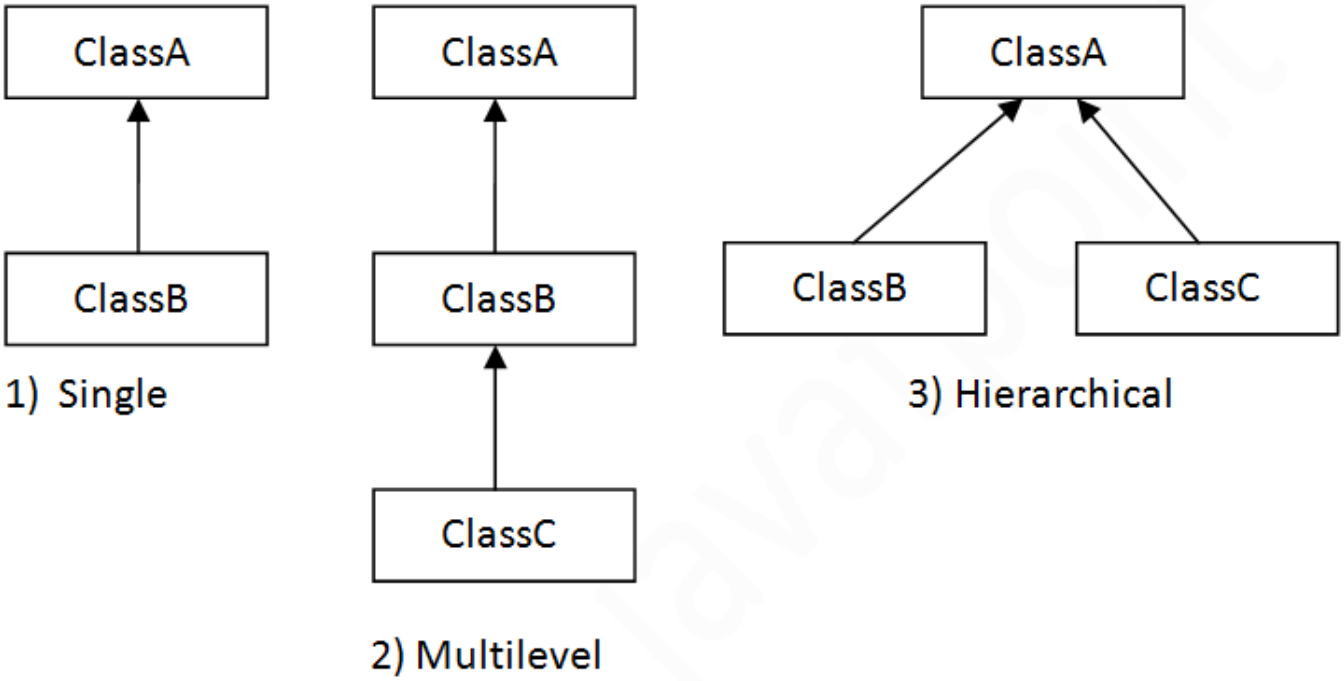
```
Output:Programmer salary is:40000.0
          Bonus of programmer is:10000
```

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

Types of Inheritance

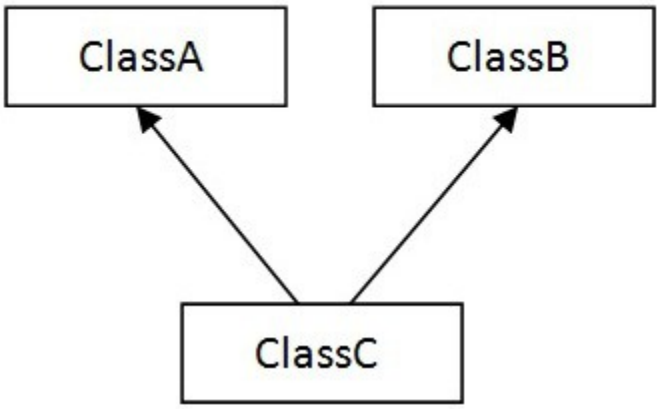
On the basis of class, there can be three types of inheritance: single, multilevel and hierarchical.

Multiple and Hybrid is supported through interface only. We will learn about interfaces later.

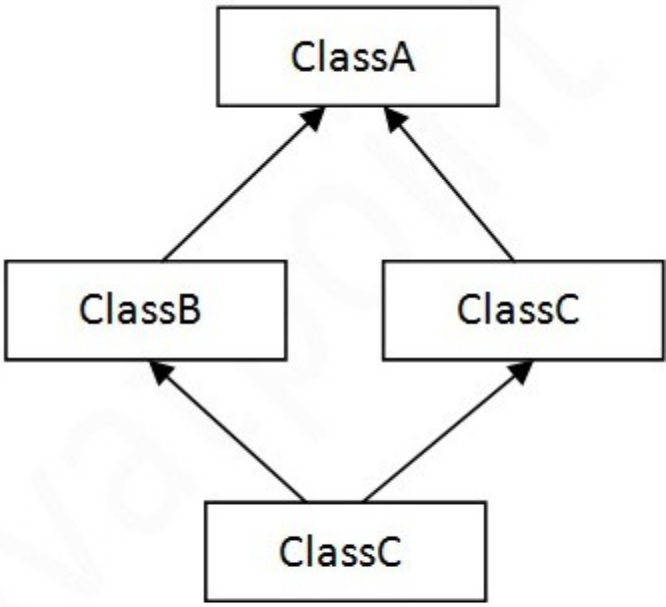


Multiple inheritance is not supported in java in case of class.

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



4) Multiple



5) Hybrid

.

Que) Why multiple inheritance is not supported in java?

- To reduce the complexity and simplify the language, multiple inheritance is not supported in java. For example:

```
1. class A{
2. void msg(){System.out.println("Hello");}
3. }
4.
5. class B{
6. void msg(){System.out.println("Welcome");}
7. }
8.
9. class C extends A,B{//suppose if it were
10.
11.        Public Static void main(String args[]){
12.         C obj=new C();
13.         obj.msg();//Now which msg() method would be invoked?
```

```
14.     }
15.     }
```

29)Which class is the superclass for every class.

Object class.

30) Why multiple inheritance is not supported in java?

Explained Above.

31)What is composition?

Holding the reference of the other class within some other class is known as composition.

32)What is difference between aggregation and composition?

Aggregation represents weak relationship whereas composition represents strong relationship. For example: bike has an indicator (aggregation) but bike has an engine (compostion).

33)Why Java does not support pointers?

Pointer is a variable that refers to the memory address. They are not used in java because they are unsafe(unsecured) and complex to understand.

34)What is super in java?

super keyword

The **super** is a reference variable that is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitely i.e. referred by super reference variable.

Usage of super Keyword

1.super is used to refer immediate parent class instance variable.

2.super() is used to invoke immediate parent class constructor.

3.super is used to invoke immediate parent class method.

1) super is used to refer immediate parent class instance variable.

Problem without super keyword

```
1. class Vehicle{
```

```
2.  int speed=50;
3. }
4.
5. class Bike extends Vehicle{
6.  int speed=100;
7.
8.  void display(){
9.   System.out.println(speed);//will print speed of Bike
10.         }
11.         public static void main(String args[]){
12.          Bike b=new Bike();
13.          b.display();
14.
15.        }
16.        }
```

```
Output:100
```

In the above example Vehicle and Bike both class have a common property speed. Instance variable of current class is refered by instance bydefault, but I have to refer parent class instance variable that is why we use super keyword to distinguish between parent class instance variable and current class instance variable.

Solution by super keyword

```
1. //example of super keyword
2.
3. class Vehicle{
4.  int speed=50;
5. }
6.
7. class Bike extends Vehicle{
8.  int speed=100;
9.
10.         void display(){
11.          System.out.println(super.speed);//will print speed of Vehicle now
```

```
12.         }
13.         public static void main(String args[]){
14.          Bike b=new Bike();
15.          b.display();
16.
17.        }
18.        }
```

```
Output:50
```

2) super is used to invoke parent class constructor.

The super keyword can also be used to invoke the parent class constructor as given below:

```
1. class Vehicle{
2.  Vehicle(){System.out.println("Vehicle is created");}
3. }
4.
5. class Bike extends Vehicle{
6.  Bike(){
7.   super();//will invoke parent class constructor
8.   System.out.println("Bike is created");
9.  }
10.         public static void main(String args[]){
11.          Bike b=new Bike();
12.
13.        }
14.        }
```

```
Output:Vehicle is created
          Bike is created
```

super() is added in each class construtor automatically by compiler.

As we know well that default constructor is provided by compiler automatically but it also adds super() for the first statement.If you are creating your own constructor and you

don't have either this() or super() as the first statement, compiler will provide super() as the first statement of the consructor.

Another example of super keyword where super() is provided by the compiler implicitely.

```
1. class Vehicle{
2.   Vehicle(){System.out.println("Vehicle is created");}
3. }
4.
5. class Bike extends Vehicle{
6.   int speed;
7.   Bike(int speed){
8.     this.speed=speed;
9.     System.out.println(speed);
10.        }
11.        public static void main(String args[]){
12.        Bike b=new Bike(10);
13.        }
14.       }
```

```
Output:Vehicle is created
          10
```

3) super can be used to invoke parent class method.

The super keyword can also be used to invoke parent class method. It should be used in case subclass contains the same method as parent class as in the example given below:

```
1. class Person{
2. void message(){System.out.println("welcome");}
3. }
4.
5. class Student extends Person{
6. void message(){System.out.println("welcome to java");}
7.
8. void dislay(){
```

```
9. message();//will invoke current class message() method
10.         super.message();//will invoke parent class message() method
11.         }
12.
13.         public static void main(String args[]){
14.         Student s=new Student();
15.         s.display();
16.         }
17.         }
```

```
Output:welcome to java
           welcome
```

In the above example Student and Person both classes have message() method if we call message() method from Student class, it will call the message() method of Student class not of Person class because priority is given to local.

In case there is no method in subclass as parent, there is no need to use super. In the example given below message() method is invoked from Student class but Student class does not have message() method, so you can directly call message() method.

Program in case super is not required

```
1. class Person{
2. void message(){System.out.println("welcome");}
3. }
4.
5. class Student extends Person{
6.
7. void dislay(){
8. message();//will invoke parent class message() method
9. }
10.
11.         public static void main(String args[]){
12.         Student s=new Student();
13.         s.display();
14.         }
```

```
15.     }
```

```
Output:welcome
```

35)Can you use this() and super() both in a constructor?

No. Because super() or this() must be the first statement.

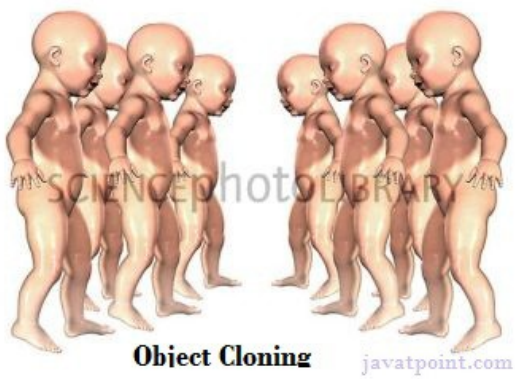
36)What is object cloning?

Object Cloning in Java

The **object cloning** is a way to create exact copy of an object. For this purpose, 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:




```
1. protected Object clone() throws CloneNotSupportedException
```

Why use clone() method ?

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 to be performed that is why we use object cloning.

Advantage of Object cloning

Less processing task.

Example of clone() method (Object cloning)

Let's see the simple example of object cloning

```
1. class Student implements Cloneable{
2. int rollno;
3. String name;
4.
5. Student(int rollno,String name){
```

```
6. this.rollno=rollno;
7. this.name=name;
8. }
9.
10.     public Object clone()throws CloneNotSupportedException{
11.     return super.clone();
12.     }
13.
14.     public static void main(String args[]){
15.     try{
16.     Student s1=new Student(101,"amit");
17.
18.     Student s2=(Student)s1.clone();
19.
20.     System.out.println(s1.rollno+" "+s1.name);
21.     System.out.println(s2.rollno+" "+s2.name);
22.
23.     }catch(CloneNotSupportedException c){}
24.
25.     }
26.     }
```

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

37)What is method overloading?

Method Overloading in Java

If a class have multiple methods by same name but different 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

1. Different ways to overload the method
1. By changing the no. of arguments
2. By changing the datatype
2. Why method overloading is not possible by changing the return type
3. Can we overload the main method
4. method overloading with Type Promotion

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 behaviour 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**.

Different ways to overload the method

There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

> *In java, Methood Overloading is not possible by changing the return type of the method.*

---

1)Example of Method Overloading by changing the no. of arguments

In this example, we have created two overloaded methods, first sum method performs addition of two numbers and second sum method performs addition of three numbers.

```
1. class Calculation{
2.   void sum(int a,int b){System.out.println(a+b);}
3.   void sum(int a,int b,int c){System.out.println(a+b+c);}
```

```
4.
5.  public static void main(String args[]){
6.  Calculation obj=new Calculation();
7.  obj.sum(10,10,10);
8.  obj.sum(20,20);
9.
10.         }
11.        }
```

```
Output:30
         40
```

---

2)Example of Method Overloading by changing data type of argument

In this example, we have created two overloaded methods that differs in data type. The first sum method receives two integer arguments and second sum method receives two double arguments.

```
1. class Calculation{
2.  void sum(int a,int b){System.out.println(a+b);}
3.  void sum(double a,double b){System.out.println(a+b);}
4.
5.  public static void main(String args[]){
6.  Calculation obj=new Calculation();
7.  obj.sum(10.5,10.5);
8.  obj.sum(20,20);
9.
10.         }
11.        }
```

```
Output:21.0
         40
```

Que) Why Method Overloaing is not possible by changing the return type of method?

In java, method overloading is not possible by changing the return type of the method because there may occur ambiguity. Let's see how ambiguity may occur:

because there was problem:

```
1. class Calculation{
2.   int sum(int a,int b){System.out.println(a+b);}
3.   double sum(int a,int b){System.out.println(a+b);}
4.
5.   public static void main(String args[]){
6.   Calculation obj=new Calculation();
7.   int result=obj.sum(20,20); //Compile Time Error
8.
9.   }
10.        }
```

int result=obj.sum(20,20); //Here how can java determine which sum() method should be called

---

Can we overload main() method?

Yes, by method overloading. You can have any number of main methods in a class by method overloading. Let's see the simple example:

```
1. class Simple{
2.   public static void main(int a){
3.   System.out.println(a);
4.   }
5.
6.   public static void main(String args[]){
7.   System.out.println("main() method invoked");
8.   main(10);
9.   }
10.        }
```

Output:main() method invoked
10

38)Why method overloading is not possible by changing the return type in java?

Explained Above.

39) Can we overload main() method?

Explained Above.

40)What is method overriding:

Method Overriding in Java

1. Understanding problem without method overriding
2. Can we override the static method
3. method overloading vs method overriding

Having the same method in the subclass as declared in the parent class is known as **method overriding**.

In other words, If subclass provides the specific implementation of the method i.e. already provided by its parent 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.
3. must be inheritance (IS-A) relationship.

Understanding the problem without mehtod overriding

Let's understand the problem that we may face in the program if we don't use method overriding.

```
1.  class Vehicle{

2.  void run(){System.out.println("Vehicle is running");}
```

```
3.  }
4.  class Bike extends Vehicle{
5.
6.  public static void main(String args[]){
7.  Bike obj = new Bike();
8.  obj.run();
9.  }
10. }
```

Output: Vehicle is running

Problem is that I have to provide a specific implementation of run() method in subclass that is why we use method overriding.

Example of method overriding

In this example, we have defined the run method in the subclass as defined in the parent class but it has some specific implementation. The name and parameter of the method is same and there is IS-A relationship between the classes, so there is method overriding.

```
1.  class Vehicle{
2.  void run(){System.out.println("Vehicle is running");}
3.  }
4.  class Bike extends Vehicle{
5.  void run(){System.out.println("Bike is running safely");}
```

```
6.
7.  public static void main(String args[]){
8.  Bike obj = new Bike();
9.  obj.run();
10. }
```

Output: Bike is running safely

41)Can we override static method?

No, you can't override the static method because they are the part of class not object.

42)Why we cannot override static method?

It is because the static method is the part of class and it is bound with class whereas instance method is bound with object and static gets memory in class area and instance gets memory in heap.

43)Can we override the overloaded method?

Yes.

44)Difference between method Overloading and Overriding.

1) Method overloading increases the readability of the program.	Method overriding provides the specific implementation of the method that is already provided by its super class.
2) method overlaoding is occurs within the class.	Method overriding occurs in two classes that have IS-A relationship.
3) In this case, parameter must be different.	In this case, parameter must be same.

45)Can you have virtual functions in Java?

Yes, all functions in Java are virtual by default.

46)What is covariant return type?

Covariant Return Type

The covariant return type specifies that the return type may vary in the same direction

as the subclass. Before Java5, it was not possible to override any method by changing the return type. But now, since Java5, it is possible to override method by changing the return type if subclass overrides any method whose return type is Non-Primitive but it changes its return type to subclass type. Let's take a simple example:

Note: If you are beginner to java, skip this topic and return to it after OOPs concepts.

Simple example of Covariant Return Type

```
1. class A{
2. A get(){return this;}
3. }
4.
5. class B extends A{
6. B get(){return this;}
7. void message(){System.out.println("welcome to covariant return
   type");}
8.
9. public static void main(String args[]){
10. new B().get().message();
11. }
12. }
```

Output:welcome to covariant return type

As you can see in the above example, the return type of the get() method of A class is A but the return type of the get() method of B class is B. Both methods have different return type but it is method overriding. This is known as covariant return type.

47)What is final variable?

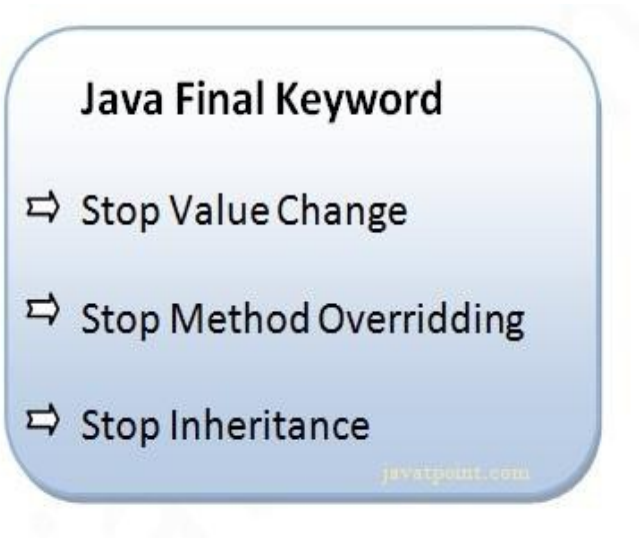
Final Keyword in Java

1. Final variable
2. Final method
3. Final class
4. Is final method inherited ?
5. Blank final variable
6. Static blank final variable
7. Final parameter
8. Can you declare a final constructor

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

1. variable
2. method
3. class

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only. We will have detailed learning of these. Let's first learn the basics of final keyword.



1) final variable

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

Example of final variable

There is a final variable speedlimit, we are going to change the value of this variable, but It can't be changed because final variable once assigned a value can never be changed.

```
1. class Bike{
2.  final int speedlimit=90;//final variable
3.
4.  void run(){
5.   speedlimit=400;
6.  }
7.
8.  public static void main(String args[]){
9.  Bike obj=new  Bike();
10. obj.run();
11. }
```

Output: Compile Time Error

48)What is final method?

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

Example of final method

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

```
3. }
4.
5. class Honda extends Bike{
6.    void run(){System.out.println("running safely with 100kmph");}
7.
8.    public static void main(String args[]){
9.    Honda honda= new Honda();
10.   honda.run();
11.   }
12.}
```

Output: Compile Time Error

49)What is final class?

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

Example of final class

```
1. final class Bike{}
2.
3. class Honda extends Bike{
4.   void run(){System.out.println("running safely with 100kmph");}
5.
6.   public static void main(String args[]){
```

```
7.    Honda honda= new Honda();
8.    honda.run();
9.    }
```

Output: Compile Time Error

50) What is blank final variable?

Q) Is final method inherited?

Ans)Yes, final method is inherited but you cannot override it. For Example:

```
1.  class Bike{
2.    final void run(){System.out.println("running...");}
3.  }
4.  class Honda extends Bike{
5.
6.     public static void main(String args[]){
7.      new Honda().run();
8.     }
9.  }
```

Output:running...

Q) What is blank final variable?

A final variable that is not initalized at the time of declaration is known as blank final

variable. If you want to create a variable that is initialized at the time of creating object and once initialized may not be changed, it is useful. For example PAN CARD number of an employee. It can be initialized only in constuctor.

Example of blank final variable

```
1. class Student{
2. int id;
3. String name;
4. final String PAN_CARD_NUMBER;
5. ...
6. }
```

51) Can we intialize blank final variable?

Yes, but only in constructor. For example:

```
1. class Bike{
2.   final int speedlimit;//blank final variable
3.      Bike(){
4.   speedlimit=70;
5.   System.out.println(speedlimit);
6.   }
7.   public Static void main(String args[]){
8.     new Bike();
9.  }
```

```
10.}
```

Output:70

Q) What is final parameter?

If you declare any parameter as final, you cannot change the value of it.

```
1.  class Bike{
2.    int cube(final int n){
3.     n=n+2;//can't be changed as n is final
4.     n*n*n;
5.    }
6.    public Static void main(String args[]){
7.      Bike b=new Bike();
8.      b.cube(5);
9.   }
10.}
```

Output: Compile Time Error

Q) Can we declare a constructor final?

No, because constructor is never inherited.

52)Can you declare the main method as final?

Yes, such as, public static final void main(String[] args) {}.

53) What is Runtime Polymorphism?

Runtime Polymorphism

Runtime polymorphism or **Dynamic Method Dispatch** is a process in which a call to an overridden method is resolved at runtime rather than compile-time.

In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.
Let's first understand the upcasting before Runtime Polymorphism.

Upcasting

When reference variable of Parent class refers to the object of Child class, it is known as upcasting. For example:


```
1. class A{}

2. class B extends A{}

1. A a=new B();//upcasting
```

Example of Runtime Polymorphism

In this example, we are creating two classes Bike and Splendar. Splendar class extends Bike class and overrides its run() method. We are calling the run method by the reference variable of Parent class. Since it refers to the subclass object and subclass method overrides the Parent class method, subclass method is invoked at runtime.
Since method invocation is determined by the JVM not compiler, it is known as runtime polymorphism.

```
1. class Bike{

2.   void run(){System.out.println("running");}
```

```
3.  }
4.  class Splender extends Bike{
5.    void run(){System.out.println("running safely with 60km");}
6.
7.    public static void main(String args[]){
8.      Bike b = new Splender();//upcasting
9.      b.run();
10.   }
11. }
```

Output: running safely with 60km.

54)Can you achieve Runtime Polymorphism by data members?

Runtime Polymorphism with data member

Method is overriden not the datamembers, so runtime polymorphism can't be achieved by data members.
In the example given below, both the classes have a datamember speedlimit, we are accessing the datamember by the reference variable of Parent class which refers to the subclass object. Since we are accessing the datamember which is not overridden, hence it will access the datamember of Parent class always.

Rule: Runtime polymorphism can't be achieved by data members.

```
1.  class Bike{
2.   int speedlimit=90;
3.  }
```

```
4. class Honda extends Bike{
5.  int speedlimit=150;
6.
7.  public static void main(String args[]){
8.   Bike obj=new Honda();
9.   System.out.println(obj.speedlimit);//90
10. }
```

Output: 90

Runtime Polymorphism with Multilevel Inheritance

Let's see the simple example of Runtime Polymorphism with multilevel inheritance.

```
1. class Animal{
2. void eat(){System.out.println("eating");}
3. }
4.
5. class Dog extends Animal{
6. void eat(){System.out.println("eating fruits");}
7. }
8.
9. class BabyDog extends Dog{
10. void eat(){System.out.println("drinking milk");}
11.
```

```
12. public static void main(String args[]){
13. Animal a1,a2,a3;
14. a1=new Animal();
15. a2=new Dog();
16. a3=new BabyDog();
17.
18. a1.eat();
19. a2.eat();
20. a3.eat();
21. }
22. }
```

Output: eating
eating fruits
drinking Milk

Try for Output

```
1. class Animal{
2. void eat(){System.out.println("animal is eating...");}
3. }
4.
5. class Dog extends Animal{
6. void eat(){System.out.println("dog is eating...");}
7. }
```

```
8.
9.  class BabyDog extends Dog{
10. public static void main(String args[]){
11. Animal a=new BabyDog();
12. a.eat();
13. }}
```

Output: Dog is eating

Since, BabyDog is not overriding the eat() method, so eat() method of Dog class is invoked.

55) What is the difference between static binding and dynamic binding?

Static Binding and Dynamic Binding

Connecting a method call to the method body is known as binding.

There are two types of binding

1. static binding (also known as early binding).
2. dynamic binding (also known as late binding).

Understanding Type

Let's understand the type of instance.

1) variables have a type

Each variable has a type, it may be primitive and non-primitive.

```
1.  int data=30;
```

Here data variable is a type of int.

2) References have a type

```
1.  class Dog{
2.   public static void main(String args[]){
```

```
3.    Dog d1;//Here d1 is a type of Dog
4.   }
5.  }
```

3) Objects have a type

An object is an instance of particular java class,but it is also an instance of its superclass.

```
1.  class Animal{}
2.
3.  class Dog extends Animal{
4.   public static void main(String args[]){
5.    Dog d1=new Dog();
6.   }
7.  }
```

Here d1 is an instance of Dog class, but it is also an instance of Animal.

static binding

When type of the object is determined at compiled time(by the compiler), it is known as static binding.

If there is any private, final or static method in a class, there is static binding.

Example of static binding

```
1.  class Dog{
2.   private void eat(){System.out.println("dog is eating...");}
3.
4.   public static void main(String args[]){
```

```
5.    Dog d1=new Dog();
6.    d1.eat();
7.   }
8.  }
```

Dynamic binding

When type of the object is determined at run-time, it is known as dynamic binding.

Example of dynamic binding:

```
1.  class Animal{
2.   void eat(){System.out.println("animal is eating...");}
3.  }
4.
5.  class Dog extends Animal{
6.   void eat(){System.out.println("dog is eating...");}
7.
8.   public static void main(String args[]){
9.    Animal a=new Dog();
10.   a.eat();
11.  }
12. }
```

Output:dog is eating...

In the above example object type cannot be determined by the compiler, because the instance of Dog is also an instance of Animal.So compiler doesn't know its type, only

its base type.

56) What is abstraction?

Abstraction

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 Abstaction

There are two ways to achieve abstraction in java

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

57) What is the difference between abstraction and encapsulation?

Abstraction hides the implementation details whereas encapsulation hides the data.**more details...**
Abstraction lets you focus on what the object does instead of how it does it.

58) What is abstract class?

Abstract class

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 to declare the abstract class

```
1. 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

```
1. 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. It implementation is provided by the Honda class.

```
1. abstract class Bike{
2.   abstract void run();
3. }
4.
5. class Honda extends Bike{
6. void run(){System.out.println("running safely..");}
7.
8. public static void main(String args[]){
9.  Bike obj = new Honda();
10. obj.run();
11.}
```

Output:running safely..

Understanding the real scenario of abstract class

In this example, Shape is the abstract class, its implementation is provided by the Rectangle and Circle classes. Mostly, we don't know about the implementation class (i.e. hidden to the end user) and object of the implementation class is provided by the **factory method**.
A **factory method** is the method that returns the instance of the class. We will learn about the factory method later.
In this example, if you create the instance of Rectangle class, draw method of Rectangle class will be invoked.

```
1.  abstract class Shape{
2. abstract void draw();
```

```
}

class Rectangle extends Shape{
void draw(){System.out.println("drawing rectangle");}
}

class Circle extends Shape{
void draw(){System.out.println("drawing circle");}
}

class Test{
public static void main(String args[]){
Shape s=new Circle();
//In real scenario, Object is provided through factory method
s.draw();
}
}
```

Output:drawing circle

Abstract class having constructor, data member, methods etc.

Note: An abstract class can have data member, abstract method, method body, constructor and even main() method.

```
//example of abstract class that have method body
```

```
2.
3.  abstract class Bike{
4.    abstract void run();
5.    void changeGear(){System.out.println("gear changed");}
6.  }
7.
8.  class Honda extends Bike{
9.  void run(){System.out.println("running safely..");}
10.
11.  public static void main(String args[]){
12.   Bike obj = new Honda();
13.   obj.run();
14.   obj.changeGear();
15.  }
16. }
```

Output:running safely..
gear changed

```
1.  //example of abstract class having constructor, field and method
2.  abstract class Bike
3.  {
4.   int limit=30;
```

```
5.    Bike(){System.out.println("constructor is invoked");}
6.    void getDetails(){System.out.println("it has two wheels");}
7.    abstract void run();
8.  }
9.
10. class Honda extends Bike{
11.   void run(){System.out.println("running safely..");}
12.
13.   public static void main(String args[]){
14.    Bike obj = new Honda();
15.    obj.run();
16.    obj.getDetails();
17.    System.out.println(obj.limit);
18.  }
19. }
```

Output:constructor is invoked
running safely..
it has two wheels
30

Rule: If there is any abstract method in a class, that class must be abstract.

```
1.  class Bike{
2.  abstract void run();
3.  }
```

Output:compile time error

Rule: If you are extending any abstact class that have abstract method, you must either provide the implementation of the method or make this class abstract.

Another real scenario of abstract class

The abstract class can also be used to provide some implementation of the interface. In such case, the end user may not be forced to override all the methods of the interface.

Note: If you are beginner to java, learn interface first and skip this example.

```
1. interface A{
2. void a();
3. void b();
4. void c();
5. void d();
6. }
7.
8. abstract class B implements A{
9. public void c(){System.out.println("I am C");}
10.}
11.
12. class M extends B{
13. public void a(){System.out.println("I am a");}
14. public void b(){System.out.println("I am b");}
15. public void d(){System.out.println("I am d");}
```

```
16. }
17.
18. class Test{
19. public static void main(String args[]){
20. A a=new M();
21. a.a();
22. a.b();
23. a.c();
24. a.d();
25. }}
```

Output:I am a
I am b
I am c
I am d

59) Can there be any abstract method without abstract class?

No, if there is any abstract method in a class, that class must be abstract.

60) Can you use abstract and final both with a method?

No, because abstract method needs to be overridden whereas you can't override final method.

61) Is it possible to instantiate the abstract class?

No, abstract class can never be instantiated.

62) What is interface?

Interface

An **interface** is a blueprint of a class. It has static constants and abstract methods.

The interface is **a mechanism to achieve fully abstraction** in java. There can be only abstract methods in the interface. It is used to achieve fully abstraction and multiple inheritance in Java.
Interface also **represents IS-A relationship**.
It cannot be instantiated just like abstract class.

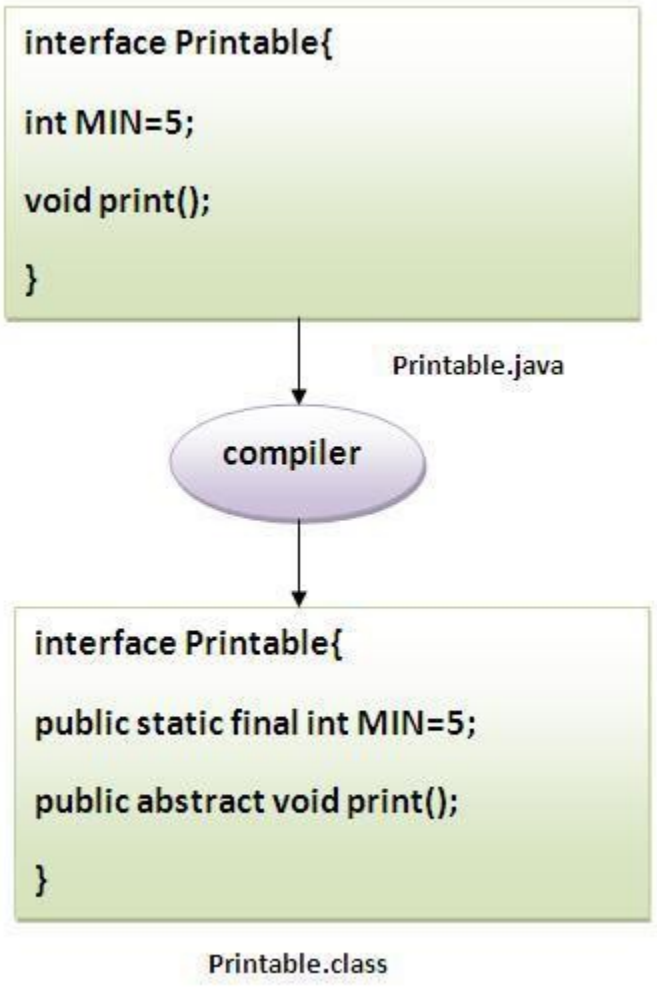
Why use Interface?

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

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

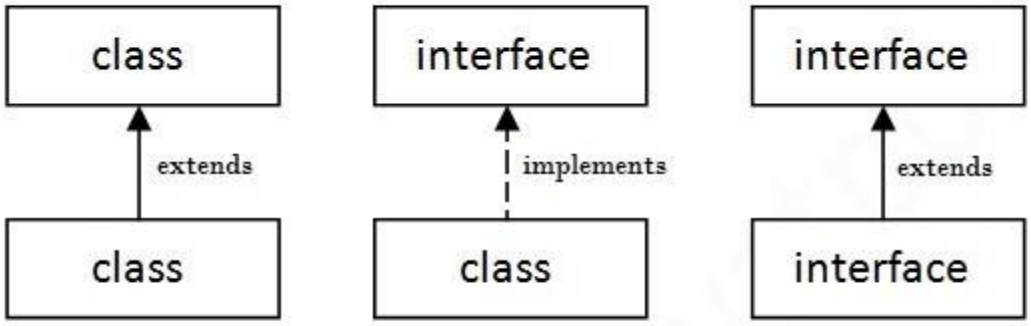
The java compiler adds public and abstract keywords before the interface method and public, static and final keywords before data members.

In other words, Interface fields are public, static and final bydefault, and methods are public and abstract.



Understanding relationship between classes and interfaces

As shown in the figure given below, a class extends another class, an interface extends another interface but a **class implements an interface**.



Simple example of Interface

In this exmple, Printable interface have only one method, its implemenation is provided in the A class.

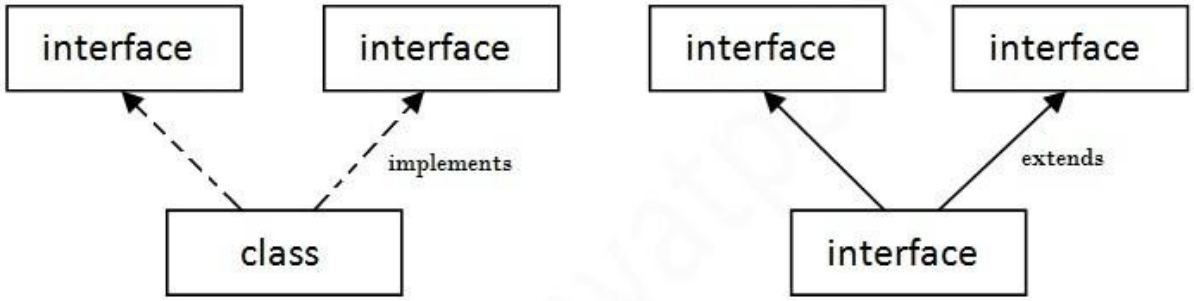
```
1. interface printable{
2. void print();
3. }
4.
5. class A implements printable{
6. public void print(){System.out.println("Hello");}
7.
8. public static void main(String args[]){
9. A obj = new A();
10. obj.print();
11. }
```

```
12. }
```

Output: Hello

Multiple inheritance in Java by interface

If a class implements multiple interfaces, or an interface extends multiple interfaces i.e. known as multiple inheritance.



Multiple Inheritance in Java

```
1. interface Printable{
2. void print();
3. }
4.
5. interface Showable{
6. void show();
7. }
8.
9. class A implements Printable,Showable{
10.
11. public void print(){System.out.println("Hello");}
```

```
12. public void show(){System.out.println("Welcome");}
13.
14. public static void main(String args[]){
15. A obj = new A();
16. obj.print();
17. obj.show();
18.  }
19. }
```

Output: Hello
Welcome

Q) Multiple inheritance is not supported in case of class but it is supported in case of interface, why?

As we have explained in the inheritance chapter, multiple inheritance is not supported in case of class. But it is supported in case of interface because there is no ambiguity as implmentation is provided by the implementation class. For example:

```
1. interface Printable{
2. void print();
3. }
4.
5. interface Showable{
6. void print();
```

```
}

class A implements Printable,Showable{

public void print(){System.out.println("Hello");}

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

Output: Hello

As you can see in the above example, Printable and Showable interface have same methods but its implementation is provided by class A, so there is no ambiguity.

Note: A class implements interface but One interface extends another interface .

```
interface Printable{
void print();
}

interface Showable extends Printable{
void show();
```

```
7.  }
8.
9.  class A implements Showable{
10.
11. public void print(){System.out.println("Hello");}
12. public void show(){System.out.println("Welcome");}
13.
14. public static void main(String args[]){
15. A obj = new A();
16. obj.print();
17. obj.show();
18.  }
19. }
```

Output: Hello
Welcome

63) Can you declare an interface method static?

No, because methods of an interface is abstract bydefault, and static and abstract keywords can't be used together.

64)Can an Interface be final?

No, because its implementation is provided by another
class.

65) What is marker interface?

An interface that have no data member and method is known as a marker interface.For example Serializable,Cloneable etc.

An interface that have no member is known as marker or tagged interface. For example: Serializable, Cloneable, Remote etc. They are used to provide some essential information to the JVM so that JVM may perform some useful operation.

```
1. //How Serializable interface is written?
2.
3. public interface Serializable{
4. }
```

Nested Interface

Note: An interface can have another interface i.e. known as nested interface. We will learn it in detail in the nested classes chapter. For example:

```
1. interface printable{
2.  void print();
3.  interface MessagePrintable{
4.    void msg();
5.  }
6. }
```

66) What is difference between abstract class and interface?

1)An abstract class can have method body (non-abstract methods).	Interface have only abstract methods.
2)An abstract class can have instance variables.	An interface cannot have instance variables.
3)An abstract class can have constructor.	Interface cannot have constructor.
4)An abstract class can have static methods.	Interface cannot have static methods.

5)You can extends one abstract class.	You can implement multiple interfaces.

67)Can we define private and protected modifiers for variables in interfaces?

No, they are implicitely

public.

68)When can an object reference be cast to an interface reference?

An object reference can be cast to an interface reference when the object implements the referenced interface.

69)What is package?

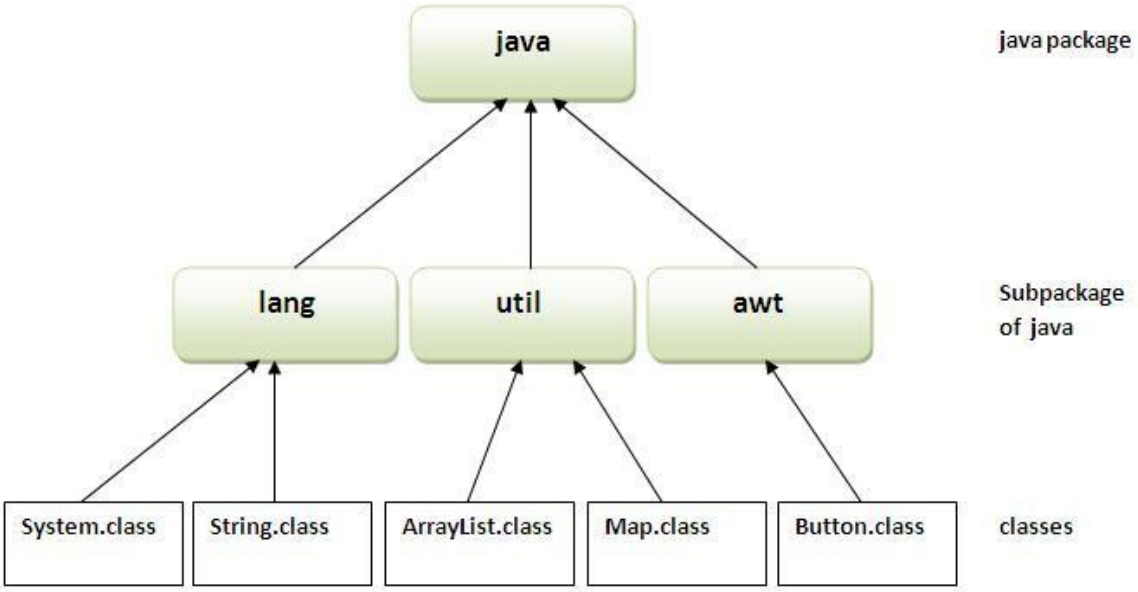
Package

A **package** is a group of similar types of classes, interfaces and sub-packages.

Package can be categorized in two form, built-in package and user-defined package. There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc. Here, we will have the detailed learning of creating and using user-defined packages.

Advantage of Package

- Package is used to categorize the classes and interfaces so that they can be easily maintained.
- Package provids access protection.
- Package removes naming collision.



Simple example of package

The **package keyword** is used to create a package.

```
1. //save as Simple.java
2.
3. package mypack;
4. public class Simple{
5.  public static void main(String args[]){
6.      System.out.println("Welcome to package");
7.    }
8. }
```

How to compile the Package (if not using IDE)

If you are not using any IDE, you need to follow the **syntax** given below:

1. javac -d directory javafilename

For **example**

1. javac -d . Simple.java

The -d switch specifies the destination where to put the generated class file. You can use any directory name like /home (in case of Linux), d:/abc (in case of windows) etc. If you want to keep the package within the same directory, you can use . (dot).

How to run the Package (if not using IDE)

You need to use fully qualified name e.g. mypack.Simple etc to run the class.

To Compile: javac -d . Simple.java
To Run: java mypack.Simple
Output: Welcome to package

The -d is a switch that tells the compiler where to put the class file i.e. it represents destination. The . represents the current folder.

How to access package from another package?

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

1. import package.*;
2. import package.classname;
3. fully qualified name.

Using packagename.*

If you use package.* then all the classes and interfaces of this package will be accessible but not subpackages.

The import keyword is used to make the classes and interface of another package accessible to the current package.

Example of package that import the packagename.*

```
1. //save by A.java
2.
3. package pack;
4. public class A{
5.   public void msg(){System.out.println("Hello");}
6. }
```

```
1. //save by B.java
2.
3. package mypack;
4. import pack.*;
5.
6. class B{
7.   public static void main(String args[]){
8.    A obj = new A();
9.    obj.msg();
10.  }
11.}
```

Output: Hello

Using packagename.classname

If you import package.classname then only declared class of this package will be accessible.

Example of package by import package.classname

```
1. //save by A.java
2.
3. package pack;
4. public class A{
5.   public void msg(){System.out.println("Hello");}
6. }
```

```
1. //save by B.java
2.
3. package mypack;
4. import pack.A;
5.
6. class B{
7.   public static void main(String args[]){
8.    A obj = new A();
9.    obj.msg();
10.  }
11.}
```

Output: Hello

Using fully qualified name

If you use fully qualified name then only declared class of this package will be accessible. Now there is no need to import. But you need to use fully qualified name every time when

you are accessing the class or interface.
It is generally used when two packages have same class name e.g. java.util and java.sql packages contain Date class.

Example of package by import fully qualified name

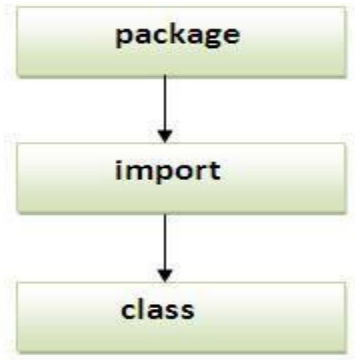
```
1. //save by A.java
2.
3. package pack;
4. public class A{
5.   public void msg(){System.out.println("Hello");}
6. }
1. //save by B.java
2.
3. package mypack;
4. class B{
5.   public static void main(String args[]){
6.    pack.A obj = new pack.A();//using fully qualified name
7.    obj.msg();
8.   }
9. }
```

Output: Hello

Note: If you import a package, subpackages will not be imported.

If you import a package, all the classes and interface of that package will be imported excluding the classes and interfaces of the subpackages. Hence, you need to import the subpackage as well.

Note: Sequence of the program must be package then import then class.



Subpackage

Package inside the package is called the **subpackage**. It should be created **to categorize the package further**.

Let's take an example, Sun Microsystem has definded a package named java that contains many classes like System, String, Reader, Writer, Socket etc. These classes represent a particular group e.g. Reader and Writer classes are for Input/Output operation, Socket and ServerSocket classes are for networking etc and so on. So, Sun has subcategorized the java package into subpackages such as lang, net, io etc. and put the Input/Output related classes in io package, Server and ServerSocket classes in net packages and so on.

The standard of defining package is domain.company.package e.g. com.javatpoint.bean or org.sssit.dao.

Example of Subpackage

```
1. package com.javatpoint.core;

2. class Simple{

3.   public static void main(String args[]){
```

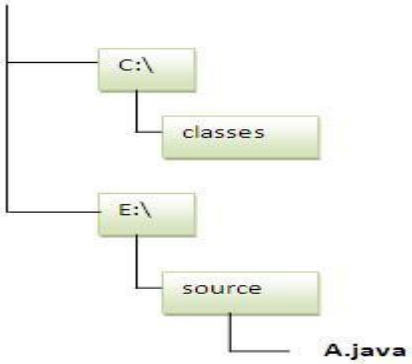
```
4.      System.out.println("Hello subpackage");
5.    }
6.  }
```

To Compile: javac -d . Simple.java
To Run: java com.javatpoint.core.Simple

Output: Hello subpackage

How to send the class file to another directory or drive?

There is a scenario, I want to put the class file of A.java source file in classes folder of c: drive. For example:




```
1.  //save as Simple.java
2.
3.  package mypack;
4.  public class Simple{
5.   public static void main(String args[]){
6.      System.out.println("Welcome to package");
7.    }
8.  }
```

To Compile:

e:\sources> javac -d c:\classes

Simple.java

To Run:

To run this program from e:\source directory, you need to set classpath of the directory where the class file resides.
e:\sources> set classpath=c:\classes;.;
e:\sources> java mypack.Simple

Another way to run this program by -classpath switch of java:

The -classpath switch can be used with javac and java tool.

To run this program from e:\source directory, you can use -classpath switch of java that tells where to look for class file. For example:
e:\sources> java -classpath c:\classes mypack.Simple
Output: Welcome to package

Ways to load the class files or jar files

There are two ways to load the class files temporary and permanent.

- Temporary
    - By setting the classpath in the command prompt
    - By -classpath switch
- Permanent
    - By setting the classpath in the environment variables
    - By creating the jar file, that contains all the class files, and copying the jar file in the jre/lib/ext folder.

Rule: There can be only one public class in a java source file and it must be saved by the public class name.

1. //save as C.java otherwise Compilte Time Error
2.

```
3. class A{}
4. class B{}
5. public class C{}
```

How to put two public classes in a package?

If you want to put two public classes in a package, have two java source files containing one public class, but keep the package name same. For example:

```
1. //save as A.java
2.
3. package javatpoint;
4. public class A{}
```

```
1. //save as B.java
2.
3. package javatpoint;
4. public class B{}
```

What is static import feature of Java5?

Static Import:

The static import feature of Java 5 facilitate the java programmer to access any static member of a class directly. There is no need to qualify it by the class name.

Advantage of static import:

- Less coding is required if you have access any static member of a class oftenly.

Disadvantage of static import:

- If you overuse the static import feature, it makes the program unreadable and unmaintainable.

Simple Example of static import

```
1. import static java.lang.System.*;
2. class StaticImportExample{
3.   public static void main(String args[]){
4.
5.    out.println("Hello");//Now no need of System.out
6.    out.println("Java");
7.
8.  }
9. }
10.
```

Output: Hello
Java

What is the difference between import and static import?

The import allows the java programmer to access classes of a package without package qualification whereas the static import feature allows to access the static members of a class without the class qualification. The import provides accessibility to classes and interface whereas static import provides accessibility to static members of the class.

What about package class?

Package class

The package class provides methods to get information about the specification and implementation of a package. It provides methods such as getName(), getImplementationTitle(), getImplementationVendor(), getImplementationVersion() etc.

Example of Package class

In this example, we are printing the details of java.lang package by invoking the methods of package class.

```
1. class PackageInfo{
2. public static void main(String args[]){
3.
4. Package p=Package.getPackage("java.lang");
5.
6. System.out.println("package name: "+p.getName());
7.
8. System.out.println("Specification Title: "+p.getSpecificationTitle());
9. System.out.println("Specification Vendor: "+p.getSpecificationVendor());
10. System.out.println("Specification Version: "+p.getSpecificationVersion());
11.
12. System.out.println("Implementaion Title: "+p.getImplementationTitle());
13. System.out.println("Implementation Vendor: "+p.getImplementationVendor());
14. System.out.println("Implementation Version: "+p.getImplementationVersion());
15. System.out.println("Is sealed: "+p.isSealed());
16.
```

```
17.

18. }

19.}
```

Output:package name: java.lang

Specification Title: Java Plateform API Specification

Specification Vendor: Sun Microsystems, Inc.

Specification Version: 1.6

Implemenation Title: Java Runtime Environment

Implemenation Vendor: Sun Microsystems, Inc.

Implemenation Version: 1.6.0_30

IS sealed: false

70)Do I need to import java.lang package any time? Why ?

No. It is by default loaded internally by the
JVM.

71)Can I import same package/class twice? Will the JVM load the package twice at runtime?

One can import the same package or same class multiple times. Neither compiler nor JVM complains about it.But the JVM will internally load the class only once no matter how many times you import the same class.

73) What is Exception Handling?

Exception Handling in Java

The exception handling is one of the powerful mechanism provided in java. It provides the mechanism to handle the runtime errors so that normal flow of the application can be maintained.

In this page, we will know about exception, its type and the difference between checked and unchecked exceptions.

Exception

- **Dictionary Meaning:**Exception is an abnormal condition.
- In java, exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

Exception Handling

Exception Handling is a mechanism to handle runtime errors.

Types of Exception:

There are mainly two types of exceptions: checked and unchecked where error is considered as unchecked exception. The sun microsystem says there are three types of exceptions:

1. **Checked Exception**
2. **Unchecked Exception**
3. **Error**

74)What is difference between Checked Exception and Unchecked Exception?

1)Checked Exception

The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions e.g.IOException, SQLException etc. Checked exceptions are checked at compile-time.

2)Unchecked Exception

The classes that extend RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time rather they are checked at runtime.

3)Error

Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

75)What is the base class for Error and Exception?

Throwable

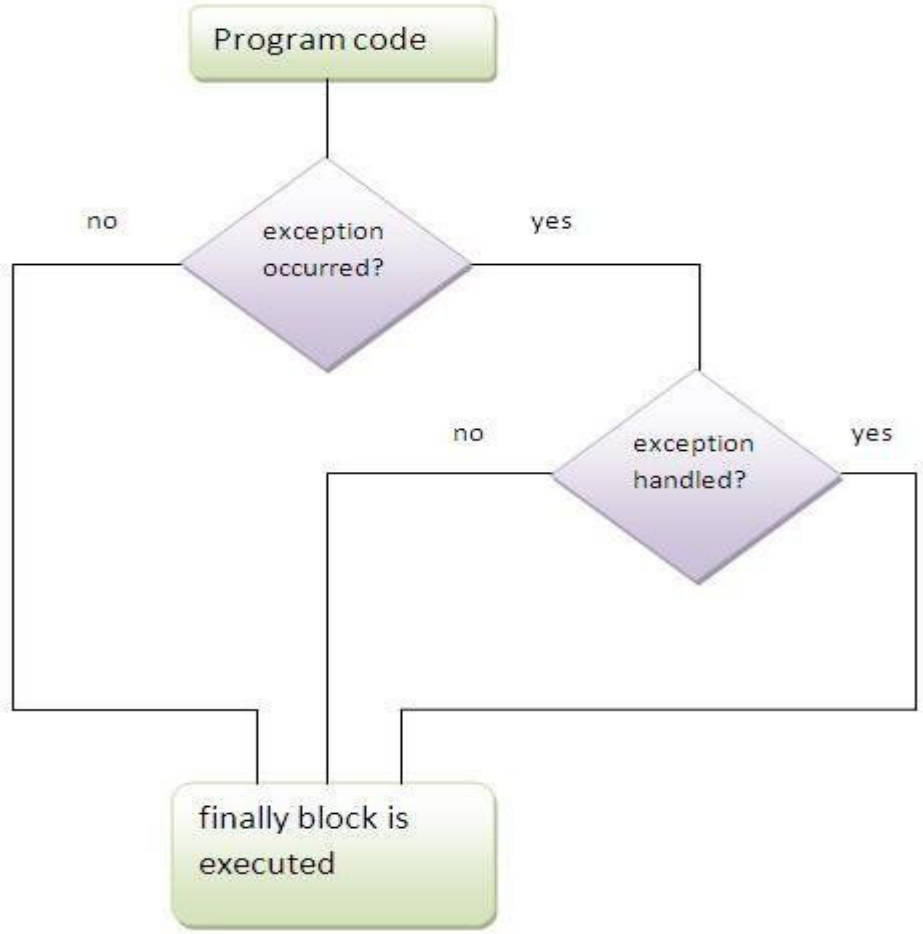
.

76)Is it necessary that each try block must be followed by a catch block?

It is not necessary that each try block must be followed by a catch block. It should be followed by either a catch block OR a finally block. And whatever exceptions are likely to be thrown should be declared in the throws clause of the method.

77)What is finally block?

The finally block is a block that is always executed. It is mainly used to perform some important tasks such as closing connection, stream etc.



Note: Before terminating the program, JVM executes finally block(if any).

Note:finally must be followed by try or catch block.

78)Can finally block be used without catch?

Yes, by try block. finally must be followed by either try or catch.

79)Is there any case when finally will not be executed?

- finally block will not be executed if program exits(either by calling System.exit() or by causing a fatal error that causes the process to abort).

80)What is difference between throw and throws?

1)throw is used to explicitly throw an exception.	throws is used to declare an exception.
2)checked exceptions can not be propagated with throw only.	checked exception can be propagated with throws.
3)throw is followed by an instance.	throws is followed by class.
4)throw is used within the method.	throws is used with the method signature.
5)You cannot throw multiple exception	You can declare multiple exception e.g. public void method()throws IOException,SQLException.

81)Can an exception be rethrown?

Yes.

82)Can subclass overriding method declare an exception if parent class method doesn't throw an exception ?

Yes but only unchecked exception not checked.

83)What is exception propagation ?

Forwarding the exception object to the invoking method is known as exception propagation.

84)What is the meaning of immutable in terms of String?

The simple meaning of immutable is unmodifiable or unchangeable. Once string object has been created, its value can't be changed.

85)Why string objects are immutable in java?

Because java uses the concept of string literal. Suppose there are 5 reference variables,all referes to one object "sachin".If one reference variable changes the value of the object, it will be affected to all the reference variables. That is why string objects are immutable in java.

86)How many ways we can create the string object?

How to create String object?

There are two ways to create String

object:

1. **By string literal**
2. **By new keyword**

1) String literal:

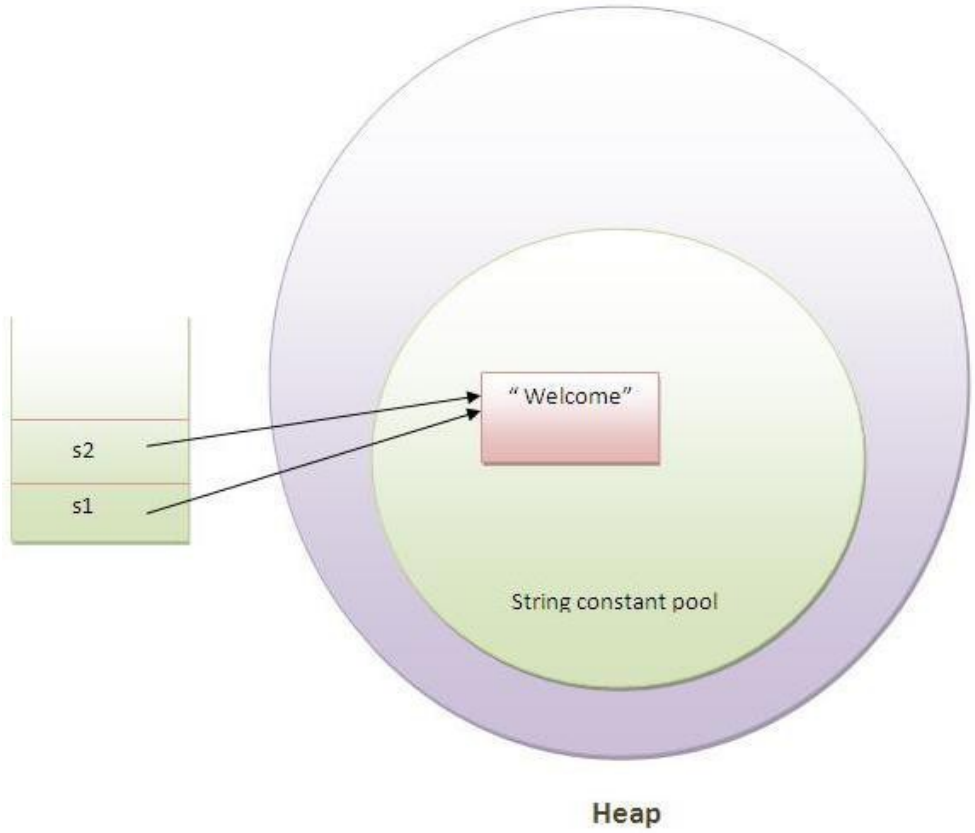
String literal is created by double quote.For
Example:

```
1. String s="Hello";
```

Each time you create a string literal, the JVM checks the string constant pool first. If the string already exists in the pool, a reference to the pooled instance returns. If the string does not exist in the pool, a new String object instantiates, then is placed in the pool.For example:

```
1. String s1="Welcome";

2. String s2="Welcome";//no new object will be created
```

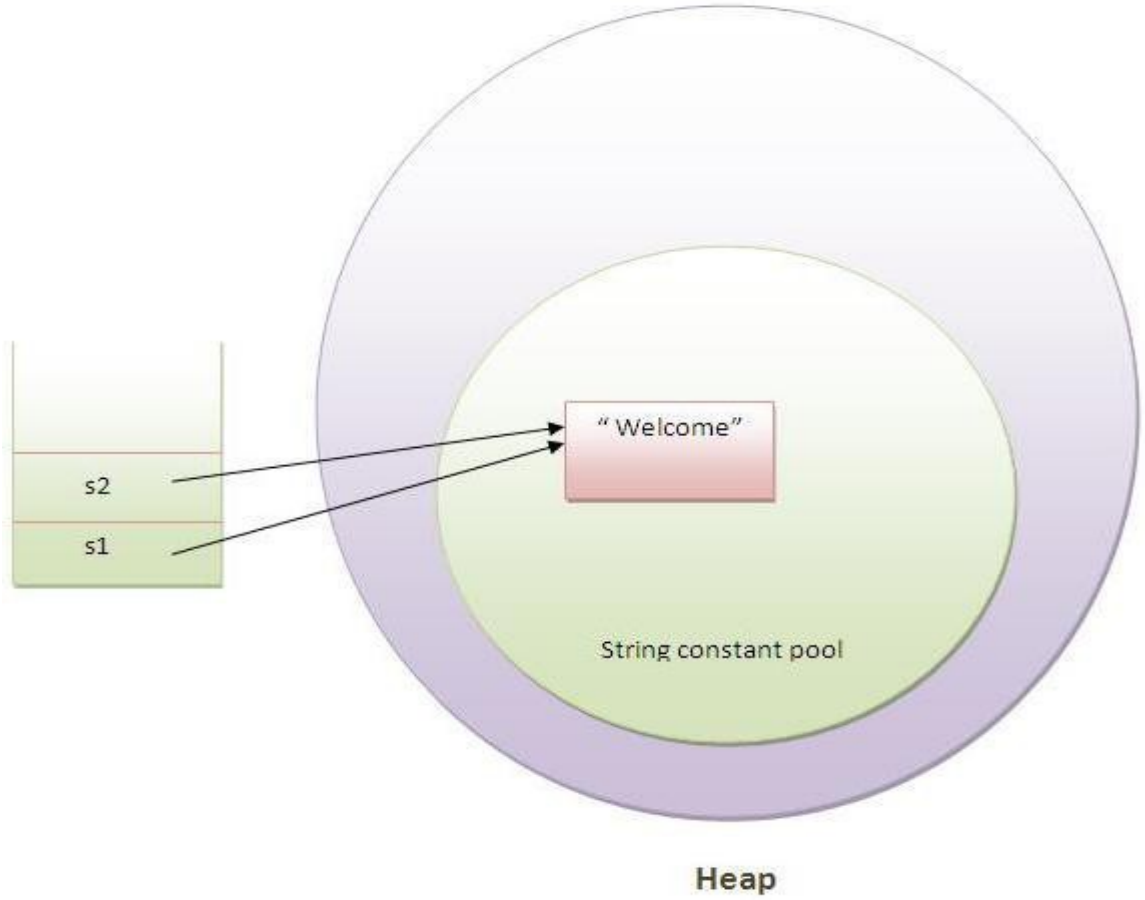


In the above example only one object will be created.First time JVM will find no string object with the name "Welcome" in string constant pool,so it will create a new object.Second time it will find the string with the name "Welcome" in string constant pool,so it will not create new object whether will return the reference to the same instance.

Note: **String objects are stored in a special memory area known as string constant pool inside the Heap memory.**

Each time you create a string literal, the JVM checks the string constant pool first. If the string already exists in the pool, a reference to the pooled instance returns. If the string does not exist in the pool, a new String object instantiates, then is placed in the pool.For example:

1. String s1="Welcome";
2. String s2="Welcome";//no new object will be created



In the above example only one object will be created.First time JVM will find no string object with the name "Welcome" in string constant pool,so it will create a new object.Second time it will find the string with the name "Welcome" in string constant pool,so it will not create new object whether will return the reference to the same instance.

1. ***Note:*** **String objects are stored in a special memory area known as string constant pool inside the Heap memory.**

2) By new keyword:

1. String s=**new** String("Welcome");//creates two objects and one reference variable

87)How many objects will be created in the following code?

1. String s1="Welcome";
2. String s2="Welcome";
3. String s3="Welcome";

Only one
object

88)Why java uses the concept of string literal?

To make Java more memory efficient (because no new objects are created if it exists already in string constant pool)

89)How many objects will be created in the following code?

```
1. String s=new String("Welcome");
```

Two objects, one in string constant pool and other in non-pool(heap)

90)What is the basic difference between string and stringbuffer object?

String is an immutable object. StringBuffer is a mutable object.

91)What is the difference between StringBuffer and StringBuilder ?

StringBuffer is synchronized whereas StringBuilder is not synchronized.

92)How can we create immutable class in java ?

How to create Immutable class?

There are many immutable classes like String, Boolean, Byte, Short, Integer, Long, Float, Double etc. In short, all the wrapper classes and String class is immutable. We can also create immutable class by creating final class that have final data members as the example given below:

Example to create Immutable class

In this example, we have created a final class named Employee. It have one final datamember, a parameterized constructor and getter method.

```
1. public final class Employee{
2. final String pancardNumber;
3.
4. public Employee(String pancardNumber){
5. this.pancardNumber=pancardNumber;
```

Interrupting a Thread:

If any thread is in sleeping or waiting state (i.e. sleep() or wait() is invoked), calling the interrupt() method on the thread, breaks out the sleeping or waiting state throwing InterruptedException. If the thread is not in the sleeping or waiting state, calling the interrupt() method performs normal behaviour and doesn't interrupt the thread but sets the interrupt flag to true. Let's first see the methods provided by the Thread class for thread interruption.

The 3 methods provided by the Thread class for interrupting a thread

- **public void interrupt()**
- **public static boolean interrupted()**
- **public boolean isInterrupted()**

Example of interrupting a thread that stops working

In this example, after interrupting the thread, we are propagating it, so it will stop working. If we don't want to stop the thread, we can handle it where sleep() or wait() method is invoked. Let's first see the example where we are propagating the exception.

```
1. class A extends Thread{
2. public void run(){
```

```
3.  try{
4.  Thread.sleep(1000);
5.  System.out.println("task");
6.  }catch(InterruptedException e){
7.  throw new RuntimeException("Thread interrupted..."+e);
8.  }
9.  }
10. public static void main(String args[]){
11. A t1=new A();
12. t1.start();
13. try{
14. t1.interrupt();
15. }catch(Exception e){System.out.println("Exception handled "+e);}
16.   }
17. }
```

```
1.  <strong>Output:</strong>Exception in thread-0
2.        java.lang.RuntimeException: Thread interrupted...
3.        java.lang.InterruptedException: sleep interrupted
4.        at A.run(A.java:7)
```

Example of interrupting a thread that doesn't stop working

In this example, after interrupting the thread, we handle the exception, so it will break out the sleeping but will not stop working.

```
1. class A extends Thread{
2. public void run(){
3. try{
4. Thread.sleep(1000);
5. System.out.println("task");
6. }catch(InterruptedException e){
7. System.out.println("Exception handled "+e);
8. }
9. System.out.println("thread is running...");
10. }
11.
12. public static void main(String args[]){
13. A t1=new A();
14. t1.start();
15.
16. t1.interrupt();
```

```
17.
18. }
19. }
```

```
1. <strong>Output:</strong>Exception handled
2.     java.lang.InterruptedException: sleep interrupted
3.     thread is running...
```

Example of interrupting thread that behaves normally

If thread is not in sleeping or waiting state, calling the interrupt() method sets the interrupted flag to true that can be used to stop the thread by the java programmer later.

```
1. class A extends Thread{
2.
3. public void run(){
4. for(int i=1;i<=5;i++)
5. System.out.println(i);
6. }
7.
8. public static void main(String args[]){
```

```
9.  A t1=new A();
10. t1.start();
11.
12. t1.interrupt();
13.
14. }
15. }
```

```
1.  <strong>Output:</strong>1
2.      2
3.      3
4.      4
5.      5
```

What about isInterrupted and interrupted method?

The isInterrupted() method returns the interrupted flag either true or false. The static interrupted() method returns the interrupted flag afterthat it sets the flag to false if it is true.

```
1.  class InterruptedDemo extends Thread{
2.
3.  public void run(){
```

```
4.  for(int i=1;i<=2;i++){
5.  if(Thread.interrupted()){
6.  System.out.println("code for interrupted thread");
7.  }
8.  else{
9.  System.out.println("code for normal thread");
10. }
11.
12. }//end of for loop
13. }
14.
15. public static void main(String args[]){
16.
17. InterruptedDemo t1=new InterruptedDemo();
18. InterruptedDemo t2=new InterruptedDemo();
19.
20. t1.start();
21. t1.interrupt();
22.
23. t2.start();
24.
25. }
```

```
26. }
```

```
1. <strong>Output:</strong>Code for interrupted thread
2.         code for normal thread
3.         code for normal thread
4.         code for normal thread
```

111) What is synchronization?

Synchronization

Synchronization is the capabilility of control the access of multiple threads to any shared resource. Synchronization is better in case we want only one thread can access the shared resource at a time.

Why use Synchronization?

The synchronization is mainly used to

1. To prevent thread interference.
2. To prevent consistency problem.

Types of Synchronization

There are two types of synchronization

1. Process Synchronization
2. Thread Synchronization

Here, we will discuss only thread synchronization.

Thread Synchronization

There are two types of thread synchronization mutual exclusive and inter-thread communication.

- Mutual Exclusive
    1. Synchronized method.

2. Synchronized block.
3. static synchronization.

- Cooperation (Inter-thread communication)

Mutual Exclusive

Mutual Exclusive helps keep threads from interfering with one another while sharing data. This can be done by three ways in java:

1. by synchronized method
2. by synchronized block
3. by static synchronization

Understanding the concept of Lock

Synchronization is built around an internal entity known as the lock or monitor.Every object has an lock associated with it. By convention, a thread that needs consistent access to an object's fields has to acquire the object's lock before accessing them, and then release the lock when it's done with them.

From Java 5 the package java.util.concurrent.locks contains several lock implementations.

Understanding the problem without Synchronization

In this example, there is no synchronization, so output is inconsistent. Let's see the example:

1. Class Table{

```
2.
3.  void printTable(int n){//method not synchronized
4.    for(int i=1;i<=5;i++){
5.      System.out.println(n*i);
6.      try{
7.       Thread.sleep(400);
8.      }catch(Exception e){System.out.println(e);}
9.    }
10.
11.  }
12. }
13.
14. class MyThread1 extends Thread{
15. Table t;
16. MyThread1(Table t){
17. this.t=t;
18. }
19. public void run(){
20. t.printTable(5);
21. }
22.
```

```
23. }
24. class MyThread2 extends Thread{
25. Table t;
26. MyThread2(Table t){
27. this.t=t;
28. }
29. public void run(){
30. t.printTable(100);
31. }
32. }
33.
34. class Use{
35. public static void main(String args[]){
36. Table obj = new Table();//only one object
37. MyThread1 t1=new MyThread1(obj);
38. MyThread2 t2=new MyThread2(obj);
39. t1.start();
40. t2.start();
41. }
42. }
```

Output: 5
100
10

200
15
300
20
400
25
500

Solution by synchronized method

- If you declare any method as synchronized, it is known as synchronized method.
- Synchronized method is used to lock an object for any shared resource.
- When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the method returns.

112) What is the purpose of Synchnorized block?

Synchronized block

Synchronized block can be used to perform synchronization on any specific resouce of the method.
Suppose you have 50 lines of code in your method, but you want to synchronize only 5 lines, you can use synchronized block.
If you put all the codes of the method in the synchronized block, it will work same as the synchronized method.

Points to remember for Synchronized block

- Synchronized block is used to lock an object for any shared resource.
- Scope of synchronized block is smaller than the method.

Syntax to use synchronized block

1. **synchronized** (object reference expression) {

```
2.     //code block
3.  }
```

Example of synchronized block

Let's see the simple example of synchronized block.

```
1.  <b><i>//Program of synchronized block</i></b>
2.
3.  class Table{
4.
5.   void printTable(int n){
6.     synchronized(this){//synchronized block
7.       for(int i=1;i<=5;i++){
8.        System.out.println(n*i);
9.        try{
10.        Thread.sleep(400);
11.       }catch(Exception e){System.out.println(e);}
12.      }
13.    }
14.  }//end of the method
15. }
16.
17.  class MyThread1 extends Thread{
18. Table t;
```

```
19. MyThread1(Table t){
20. this.t=t;
21. }
22. public void run(){
23. t.printTable(5);
24. }
25.
26. }
27. class MyThread2 extends Thread{
28. Table t;
29. MyThread2(Table t){
30. this.t=t;
31. }
32. public void run(){
33. t.printTable(100);
34. }
35. }
36.
37. class Use{
38. public static void main(String args[]){
39. Table obj = new Table();//only one object
40. MyThread1 t1=new MyThread1(obj);
```

```
41. MyThread2 t2=new MyThread2(obj);

42. t1.start();

43. t2.start();

44. }

45. }
```

Output:5

```
10
15
20
25
100
200
300
400
500
```

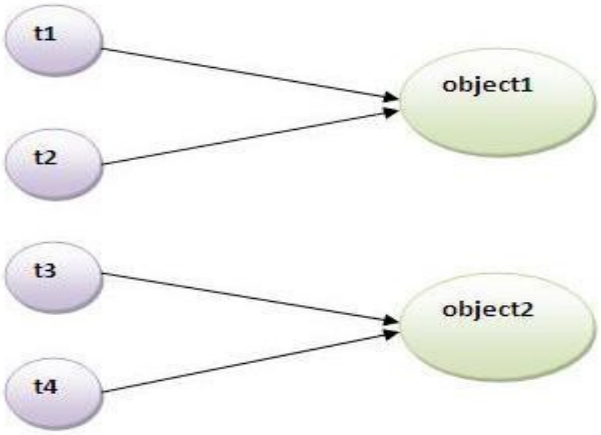
113)Can Java object be locked down for exclusive use by a given thread?

Yes. You can lock an object by putting it in a "synchronized" block. The locked object is inaccessible to any thread other than the one that explicitly claimed it.

114) What is static synchronization?

Static synchronization

If you make any static method as synchronized, the lock will be on the class not on object.



Problem without static synchronization

Suppose there are two objects of a shared class(e.g. Table) named object1 and object2.In case of synchonized method and synchronized block there cannot be inteference between t1 and t2 or t3 and t4 because t1 and t2 both refers to a common object that have a single lock.But there can be interference between t1 and t3 or t2 and t4 because t1 acquires another lock and t3 acquires another lock.I want no interference between t1 and t3 or t2 and t4.Static synchronization solves this problem.

Example of static synchronization

In this example we are applying synchronized keyword on the static method to perform statis synchrnization.

```
1. class Table{
```

```
2.   synchronized static void printTable(int n){
3.     for(int i=1;i<=10;i++){
4.       System.out.println(n*i);
5.       try{
6.         Thread.sleep(400);
7.       }catch(Exception e){}
8.     }
9.   }
10. }
11. class MyThread1 extends Thread{
12. public void run(){
13. Table.printTable(1);
14. }
15. }
16. class MyThread2 extends Thread{
17. public void run(){
18. Table.printTable(10);
19. }
20. }
21.
22. class MyThread3 extends Thread{
```

```
23. public void run(){
24. Table.printTable(100);
25. }
26. }
27.
28.
29.
30.
31. class MyThread4 extends Thread{
32. public void run(){
33. Table.printTable(1000);
34. }
35. }
36.
37. class Use{
38. public static void main(String t[]){
39. MyThread1 t1=new MyThread1();
40. MyThread2 t2=new MyThread2();
41. MyThread3 t3=new MyThread3();
42. MyThread4 t4=new MyThread4();
43. t1.start();
44. t2.start();
```

```
45. t3.start();

46. t4.start();

47. }

48. }
```

Output: 1
2
3
4
5
6
7
8
9
10
10
20
30
40
50
60
70
80
90
100
100
200
300
400
500
600
700
800
900
1000
1000

2000
3000
4000
5000
6000
7000
8000
9000
10000

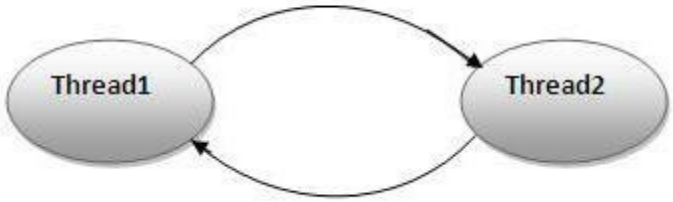
115)What is the difference between notify() and notifyAll()?

notify() is used to unblock one waiting thread whereas notifyAll() method is used to unblock all the threads in waiting state.

116)What is deadlock?

Deadlock:

Deadlock can occur in a situation when a thread is waiting for an object lock, that is acquired by another thread and second thread is waiting for an object lock that is acquired by first thread. Since, both threads are waiting for each other to release the lock, the condition is called daedlock.



Example of Deadlock in java:

```
1.  public class DeadlockExample {
2.    public static void main(String[] args) {
3.     final String resource1 = "ratan jaiswal";
4.     final String resource2 = "vimal jaiswal";
5.     // t1 tries to lock resource1 then resource2
6.     Thread t1 = new Thread() {
7.       public void run() {
8.           synchronized (resource1) {
9.            System.out.println("Thread 1: locked resource 1");
10.
11.           try { Thread.sleep(100);} catch (Exception e) {}
12.
13.           synchronized (resource2) {
14.            System.out.println("Thread 1: locked resource 2");
15.           }
16.         }
```

```
17.       }
18.     };
19.
20.     // t2 tries to lock resource2 then resource1
21.     Thread t2 = new Thread() {
22.       public void run() {
23.         synchronized (resource2) {
24.           System.out.println("Thread 2: locked resource 2");
25.
26.           try { Thread.sleep(100);} catch (Exception e) {}
27.
28.           synchronized (resource1) {
29.             System.out.println("Thread 2: locked resource 1");
30.           }
31.         }
32.       }
33.     };
34.
35.
36.     t1.start();
37.     t2.start();
```

```
38.    }
39. }
40.
```

Output: Thread 1: locked resource 1
Thread 2: locked resource 2

117) What is Garbage Collection?

Garbage Collection:

In java, garbage means unreferenced objects.

Garbage Collection is process of reclaiming the runtime unused memory automatically.

Advantage of Garbage Collection:

- It makes java memory efficient because garbage collector removes the unreferenced objects from heap memory.
- It is automatically done by the garbage collector so we don't need to make extra efforts.

How can an object be unreferenced?

There are many ways:

- By nulling the reference
- By assigning a reference to another
- By annonymous object etc.

1) By nulling a reference:

```
1. Employee e=new Employee();

2. e=null;
```

2) By assigning a reference to another:

```
1. Employee e1=new Employee();

2. Employee e2=new Employee();

3.

4. e1=e2;//now the first object refered by e1 is available for garbage collection
```

3) By annonymous object:

```
1. new Employee();
```

---

finalize() method:

The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing. This method is defined in System class as:

```
1. protected void finalize(){}
```

Note: The Garbage collector of JVM collects only those objects that are created by new keyword. So if you have created any object without new, you can use finalize method to perform cleanup processing (destroying remaining objects).

118) What is gc()?

gc() is a daemon thread.gc() method is defined in System class that is used to send request to JVM to perform garbage collection.

119) What is the purpose of finalize() method?

finalize() method is invoked just before the object is garbage collected.It is used to perform cleanup processing.

120) Can an unrefrenced objects be refrenced again?

Yes

.

121)What kind of thread is the Garbage collector thread?

Daemon
thread.

122)What is difference between final, finally and finalize?

final: final is a keyword, final can be variable, method or class.You, can't change the value of final variable, can't override final method, can't inherit final class.

finally: finally block is used in exception handling. finally block is always executed.

finalize():finalize() method is used in garbage collection.finalize() method is invoked just before the object is garbage collected.The finalize() method can be used to perform any cleanup processing.

---

123)What is the purpose of the Runtime class?

The purpose of the Runtime class is to provide access to the Java runtime system.

124)How will you invoke any external process in Java?

By Runtime.getRuntime().exec(?)
method.

125)What is the difference between the Reader/Writer class hierarchy and the InputStream/OutputStream class hierarchy?

The Reader/Writer class hierarchy is character-oriented, and the InputStream/OutputStream class hierarchy is byte-oriented.

126)What an I/O filter?

An I/O filter is an object that reads from one stream and writes to another, usually altering the data in some way as it is passed from one stream to another.

127) What is serialization?

Serialization

1. Serialization
2. Serializable Interface
3. ObjectOutputStream class
4. Example of Serialization
5. Deserialization
6. ObjectInputStream class
7. Example of Deserialization
8. Serialization with Inheritance
9. Externalizable interface
10. Serialization and static datamember
11. Serializing the array or collection objects
12. Serializing the object of a class that has non-serialzable object
13. Deserializing the class object that have parameterized constructor only

Serialization is a machanism of writing the state of an object into a byte stream. It is mainly used in Hibernate, JPA, EJB etc. The reverse operation of the serialization is called deserialization. The String class and all the wrapper classes implements Serializable interface bydefault.

Advantage of Serialization

It is mainly used to travel object's state on the network.

About Serializable interface

Serializable is a marker interface(have no body). It is just used to "mark" Java classes which support a certain capability. It must be implemented by the class whose object you want to persist. Let's see the example given below:

```
1. import java.io.Serializable;
```

```
2.
3.  public class Student implements Serializable{
4.   int id;
5.   String name;
6.   public Student(int id, String name) {
7.    this.id = id;
8.    this.name = name;
9.   }
10. }
```

ObjectOutputStream class:

An ObjectOutputStream is used to write primitive data types and Java objects to an OutputStream.Only objects that support the java.io.Serializable interface can be written to streams.

Commonly used Constructors:

1) public ObjectOutputStream(OutputStream out) throws IOException {}creates an ObjectOutputStream that writes to the specified OutputStream.

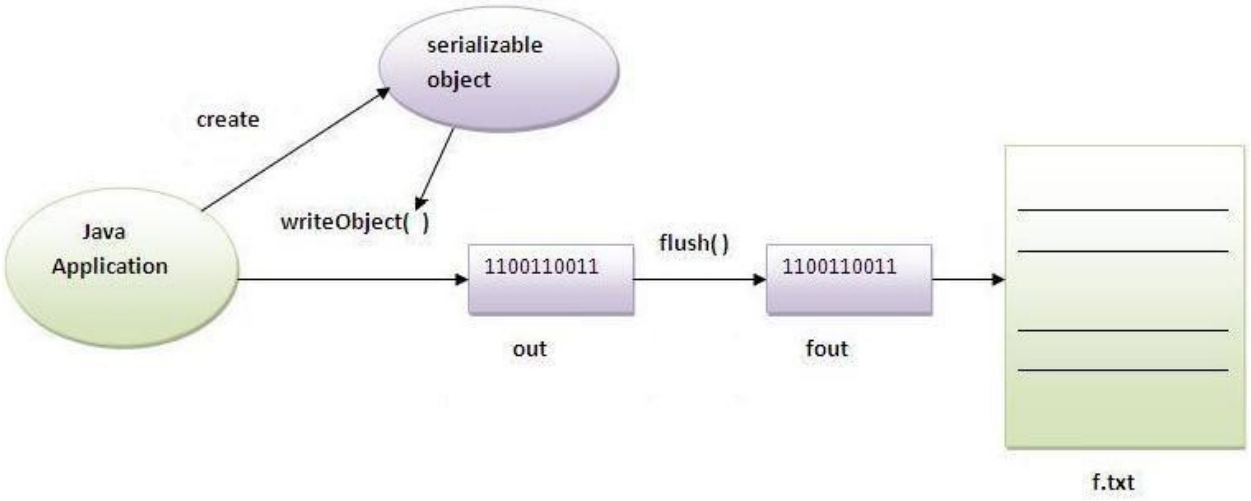
Commonly used Methods:

1) public final void writeObject(Object obj) throws IOException {}write the specified object to the ObjectOutputStream.

2) public void flush() throws IOException {}flushes the current output stream.

Example of Serialization

In this example, we are going to serialize the object of Student class. The writeObject() method of ObjectOutputStream class provides the functionality to serialize the object. We are saving the state of the object in the file named f.txt.




```
1. import java.io.*;
2. class Persist{
3.  public static void main(String args[])throws Exception{
4.   Student s1 =new Student(211,"ravi");
5.
6.   FileOutputStream fout=new FileOutputStream("f.txt");
7.   ObjectOutputStream out=new ObjectOutputStream(fout);
8.
9.   out.writeObject(s1);
```

```
10.   out.flush();
11.
12.   System.out.println("success");
13.  }
14. }
```

```
1.  <strong>Output:</strong>success
```

download this example of serialization

128) What is Deserialization?

Deserialization is the process of reconstructing the object from the serialized state.It is the reverse operation of serialization.

Deserilization:

Deserialization is the process of reconstructing the object from the serialized state.It is the reverse operation of serialization.

ObjectInputStream class:

An ObjectInputStream deserializes objects and primitive data written using an ObjectOutputStream.

Commonly used Constructors:

1) public ObjectInputStream(InputStream in) throws IOException {}creates an ObjectInputStream that reads from the specified InputStream.

Commonly used Methods:

1) public final Object readObject() throws IOException, ClassNotFoundException{}reads an object from the input stream.

Example of Deserialization:

```
1. import java.io.*;
```

```
2. class Depersist{
3.  public static void main(String args[])throws Exception{
4.
5.   ObjectInputStream in=new ObjectInputStream(new FileInputStream("f.txt"));
6.   Student s=(Student)in.readObject();
7.   System.out.println(s.id+" "+s.name);
8.
9.   in.close();
10. }
11.}
```

```
1. <strong>Output:</strong>211 ravi
```

129) What is transient keyword?

If you define any data member as transient,it will not be serialized.

130)What is Externalizable?

Externalizable interface is used to write the state of an object into a byte stream in compressed format.It is not a marker interface.

131)What is the difference between Serializalble and Externalizable interface?

Serializable is a marker interface but Externalizable is not a marker interface.When you use Serializable interface, your class is serialized automatically by default. But you can override writeObject() and readObject() two methods to control more complex object serailization process. When you use Externalizable interface, you have a complete control over your class's serialization process.

132)How do I convert a numeric IP address like 192.18.97.39 into a hostname like java.sun.com?

By InetAddress.getByName("192.18.97.39").getHostName() where 192.18.97.39 is the IP address.

133) What is reflection?

Reflection is the process of examining or modifying the runtime behaviour of a class at runtime.It is used in:

- IDE (Integreted Development Environment) e.g. Eclipse,MyEclipse,NetBeans.
- Debugger
- Test Tools etc.

134) Can you access the private method from outside the class?

Yes, by changing the runtime behaviour of a class if the class is not secured.

135) What is difference between ArrayList and Vector?

ArrayList	Vector
1) ArrayList is not synchronized.	1) Vector is synchronized.
2) ArrayList is not a legacy class.	2) Vector is a legacy class.

3) ArrayList increases its size by 50% of the array size.	**3) Vector increases its size by doubling the array size.**

136) What is difference between ArrayList and LinkedList?

ArrayList	**LinkedList**
1) ArrayList uses a dynamic array.	**1) LinkedList uses doubly linked list.**

2) ArrayList is not efficient for manipulation because a lot of shifting is required.	**2) LinkedList is efficient for manipulation.**

137) What is difference between HashMap and Hashtable?

HashMap	**Hashtable**
1) HashMap is not synchronized.	**1) Hashtable is synchronized.**

2) HashMap can contain one null key and multiple null values.	**2) Hashtable cannot contain any null key nor value.**

138)What is hash-collision in Hashtable and how it is handled in Java?

Two different keys with the same hash value. Two different entries will be kept in a single hash bucket to avoid the collision.

139) What is difference between HashSet and HashMap?

HashSet contains only values whereas HashMap contains entry(key,value).

140)What is difference between HashMap and TreeMap?

HashMap	TreeMap
1) HashMap is can contain one null key.	1) TreeMap connot contain any null key.
2) HashMap maintains no order.	2) TreeMap maintains ascending order.

141) What is difference between HashSet and TreeSet?

HashSet maintains no order whereas TreeSet maintains ascending order.

142) What is difference between List and Set?

List can contain duplicate elements whereas Set contains only unique elements.

143) What is difference between Iterator and ListIterator?

Iterator traverses the elements in forward direction only whereas ListIterator traverses the elements in forward and backward direction.

144) Can you make List,Set and Map elements synchronized?

Yes, Collections class provides methods to make List,Set or Map elements as synchronized:

public static List synchronizedList(List l){}

public static Set synchronizedSet(Set s){}

public static SortedSet synchronizedSortedSet(SortedSet s){}

public static Map synchronizedMap(Map m){}

public static SortedMap synchronizedSortedMap(SortedMap m){}

145) What is difference between Iterator and Enumeration?

Iterator	Enumeration

1) Iterator can traverse legacy and non-legacy elements.	**1) Enumeration can traverse only legacy elements.**
2) Iterator is fail-fast.	**2) Enumeration is not fail-fast.**
3) Iterator is slower than Enumeration.	**3) Enumeration is faster than Iterator.**

146) What is difference between Comparable and Comparator?

Comparable	Comparator
1) Comparable provides only one sort of sequence.	1) Comparator provides multiple sort of sequences.
2) It provides one method named compareTo().	2) It provides one method named compare().
3) It is found in java.lang package.	3) it is found in java.util package.

4) If we implement Comparable interface, actual class is modified.	**4) Actual class is not modified.**

147) What is the Dictionary class?

The Dictionary class provides the capability to store key-value pairs.

148)What are wrapper classes?

Wrapper classes are classes that allow primitive types to be accessed as objects.

149)What is a native method?

A native method is a method that is implemented in a language other than Java.

150)What is the purpose of the System class?

The purpose of the System class is to provide access to system resources.

151)What comes to mind when someone mentions a shallow copy in Java?

Object
cloning.

152)What is singleton class?

Singleton class means that any given time only one instance of the class is present, in one JVM.

153)Which containers use a border layout as their default layout?

The Window, Frame and Dialog classes use a border layout as their default layout.

154)Which containers use a FlowLayout as their default layout?

The Panel and Applet classes use the FlowLayout as their default layout.

155)What are peerless components?

The peerless components are called light weight components.

156)is the difference between a Scrollbar and a ScrollPane?

A Scrollbar is a Component, but not a Container. A ScrollPane is a Container. A ScrollPane handles its own events and performs its own scrolling.

157)What is a lightweight component?

Lightweight components are the one which doesn?t go with the native call to obtain the graphical units. They share their parent component graphical units to render them. For example, Swing components.

158)What is a heavyweight component?

For every paint call, there will be a native call to get the graphical units.For Example, AWT.

159)What is an applet?

An applet is a small java program that runs inside the browser and generates dynamic contents.

160)Can you write a Java class that could be used both as an applet as well as an application?

. Yes. Add a main() method to the
applet.

161)What is Locale?

A Locale object represents a specific geographical, political, or cultural region.

162)How will you load a specific locale?

By ResourceBundle.getBundle(?)
method.

163)What is a JavaBean?

are reusable software components written in the Java programming language, designed to be manipulated visually by a software develpoment environment, like JBuilder or VisualAge for Java.

164)Can RMI and Corba based applications interact?

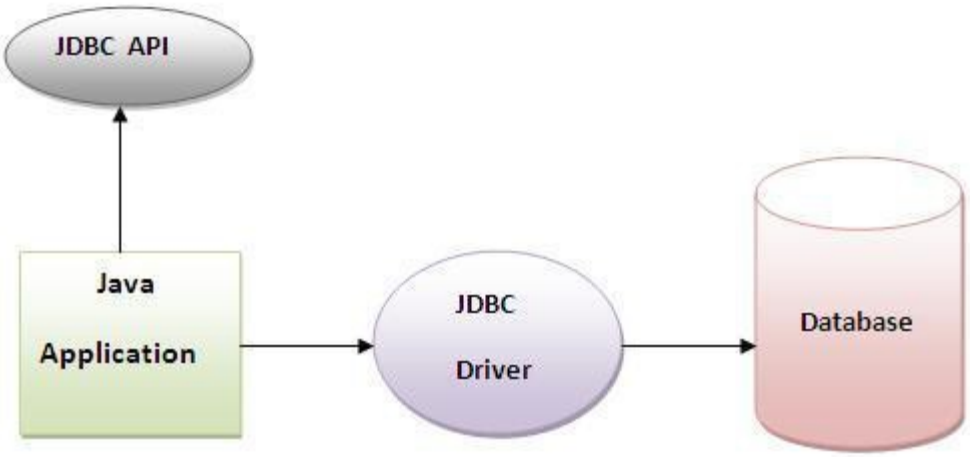
Yes they can. RMI is available with IIOP as the transport protocol instead of JRMP.

165)What is JDBC?

JDBC is a Java API that is used to connect and execute query to the database. JDBC API uses jdbc drivers to connects to the database

JDBC Tutorial

This JDBC tutorial covers all the topics of JDBC with the simple examples. JDBC is a Java API that is used to connect and execute query to the database. JDBC API uses jdbc drivers to connects to the database.



Why use JDBC?

Before JDBC, ODBC API was used to connect and execute query to the database. But ODBC API uses ODBC driver that is written in C language which is plateform dependent and unsecured. That is why Sun Microsystem has defined its own API (JDBC API) that uses JDBC driver written in Java language.

Do You Know ?

- ***How to connect Java application with Oracle and Mysql database using JDBC?***
- ***What is the difference between Statement and PreparedStatement interface?***
- ***How to print total numbers of tables and views of a database using JDBC ?***
- ***How to store and retrieve images from Oracle database using JDBC?***
- ***How to store and retrieve files from Oracle database using JDBC?***

What is API?

API (Application programming interface) is a document that contains description of all the features of a product or software. It represents classes and interfaces that software programs can follow to communicate with each other. An API can be created for applications, libraries, operating systems, etc

167)What are the steps to connect to database in java?

5 Steps to connect to the database in java

1. 5 Steps to connect to the database in java
    1. Register the driver class
    2. Create the connection object
    3. Create the Statement object
    4. Execute the query
    5. Close the connection object

There are 5 steps to connect any java application with the database in java using JDBC. They are as follows:

- Register the driver class
- Creating connection
- Creating statement
- Executing queries
- Closing connection

---

1) Register the driver class

The forName() method of Class class is used to register the driver class. This method is used to dynamically load the driver class.

Syntax of forName() method

```
1. public static void forName(String className)throws ClassNotFoundException
```

Example to register the OracleDriver class

```
1. Class.forName("oracle.jdbc.driver.OracleDriver");
```

---

2) Create the connection object

The getConnection() method of DriverManager class is used to establish connection with the database.

Syntax of getConnection() method

```
1. 1) public static Connection getConnection(String url)throws SQLException

2. 2) public static Connection getConnection(String url,String name,String password)
```

3. **throws** SQLException

Example to establish connection with the Oracle database

```
1. Connection con=DriverManager.getConnection(
2. "jdbc:oracle:thin:@localhost:1521:xe","system","password");
```

---

3) Create the Statement object

The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.

Syntax of createStatement() method

```
1. public Statement createStatement()throws SQLException
```

Example to create the statement object

```
1. Statement stmt=con.createStatement();
```

---

4) Execute the query

The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the rocords of a table.

Syntax of executeQuery() method

```
1. public ResultSet executeQuery(String sql)throws SQLException
```

Example to execute query

```
1. ResultSet rs=stmt.executeQuery("select * from emp");
2.
3. while(rs.next()){
4. System.out.println(rs.getInt(1)+" "+rs.getString(2));
5. }
```

---

5) Close the connection object

By closing connection object statment and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.

Syntax of close() method

```
1. public void close()throws SQLException
```

Example to close connection

```
1. con.close();
```

168)What is the difference between Statement and PreparedStatement interface?

PreparedStatement:

The PreparedStatement interface is a subinterface of Statement. It is used to exeucte parameterized query.

Why use PreparedStatement?

The performance of the application will be faster if you use PreparedStatement interface because query is compiled only once.

The prepareStatement() method of Connection interface is used to return the object of PreparedStatement.Syntax:

```
1. public PreparedStatement prepareStatement(String query)throws SQLException{}
```

```
1. create table emp(id number(10),name varchar2(50));
```

169)How can we execute stored procedures and functions?

By using Callable statement interface, we can execute procedures and funtions.

170)How can we store and retrieve images from the database?

Example to store image in Oracle database:

The setBinaryStream() method of PreparedStatement is used to set Binary information into the parameterIndex.

Syntax:

```
1) public void setBinaryStream(int paramIndex,InputStream stream)
throws SQLException
2) public void setBinaryStream(int paramIndex,InputStream stream,long length)
throws SQLException
```

For storing image into the database, BLOB (Binary Large Object) datatype is used in the table. For example:

```
CREATE TABLE  "IMGTABLE"
   (    "NAME" VARCHAR2(4000),
    "PHOTO" BLOB
   )
```

5. /

```
1. import java.sql.*;
2. import java.io.*;
3. public class InsertImage {
4. public static void main(String[] args) {
5. try{
6. Class.forName("oracle.jdbc.driver.OracleDriver");
7. Connection con=DriverManager.getConnection(
8. "jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
9.
10. PreparedStatement ps=con.prepareStatement("insert into imgtable values(?,?)");
11.
12. FileInputStream fin=new FileInputStream("d:\\g.jpg");
13.
14. ps.setString(1,"sonoo");
15. ps.setBinaryStream(2,fin,fin.available());
16. int i=ps.executeUpdate();
17. System.out.println(i+" records affected");
18.
19. con.close();
20.
```

```
21. }catch (Exception e) {e.printStackTrace();}
22. }
23. }
```