<https://www.softwaretestinghelp.com/core-java-interview-questions/>

<https://www.javatpoint.com/corejava-interview-questions>

Object: An Object can be defined as an instance of a class.

Class: Collection of objects is called class. It is a logical entity.

Inheritance**:** It is used to achieve runtime polymorphism.

Polymorphism: If *one task is performed by different ways*. example: to convince the customer differently.

In Java, we use method overloading and method overriding to achieve polymorphism.

Another example can be to speak something; for example, a cat speaks meow, dog barks woof, etc.

#### Abstraction

Hiding internal details and showing functionality is known as abstraction. For example phone call, we don't know the internal processing.

In Java, we use abstract class and interface to achieve abstraction.

#### Advantage of Method

* Code Reusability
* Code Optimization

### new keyword in Java

The new keyword is used to allocate memory at runtime. All objects get memory in Heap memory area.

## 3 Ways to initialize object

There are 3 ways to initialize object in java.

1. By reference variable
2. By method
3. By constructor

## Different ways to create an object in Java?

* By new keyword
* By newInstance() method
* By clone() method
* By deserialization
* By factory method etc.

# Constructors

It is a special type of method which is used to initialize the object.

It is called constructor because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

### Rules for creating Java constructor

1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type
3. A Java constructor cannot be abstract, static, final, and synchronized

## Types of Java constructors

1. Default constructor (no-arg constructor)
2. Parameterized constructor

## Constructor Overloading in Java

In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods.

**Static keyword** in Java is used for memory management mainly

The static can be:

1. Variable (also known as a class variable)
2. Method (also known as a class method)
3. Block
4. Nested class

The static variable gets memory only once in the class area at the time of class loading.

### Advantages of static variable

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

## Java 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 the object of a class.
* A static method can be invoked without the need for creating an instance of a class.
* A static method can access static data member and can change the value of it.

## Usage of java this keyword

this is a **reference variable** that refers to the current object

Here is given the 6 usage of java this keyword.

1. this can be used to refer current class instance variable.
2. this can be used to invoke current class method (implicitly)
3. this() can be used to invoke current class constructor.
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 can be used to return the current class instance from the method.

### 1) this: to refer current class instance variable

### 2) this: to invoke current class method

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

### 3) this() : to invoke current class constructor

class A{

A(){System.out.println("hello a");}

A(int x){

this();

System.out.println(x);

}

}

class TestThis5{

public static void main(String args[]){

A a=new A(10);

}}

### 4) this: to pass as an argument in the method

### 5) this: to pass as argument in the constructor call

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

1. return\_type method\_name(){
2. return this;
3. }

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

class A{

A getA(){

return this;  }

void msg(){System.out.println("Hello java");} }

class Test1{

public static void main(String args[]){

new A().getA().msg();  }

}

### 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 A5{
2. void m(){
3. System.out.println(this);//prints same reference ID
4. }
5. public static void main(String args[]){
6. A5 obj=new A5();
7. System.out.println(obj);//prints the reference ID
8. obj.m();
9. }
10. }

Output:

A5@22b3ea59

A5@22b3ea59

**Why use inheritance in java**

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

## Types of inheritance in java

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

In java programming, multiple and hybrid inheritance is supported through interface only. We will learn about interfaces later.

## Method Overloading in Java

If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.

There are two ways to overload the method in java

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

## 1) Method Overloading: changing no. of arguments

## 2) Method Overloading: changing data type of arguments

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

We can have any number of main methods in a class by method overloading. But JVM calls main() method which receives string array as arguments only.

## Method Overriding in Java

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**.

Usage of Java Method Overriding

* Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
* Method overriding is used for runtime polymorphism

### Can we override static method?

No, a static method cannot be overridden. It can be proved by runtime polymorphism, so we will learn it later.

### Why can we not override static method?

It is because the static method is bound with class whereas instance method is bound with an object. Static belongs to the class area, and an instance belongs to the heap area.

# Super Keyword in Java

The **super** keyword in Java is a reference variable which is used to refer immediate parent class object.

# Instance initializer block

Suppose I have to perform some operations while assigning value to instance data member e.g. a for loop to fill a complex array or error handling etc.

**class** Bike7{

**int** speed;

    Bike7(){System.out.println("speed is "+speed);}

    {speed=100;}

**public** **static** **void** main(String args[]){

    Bike7 b1=**new** Bike7();

    Bike7 b2=**new** Bike7();    } }

There are three places in java where you can perform operations:

1. method
2. constructor
3. block

## What is invoked first, instance initializer block or constructor?

**class** Bike8{

**int** speed;

    Bike8(){System.out.println("constructor is invoked");}

    {System.out.println("instance initializer block invoked");}

**public** **static** **void** main(String args[]){

    Bike8 b1=**new** Bike8();

    Bike8 b2=**new** Bike8();

    }

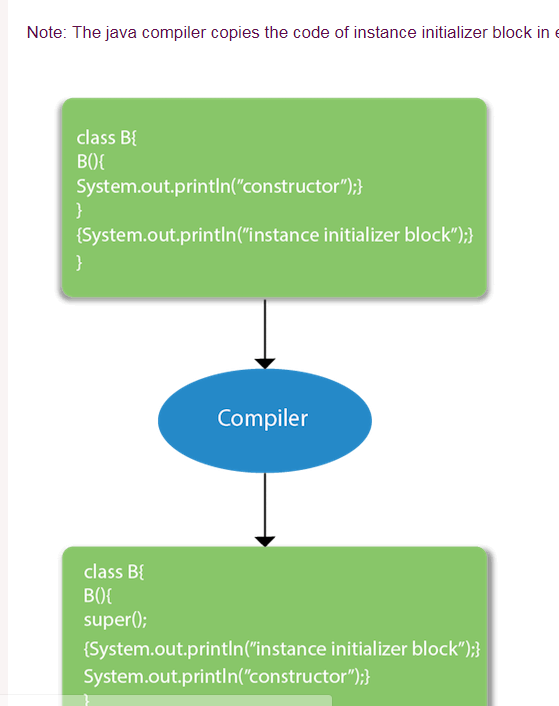
}

Output:instance initializer block invoked

constructor is invoked

instance initializer block invoked

constructor is invoked



|  |
| --- |
| There are mainly three rules for the instance initializer block. They are as follows: |

1. The instance initializer block is created when instance of the class is created.
2. The instance initializer block is invoked after the parent class constructor is invoked (i.e. after super() constructor call).
3. The instance initializer block comes in the order in which they appear.

**class** A{

A(){

System.out.println("parent class constructor invoked");  }  }

**class** B2 **extends** A{

B2(){

**super**();

System.out.println("child class constructor invoked");  }

{System.out.println("instance initializer block is invoked");}

**public** **static** **void** main(String args[]){

B2 b=**new** B2();  } }

Output:parent class constructor invoked

instance initializer block is invoked

child class constructor invoked

# Final Keyword In Java

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

1. variable
2. method
3. class

## 1) Java final variable

## 2) Java final method

## 3) Java final class

### Is final method inherited?

### blank final variable

**class** Student{

**int** id;

String name;

**final** String PAN\_CARD\_NUMBER;

}

### static blank final variable

A static final variable that is not initialized at the time of declaration is known as static blank final variable. It can be initialized only in static block.

**class** A{

**static** **final** **int** data;//static blank final variable

**static**{ data=50;}

**public** **static** **void** main(String args[]){

    System.out.println(A.data);   }  }

### What is final parameter?

**class** Bike11{

**int** cube(**final** **int** n){

   n=n+2;//can't be changed as n is final

   n\*n\*n;

  }

**public** **static** **void** main(String args[]){

    Bike11 b=**new** Bike11();

    b.cube(5);   }  }

### Can we declare a constructor final?

No, because constructor is never inherited.

# Polymorphism in Java

**Polymorphism in Java** is a concept by which we can perform a single action in different ways.There are two types of polymorphism in Java: compile-time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading and method overriding. If you overload a static method in Java, it is the example of compile time polymorphism. Here, we will focus on runtime polymorphism in java.

## Java Runtime Polymorphism with Data Member

A method is overridden, not the data members, so runtime polymorphism can't be achieved by data members.

#### Runtime polymorphism can't be achieved by data members.

**class** Bike{

**int** speedlimit=90;  }

**class** Honda3 **extends** Bike{

**int** speedlimit=150;

**public** **static** **void** main(String args[]){

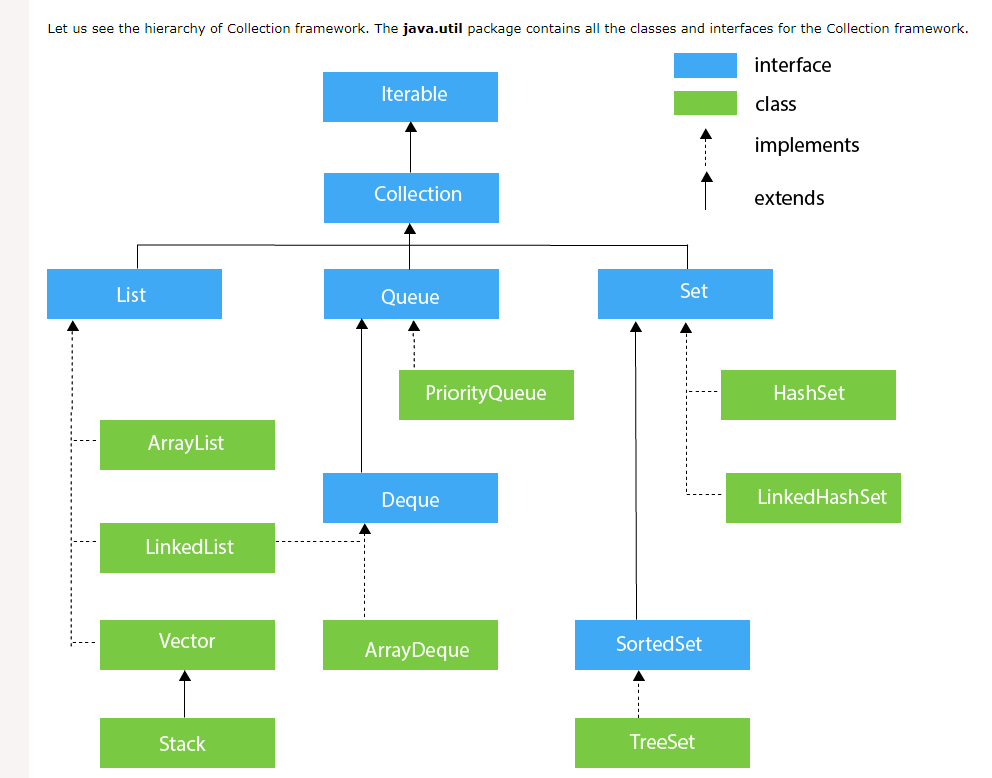
 Bike obj=**new** Honda3();

 System.out.println(obj.speedlimit);//90  }

Selenium url”

<https://www.toolsqa.com/selenium-webdriver/headless-browser-testing-selenium-webdriver/>

**COLLECTIONS:**



The **Collection in Java** is a framework that provides an architecture to store and manipulate the group of objects.

Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.

Java Collection means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).

### Iterator interface

### Iterator interface provides the facility of iterating the elements in a forward direction only.

## Iterable Interface

### The Iterable interface is the root interface for all the collection classes. The Collection interface extends the Iterable interface and therefore all the subclasses of Collection interface also implement the Iterable interface.

## Collection Interface

The Collection interface is the interface which is implemented by all the classes in the collection framework. It declares the methods that every collection will have. In other words, we can say that the Collection interface builds the foundation on which the collection framework depends.

### Some of the methods of Collection interface are Boolean add ( Object obj), Boolean addAll ( Collection c), void clear(), etc. which are implemented by all the subclasses of Collection interface.

## List Interface

List interface is the child interface of Collection interface. It inhibits a list type data structure in which we can store the ordered collection of objects. It can have **duplicate** values.

List interface is implemented by the classes ArrayList, LinkedList, Vector, and Stack.

## ArrayList

The ArrayList class implements the List interface. It uses a dynamic array to store the duplicate element of different data types. The ArrayList class maintains the insertion order and is **non-synchronized**. The elements stored in the ArrayList class can be randomly accessed.

## LinkedList

LinkedList implements the Collection interface. It uses a doubly linked list internally to store the elements. It can store the duplicate elements. It maintains the insertion order and is **not synchronized**. In LinkedList, the manipulation is fast because no shifting is required.

## Vector

Vector uses a dynamic array to store the data elements. It is similar to ArrayList. However, It is synchronized and contains many methods that are not the part of Collection framework.

## Stack

The stack is the subclass of Vector. It implements the last-in-first-out data structure, i.e., Stack. The stack contains all of the methods of Vector class and also provides its methods like boolean push(), boolean peek(), boolean push(object o), which defines its properties.

## Queue Interface

Queue interface maintains the first-in-first-out order. It can be defined as an ordered list that is used to hold the elements which are about to be processed. There are various classes like PriorityQueue, Deque, and ArrayDeque which implements the Queue interface.

## PriorityQueue

The PriorityQueue class implements the Queue interface. It holds the elements or objects which are to be processed by their priorities. PriorityQueue doesn't allow null values to be stored in the queue.

## Deque Interface

Deque interface extends the Queue interface. In Deque, we can remove and add the elements from both the side. Deque stands for a double-ended queue which enables us to perform the operations at both the ends.

## ArrayDeque

ArrayDeque class implements the Deque interface. It facilitates us to use the Deque. Unlike queue, we can add or delete the elements from both the ends.

ArrayDeque is faster than ArrayList and Stack and has no capacity restrictions.

## Set Interface

Set Interface in Java is present in java.util package. It extends the Collection interface. It represents the unordered set of elements which doesn't allow us to store the duplicate items. We can store at most one null value in Set. Set is implemented by HashSet, LinkedHashSet, and TreeSet.

## HashSet

HashSet class implements Set Interface. It represents the collection that uses a hash table for storage. Hashing is used to store the elements in the HashSet. It contains unique items.

## LinkedHashSet

LinkedHashSet class represents the LinkedList implementation of Set Interface. It extends the HashSet class and implements Set interface. Like HashSet, It also contains unique elements. It maintains the insertion order and permits null elements.

## SortedSet Interface

SortedSet is the alternate of Set interface that provides a total ordering on its elements. The elements of the SortedSet are arranged in the increasing (ascending) order. The SortedSet provides the additional methods that inhibit the natural ordering of the elements.

## TreeSet

Java TreeSet class implements the Set interface that uses a tree for storage. Like HashSet, TreeSet also contains unique elements. However, the access and retrieval time of TreeSet is quite fast. The elements in TreeSet stored in ascending order.

|  |
| --- |
|  |

Testbitbucket

Testtest/ login@123

test\_userbitbucket

Salary slip in google account and print

Scooty Insurance