**What is Java used for?** Java has been used in different domains. Some of them are listed below:

**Banking**: To deal with transaction management.

**Retail**: Billing applications that you see in a store/restaurant are completely written in Java.

**Information Technology**: Java is designed to solve implementation dependencies.

**Android**: Applications are either written in Java or use Java API.

**Financial services**: It is used in server-side applications.

**Stock market**: To write algorithms as to which company they should invest in.

**Big Data**: Hadoop Map Reduce framework is written using Java.

**Scientific and Research Community**: To deal with huge amount of data.

**Member Variables: -** A member variable plays a major role in a class as it is used to store a data value. When we define a class, we can declare a member variable. These variables are members of a class.  
Member variables are further classified into three types:

Local variable

Instance variable

Class/Static variable

**Local variable**: These are the variables which are declared within the method of a class. Let’s understand this with a programmatic example:

|  |
| --- |
| public class Car {        public void display(int m){  // Method             int model=m;                 // Created a local variable model             System.out.println("Model of the car is" +model);       } |

**Instance variable**: Instance variable is declared in a class but outside a method, constructor or any block.

|  |
| --- |
| public class Car {        public String color;     // Created an instance variable color  Car(String c)   {      color=c;     }  public void display() {  // Method        System.out.println("color of the car is"+color);     }  public static void main(String args[]){                Car obj=new Car("black");                  obj.display();         }  } |

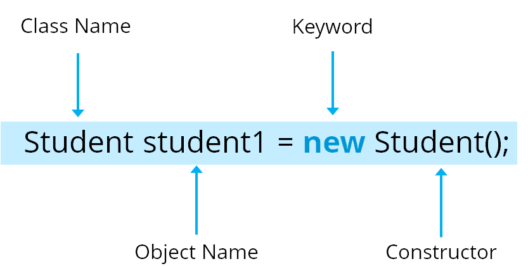
**Class variable:** Class variables are also called as static variables. These variables have only one copy that is shared by all the different objects in a class.

|  |
| --- |
| public class Car {        public static int tyres;   // Created a class variable tyres          public static void main(String args[]){             tyres=4;             System.out.println("Number of tyres are"+tyres);          }  } |

**Classes and Objects: -** A **class** in Java is a blueprint which includes all your data.  A class contain fields (variables) and methods to describe the behavior of an object. **Syntax of a class.**

|  |
| --- |
| class Abc {        member variables // class body          methods   } |

**How can you access these member variables and methods?** Here comes the concept of **Object**. An **object** is a major element in a class which has a state and behavior. It is an instance of a class which can access your data. Syntax to create an object in Java:



**How can you call a method using an object in Java?**

|  |
| --- |
| class Student()        void display(); {            // Method       ------                           // logic of method  }  public static void main(String args[]){        Student obj=new Student();   // Created an object        obj.display();               // Method called  } |

**Object Oriented Programming:**

**Inheritance: -** Inheritance is one such concept where the properties of one class can be inherited by the other. It helps to reuse the code and establish a relationship between different classes. **As we can say child inherits the properties from his father.** Similarly, in **Java, there are two classes:**

**1. Parent class (Super or Base class):-** A class whose properties are inherited is known as Parent class.

**2. Child class (Subclass or Derived class):-** A class which inherits the properties is known as Child Class

**Inheritance is further classified into 4 types:**

**Single Inheritance:** In single inheritance, one class inherits the properties of another. It enables a derived class to inherit the properties and behavior from a single parent class. This will in turn enable code reusability as well as add new features to the existing code. Here, Class A is your parent class and Class B is your child class which inherits the properties and behavior of the parent class. **syntax for single inheritance:**

|  |
| --- |
| Class A {  ---  }  Class B extends A {  ---  } |

**Multilevel Inheritance:** When a class is derived from a class which is also derived from another class, i.e. a class having more than one parent class but at different levels, such type of inheritance is called Multilevel Inheritance. **syntax for multilevel inheritance in Java:**

|  |
| --- |
| Class A{  ---  }  Class B extends A{  ---  }  Class C extends B{  ---  } |

**Hierarchical Inheritance:** When a class has more than one child classes (sub classes) or in other words, more than one child classes have the same parent class, then such kind of inheritance is known as **hierarchical**. **Syntax for hierarchical inheritance in Java:**

|  |
| --- |
| Class A{  ---  }  Class B extends A{  ---  }  Class C extends A{  ---  } |

**Hybrid Inheritance:** Hybridinheritance is a combination of multiple inheritance and multilevel inheritance. Since multiple inheritance is not supported in Java as it leads to ambiguity, so this type of inheritance can only be achieved through the use of the interfaces. **Class A is a parent class for class B and C, whereas Class B and C are the parent class of D which is the only child class of B and C.**

**Encapsulation:-**Encapsulation is a mechanism where you bind your data and code together as a single unit. It also means to hide your data in order to make it safe from any modification. **Example of a medical capsule**, where the drug is always safe inside the capsule. Similarly, through encapsulation the methods and variables of a class are well hidden and safe.

**We can achieve encapsulation in Java by:**

Declaring the variables of a class as private.

Providing public setter and getter methods to modify and view the variables values.

|  |
| --- |
| public class Employee {  private String name;  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public static void main(String[] args) {  }  } |

**Abstraction:** Abstraction refers to the quality of dealing with ideas rather than events. It basically deals with hiding the details and showing the essential things to the user. Therefore, abstraction helps to reduce complexity. You can achieve abstraction in two ways:

**a) Abstract Class**

**b) Interface**

**Abstract class:** Abstract class in Java contains the ‘abstract’ keyword. Abstract keyword mean? If a class is declared abstract, it cannot be instantiated, which means you cannot create an object of an abstract class. Also, an abstract class can contain abstract as well as concrete methods.You can achieve 0-100% abstraction using abstract class. To use an abstract class, you have to inherit it from another class where you have to provide implementations for the abstract methods there itself, else it will also become an abstract class.

**Syntax of an abstract class:**

|  |  |
| --- | --- |
| 1  2 | Abstract class Mobile {   // abstract class mobile  Abstract void run();      // abstract method |

**Interface:**Interface in Java is a blueprint of a class or you can say it is a collection of abstract methods and static constants. In an interface, each method is public and abstract but it does not contain any constructor. Along with abstraction, interface also helps to achieve multiple inheritance in Java.You can achieve 100% abstraction using interfaces.

|  |
| --- |
| public interface ParentCar {  public void changeGear( int newValue);  public void speedUp(int increment);  public void applyBrakes(int decrement);  }  public class Audi implements ParentCar {  int speed=0;  int gear=1;  public void changeGear( int value){  gear=value;  }  public void speedUp( int increment) {  speed=speed+increment;  }  public void applyBrakes(int decrement) {  speed=speed-decrement;  }  void printStates(){  System.out.println("speed:"+speed+"gear:"+gear);  }  public static void main(String[] args) {  Audi A6= new Audi();  A6.speedUp(50);  A6.printStates();  A6.changeGear(4);  A6.SpeedUp(100);  A6.printStates();  }  } |

**Polymorphism: -** Polymorphism means taking many forms, where ‘poly’ means many and ‘morph’ means forms. It is the ability of a variable, function or object to take on multiple forms. In other words, polymorphism allows you define one interface or method and have multiple implementations.

**Polymorphism in Java is of two types:**

Run time polymorphism

Compile time polymorphism

**Run time polymorphism:**In Java, runtime polymorphism refers to a process in which a call to an overridden method is resolved at runtime rather than at compile-time. In this, a reference variable is used to call an overridden method of a superclass at run time. Method overriding is an example of run time polymorphism. Let us look the following code to understand how the method overriding works:

|  |
| --- |
| public Class BowlerClass{  void bowlingMethod(){  System.out.println(" bowler ");  }  }  public Class FastPacer{  void bowlingMethod(){  System.out.println(" fast bowler ");  }  Public static void main(String[] args){  FastPacer obj= new FastPacer();  obj.bowlingMethod();  }  } |

**Compile time polymorphism:** In Java, compile time polymorphism refers to a process in which a call to an overloaded method is resolved at compile time rather than at run time. Method overloading is an example of compile time polymorphism. Method Overloading is a feature that allows a class to have two or more methods having the same name but the arguments passed to the methods are different. Unlike method overriding, arguments can differ in:

**Number of parameters passed to a method:**

Datatype of parameters

Sequence of datatypes when passed to a method.

Let us look at the following code to understand how the method overloading works:

|  |
| --- |
| class Adder {  Static int add(int a, int b){  return a+b;  }  static double add( double a, double b){  return a+b;  }  public static void main(String args[]){  System.out.println(Adder.add(11,11));  System.out.println(Adder.add(12.3,12.6));  }  } |

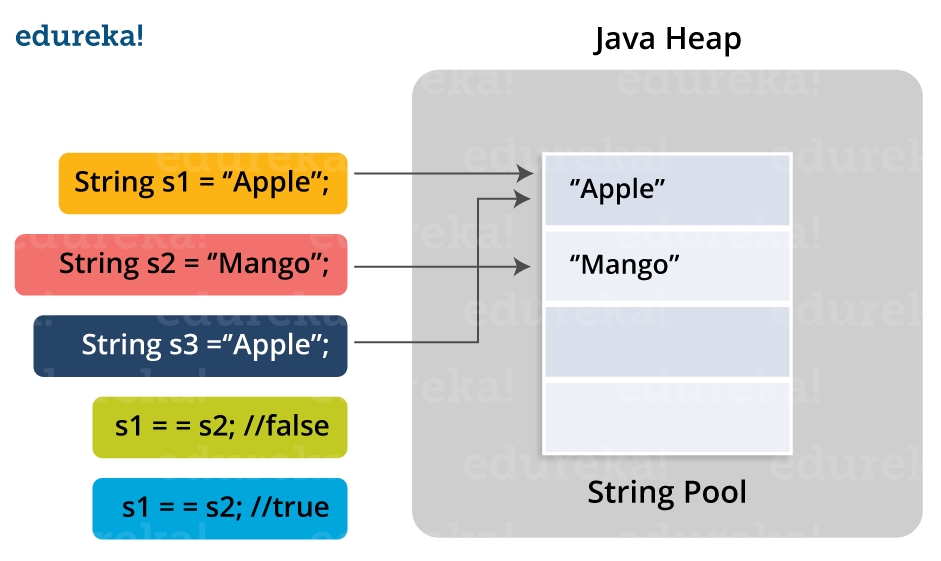
**Java String**: - String is a sequence of characters. But in Java, a string is an object that represents a sequence of characters. The java.lang.String class is used to create string object.

**There are two ways to create a String object:**

**By string literal**: Java String literal is created by using double quotes. For Example: **String s=“Welcome**”;

**By new keyword**: Java String is created by using a keyword “new”.   
**For example: String s=new String (“Welcome”);** It creates two objects (in String pool and in heap) and one reference variable where the variable‘s’ will refer to the object in the heap.

**Java String Pool:**Java String pool refers to collection of Strings which are stored in heap memory. In this, whenever a new object is created, String pool first checks whether the object is already present in the pool or not. If it is present, then same reference is returned to the variable else new object will be created in the String pool and the respective reference will be returned.



**Strings are immutable**. By immutable, we mean that Strings are constant, their values cannot be changed after they are created. Because String objects are immutable, they can be shared. For example:

**String str =”abc”;** **is equivalent to:** **Char data [] = {‘a’, ‘b’, ‘c’};    String str = new String (data);**

**Java String Methods:-**

**Java String length ()**: The Java String length () method tells the length of the string. It returns count of total number of characters present in the String. For example:

|  |
| --- |
| public class Example{  public static void main(String args[]{  String s1="hello";  String s2="whatsup";  System.out.println("string length is: "+s1.length());  //5  System.out.println("string length is: "+s2.length()); //7  }  } |

**Java String compareTo ()**: The Java String compareTo () method compares the given string with current string. It is a method of ‘Comparable’ interface which is implemented by String class. Don’t worry, we will be learning about String interfaces later. It either returns positive number, negative number or 0. For example:

|  |
| --- |
| public class CompareToExample{  public static void main(String args[]){  String s1="hello";  String s2="hello";  String s3="hemlo";  String s4="flag";  System.out.println(s1.compareTo(s2)); // 0 because both are equal  System.out.println(s1.compareTo(s3)); //-1 because "l" is only one time lower than "m"  System.out.println(s1.compareTo(s4)); // 2 because "h" is 2 times greater than "f"  }  }  **This program shows the comparison between the various strings. It is noticed that   if s1 > s2, it returns a positive number   if s1 < s2, it returns a negative number  if s1 == s2, it returns 0** |

**Java String concat ():**The Java String concat () method combines a specific string at the end of another string and ultimately returns a combined string. It is like appending another string. For example:

|  |
| --- |
| public class ConcatExample{  public static void main(String args[]){  String s1="hello";  s1=s1.concat("how are you");  System.out.println(s1); // hellohow are you  }  } |

**Java String IsEmpty ()**: This method checks whether the String contains anything or not. If the java String is Empty, it returns true else false. For example:

|  |
| --- |
| public class IsEmptyExample{  public static void main(String args[]){  String s1="";  String s2="hello";  System.out.println(s1.isEmpty());      // true  System.out.println(s2.isEmpty());      // false  }  } |

**Java String Trim ()**: The java string trim () method removes the leading and trailing spaces. It checks the Unicode value of space character (‘\u0020’) before and after the string. If it exists, then removes the spaces and return the omitted string. For example:

|  |
| --- |
| public class StringTrimExample{  public static void main(String args[]){  String s1="  hello   ";  System.out.println(s1+"how are you");        // without trim()  System.out.println(s1.trim()+"how are you"); // with trim()  }  } |

**Java String toLowerCase ()**: The java string toLowerCase () method converts all the characters of the String to lower case. For example:

|  |
| --- |
| public class StringLowerExample{  public static void main(String args[]){  String s1="HELLO HOW Are You?”;  String s1lower=s1.toLowerCase();  System.out.println(s1lower); // hello how are you  }  } |

**Java String toUpper ()**: The Java String toUpperCase () method converts all the characters of the String to upper case. For example:

|  |
| --- |
| public class StringUpperExample{  public static void main(String args[]){  String s1="hello how are you";  String s1upper=s1.toUpperCase();  System.out.println(s1upper);  // HELLO HOW ARE YOU  }  } |

**Java String ValueOf ()**: This method converts different types of values into string. Using this method, you can convert int to string, long to string, Boolean to string, character to string, float to string, double to string, object to string and char array to string. The signature or syntax of string valueOf () method is given below:  
**1. public static String valueOf(boolean b)  
2. public static String valueOf(char c)  
3. public static String valueOf(char[] c)  
4. public static String valueOf(int i)  
5. public static String valueOf(long l)  
6. public static String valueOf(float f)  
7. public static String valueOf(double d)  
8. public static String valueOf(Object o)**

|  |
| --- |
| public class StringValueOfExample{  public static void main(String args[]){  int value=20;  String s1=String.valueOf(value);  System.out.println (s1+17);       //concatenating string with 10 2017.  }  } |

**Java String replace ()**: The Java String replace () method returns a string, replacing all the old characters or CharSequence to new characters. There are 2 ways to replace methods in a Java String.

|  |
| --- |
| public class ReplaceExample1{  public static void main(String args[]){  String s1="hello how are you";  String replaceString=s1.replace('h','t');  System.out.println(replaceString);  }  } |

**Java String replace (CharSequence target, CharSequence replacement) method**:

|  |
| --- |
| public class ReplaceExample2{  public static void main(String args[]){  String s1="Hey, welcome to Edureka";  String replaceString=s1.replace("Edureka","Brainforce");  System.out.println(replaceString);  }  } |

**Java String contains ()**: The java string contains () method searches the sequence of characters in the string. If the sequences of characters are found, then it returns true otherwise returns false. For example:

|  |
| --- |
| class ContainsExample{  public static void main(String args[]){  String name=" hello how are you doing?";  System.out.println(name.contains("how are you"));  // returns true  System.out.println(name.contains("hello"));        // returns true  System.out.println(name.contains("fine"));         // returns false  }  } |

**Java String equals ()**: The Java String equals () method compares the two given strings on the basis of content of the string i.e. Java String representation. If all the characters are matched, it returns true else it will return false. For example:

|  |
| --- |
| public class EqualsExample{  public static void main(String args[]){  String s1="hello";  String s2="hello";  String s3="hi";  System.out.println(s1.equalsIgnoreCase(s2));   // returns true  System.out.println(s1.equalsIgnoreCase(s3));   // returns false  }  } |

**Java** **String equalsIgnoreCase ():**This method compares two string on the basis of content but it does not check the case like equals () method. In this method, if the characters match, it returns true else false. For example:

|  |
| --- |
| public class EqualsIgnoreCaseExample{  public static void main(String args[]){  String s1="hello";  String s2="HELLO";  String s3="hi";  System.out.println(s1.equalsIgnoreCase(s2));   // returns true  System.out.println(s1.equalsIgnoreCase(s3));   // returns false  }  } |

**Java String toCharArray ():**This method converts the string into a character array i.e first it will calculate the length of the given Java String including spaces and then create an array of char type with the same content. For example:

|  |
| --- |
| StringToCharArrayExample{  public static void main(String args[]){  String s1="Welcome to Edureka";  char[] ch=s1.toCharArray();  for(int i=0;i<ch.length;i++){  System.out.println(ch[i]);  }  }  } |

**Java StringGetBytes ()**: The Java string getBytes () method returns the sequence of bytes or you can say the byte array of the string. For example:

|  |
| --- |
| public class StringGetBytesExample {  public static void main(String args[]){  String s1="ABC";  byte[] b=s1.getBytes();  for(int i=0;i<b.length;i++){  System.out.println (b[i]); //65,66,67.  }  }} |

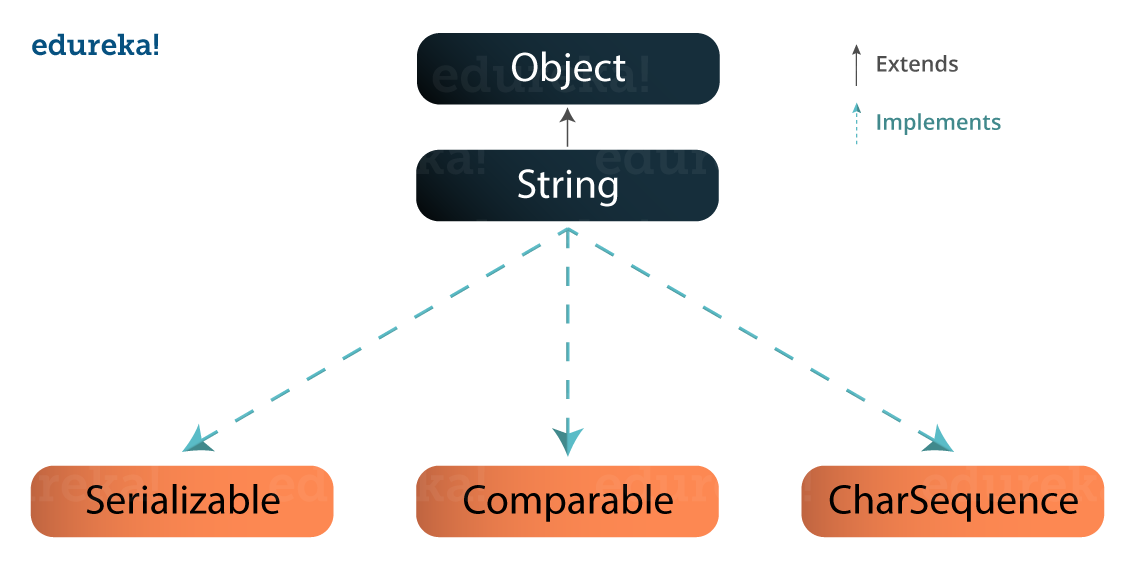
**Java String IsEmpty ()**: This method checks whether the String is empty or not. If the length of the String is 0, it returns true else false. For example:

|  |
| --- |
| public class IsEmptyExample{  public static void main(String args[]) {  String s1="";  String s2="hello";  System.out.println(s1.isEmpty());     // returns true  System.out.println(s2.isEmpty());     // returns false  }} |

**Java String endsWith ()**: The Java String endsWith () method checks if this string ends with the given suffix. If it returns with the given suffix, it will return true else returns false. For example:

|  |
| --- |
| public class EndsWithExample{  public static void main(String args[]) {  String s1="hello how are you”;  System.out.println(s1.endsWith("u"));       // returns true  System.out.println(s1.endsWith("you"));     // returns true  System.out.println(s1.endsWith("how"));     // returns false  }} |

**Java String class implements three interfaces, namely** – Serializable, Comparable and CharSequence.



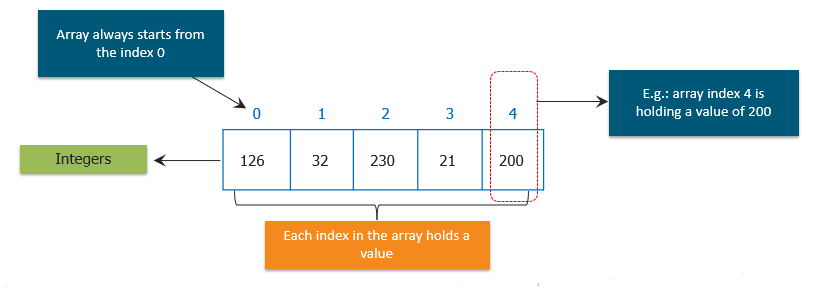
**String manipulations are resource consuming, Java provides two utility classes:** StringBuffer and StringBuilder.

**StringBuffer** and **StringBuilder** are mutable classes. **StringBuffer** operations are thread-safe and synchronized whereas **StringBuilder** operations are not thread-safe.

**StringBuffer** is to be used when multiple threads are working on same **String and StringBuilder** in the single threaded environment.

**StringBuilder** performance is faster when compared to **StringBuffer** because of no overhead of synchronized.

**What are Java Arrays?** Arrays in Java are homogeneous data structures implemented in Java as objects. Arrays store one or more values of a specific data type and provide indexed access to store the same. A specific element in an array is accessed by its index. Arrays offer a convenient means of grouping related information.



**Passing Java Array to a Method**

|  |
| --- |
| public class PMethods{  public static void display(int y[])     {               System.out.println(y[0]);               System.out.println(y[1]);               System.out.println(y[2]);       }  public static void main(String args[])  {       int x[] = { 1, 2, 3 };       display(x);       }  } |

**What is a Java Thread?** A thread is actually a lightweight process. Unlike many other computer languages, Java provides built-in support for multithreaded programming. A multithreaded program contains two or more parts that can run **concurrently**. Each part of such a program is called thread and each thread defines a separate path of execution. Thus, multithreading is a specialized form of multitasking.

**The Java Thread Model:-**The Java run-time system depends on threads for many things. Threads reduce inefficiency by preventing the waste of CPU cycles. Threads exist in several states. Following are those states:

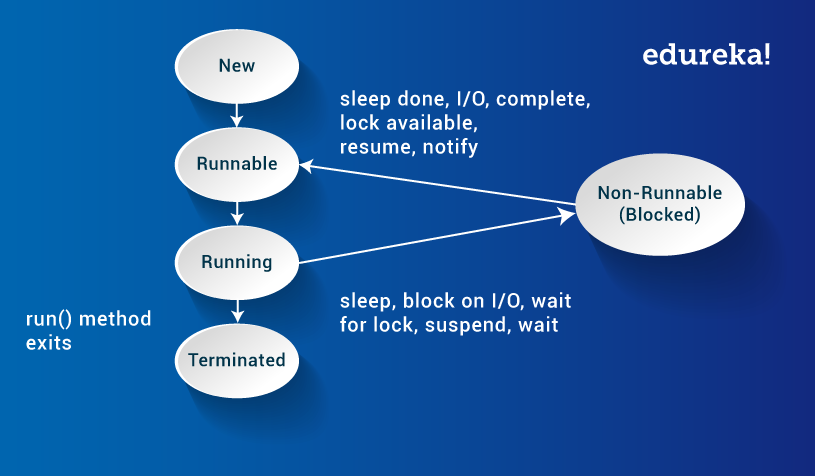
**New** – When we create an instance of Thread class, a thread is in a new state.

**Running –**TheJava thread is in running state.

**Suspended** – A running thread can be **suspended**, which temporarily suspends its activity. A suspended thread can then be resumed, allowing it to pick up where it left off.

**Blocked** – A java thread can be blocked when waiting for a resource.

**Terminated** – A thread can be terminated, which halts its execution immediately at any given time. Once a thread is terminated, it cannot be resumed.

****

**Multithreading in Java: Thread Class and Runnable Interface: -** Java’s multithreading system is built upon the Thread class, its methods, and its companion interface, **Runnable**. To create a new thread, your program will either extend **Thread** or **implement** the **Runnable** **interface**. The Thread class defines several methods that help manage threads. The table below displays the same:

|  |  |
| --- | --- |
| **Method** | **Meaning** |
| getName | Obtain thread’s name |
| getPriority | Obtain thread’s priority |
| isAlive | Determine if a thread is still running |
| join | Wait for a thread to terminate |
| run | Entry point for the thread |
| sleep | Suspend a thread for a period of time |
| start | Start a thread by calling its run method |

**Main Java Thread: -** how to use Thread and Runnable interface to create and manage threads, beginning with the **main java thread,** that all Java programs have. So, let us discuss the main thread.

**Why is Main Thread so important?**

Because this thread effects the other ‘child’ threads

Because it performs various shutdown actions

It is created automatically when your program is started.

**How to Create a Java Thread?**

By **implementing** the **Runnable** **interface**.

By **extending** the **Thread**

**Runnable Interface: -** The easiest way to create a thread is to create a class that implements the **Runnable** interface. To implement Runnable interface, a class need only implement a single method called run ( ), which is declared like this: **public void run ( )**

|  |
| --- |
| public class MyClass implements Runnable {  public void run(){  System.out.println("MyClass running");     }  }  Thread t1 = new Thread(new MyClass ());  t1.start(); |

**Extending Java Thread: -** The second way to create a thread is to create a new class that extends Thread, then override the run () method and then to create an instance of that class. The run () method is what is executed by the thread after you call start (). Here is an example of creating a Java Thread subclass:

|  |
| --- |
| public class MyClass extends Thread {       public void run(){       System.out.println("MyClass running");     }  }  MyClass t1 = new MyClass ();  T1.start(); |

**Creating Multiple Threads**

|  |
| --- |
| class MyThread implements Runnable {  String name;  Thread t;      MyThread String thread){      name = threadname;      t = new Thread(this, name);  System.out.println("New thread: " + t);  t.start();  }  public void run() {   try {       for(int i = 5; i > 0; i--) {       System.out.println(name + ": " + i);        Thread.sleep(1000);  }  }catch (InterruptedException e) {       System.out.println(name + "Interrupted");  }       System.out.println (name + " exiting.");  }  }    class MultiThread {  public static void main(String args[]) {       new MyThread("One");       new MyThread("Two");       new NewThread("Three");  try {       Thread.sleep(10000);  } catch (InterruptedException e) {        System.out.println("Main thread Interrupted");  }        System.out.println ("Main thread exiting.");        }  } |

**What is a Java Collection Framework?** A Java collection framework provides an architecture to store and manipulate a group of objects. A Java collection framework includes the following:

Interfaces

Classes

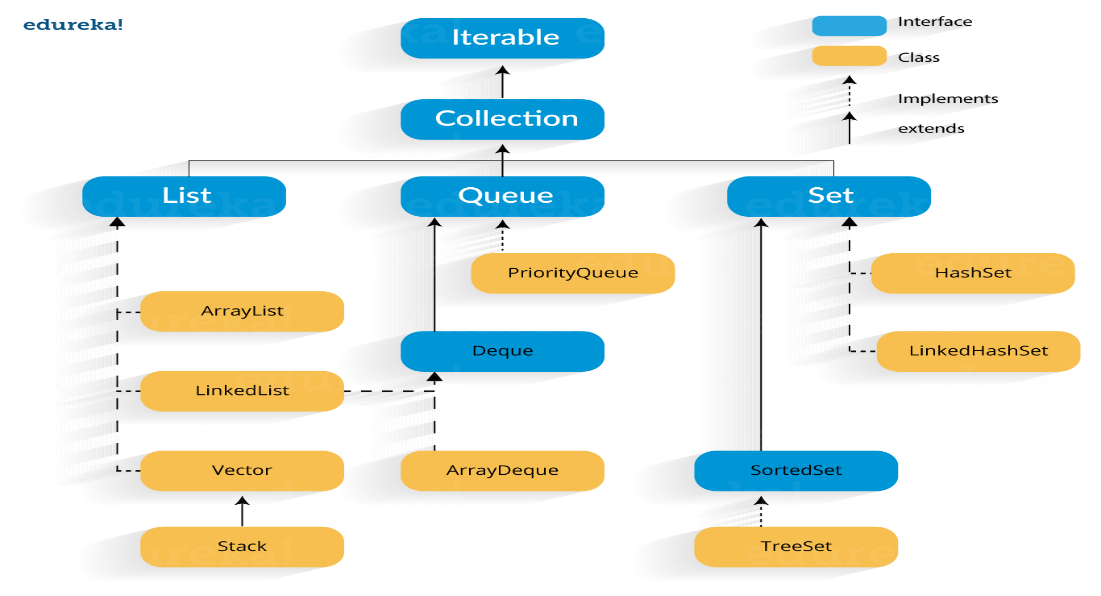
Algorithm

**Interfaces**: Interface in Java refers to the abstract data types. They allow Java collections to be manipulated independently from the details of their representation. Also, they form a hierarchy in object-oriented programming languages.

**Classes:**Classes in Java are the implementation of the collection interface. It basically refers to the data structures that are used again and again.

**Algorithm:** Algorithm refers to the methods which are used to perform operations such as searching and sorting, on objects that implement collection interfaces.

**Java Collection Framework Hierarchy:-**



**Java Collections: Interface**

**Iterator interface:** Iterator is an interface that iterates the elements. It is used to traverse the list and modify the elements. Iterator interface has three methods which are mentioned below:

**public boolean hasNext() –** This method returns true if iterator has more elements.

**public object next() –** It returns the element and moves the cursor pointer to the next element.

**public void remove() –** This method removes the last elements returned by the iterator.

**There are three components that extend the collection interface i.e List, Queue and Sets.**

**Java collections: List: -** A List is an ordered Collection of elements which may contain duplicates. It is an interface that extents the Collection interface. Lists are further classified into the following:

ArrayList

LinkedList

Vectors

**Array list:** ArrayList is the implementation of List Interface where the elements can be dynamically added or removed from the list. Also, the size of the list is increased dynamically if the elements are added more than the initial size. **ArrayList object = new ArrayList ();**

**Methods in array list are listed below:**

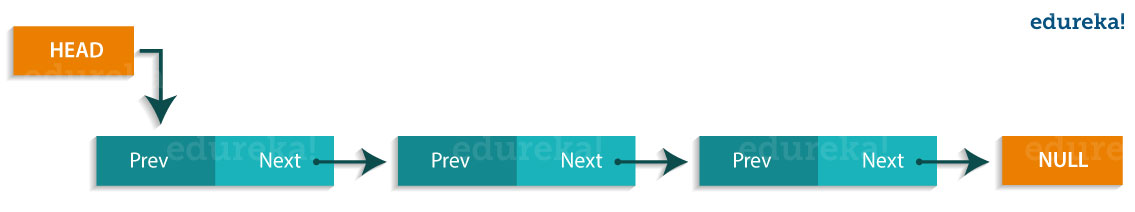
|  |  |
| --- | --- |
| **Method** | **Description** |
| boolean add(Collection c) | Appends the specified element to the end of a list. |
| void add(int index, Object element) | Inserts the specified element at the specified position. |
| void clear() | Removes all the elements from this list. |
| int lastIndexOf(Object o) | Return the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element. |
| Object clone() | Return a shallow copy of an ArrayList. |
| Object[] toArray() | Returns an array containing all the elements in the list. |
| void trimToSize() | Trims the capacity of this ArrayList instance to be the list’s current size. |

**Linked List:** Linked List is a sequence of links which contains items. Each link contains a connection to another link. **Syntax: Linkedlist object = new Linkedlist ();** **Java Linked List class uses two types of Linked list to store the elements:**

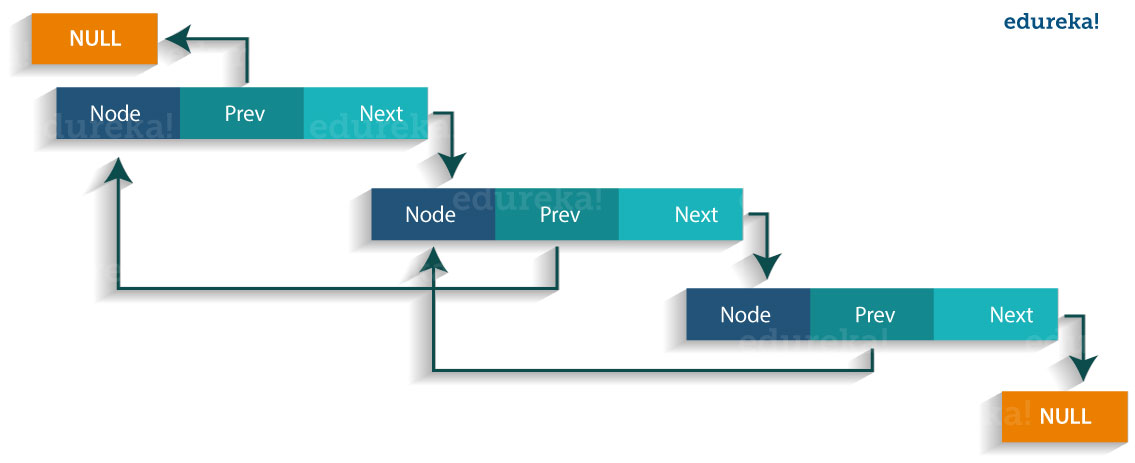
Singly Linked List

Doubly Linked List

**Singly Linked List:** In a singly linked list each node in this list stores the data of the node and a pointer or reference to the next node in the list.



**Doubly Linked List:** In a doubly Linked list, it has two references, one to the next node and another to previous node.

  
**Methods in the linked list are listed below:**

|  |  |
| --- | --- |
| **Method** | **Description** |
| boolean add( Object o) | It is used to append the specified element to the end of the vector. |
| boolean contains(Object o) | Returns true if this list contains the specified element. |
| void add (int index, Object element) | Inserts the element at the specified element in the vector. |
| void addFirst(Object o) | It is used to insert the given element at the beginning. |
| void addLast(Object o) | It is used to append the given element to the end. |
| int size() | It is used to return the number of elements in a list |
| boolean remove(Object o) | Removes the first occurrence of the specified element from this list. |
| int indexOf(Object element) | Returns the index of the first occurrence of the specified element in this list, or -1. |
| int lastIndexOf(Object element) | Returns the index of the last occurrence of the specified element in this list, or -1. |

**Vectors:** Vectors are similar to arrays, where the elements of the vector object can be accessed via an index into the vector. Vector implements a dynamic array. Also, the vector is not limited to a specific size, it can shrink or grow automatically whenever required. **It is similar to ArrayList, but with two differences:**

Vector is synchronized.

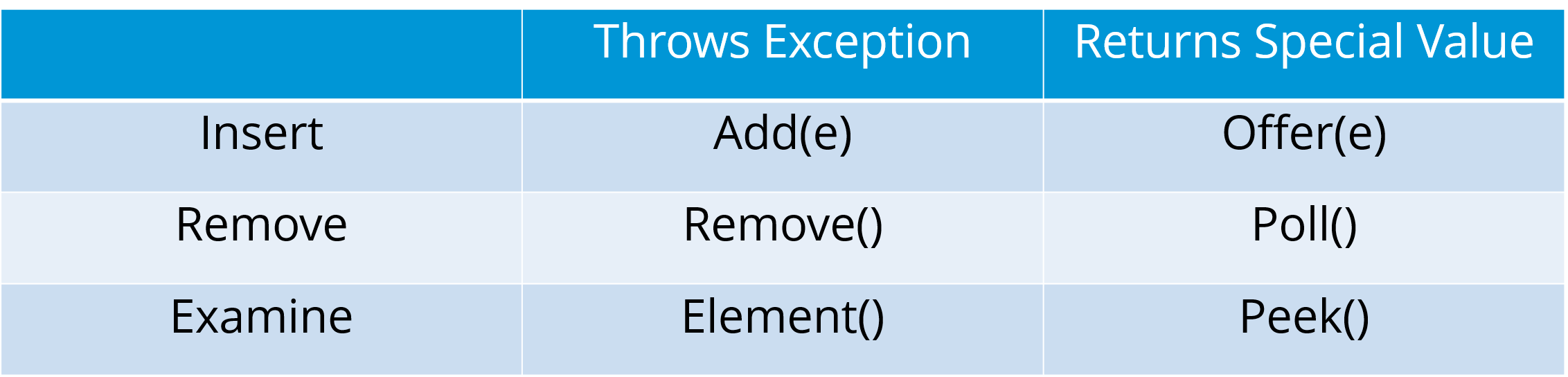
Vector contains many legacy methods that are not part of the collections framework.

**Syntax**: Vector object = new Vector (size, increment);

**Methods of the Vector class:**

|  |  |
| --- | --- |
| **Method** | **Description** |
| boolean add(Object o) | Appends the specified element to the end of the list. |
| void clear() | Removes all of the elements from this list. |
| void add(int index, Object element) | Inserts the specified element at the specified position. |
| boolean remove(Object o) | Removes the first occurrence of the specified element from this list. |
| boolean contains(Object element) | Returns true if this list contains the specified element. |
| int indexOfObject (Object element) | Returns the index of the first occurrence of the specified element in the list, or -1. |
| int size() | Returns the number of elements in this list. |
| int lastIndexOf(Object o) | Return the index of the last occurrence of the specified element in the list, or -1 if the list does not contain any element. |

**Java collections: Queue: -** Queue in Java follows a FIFO approach i.e. it orders the elements in **First in First out** manner. In a queue, the first element is removed first and last element is removed in the end. **Each basic method exists in two forms: one throws an exception if the operation fails, the other returns a special value.**



The elements of the **priority queue** are ordered according to their natural ordering, or by a Comparator provided at the queue construction time. The head of this queue is the least element with respect to the specified ordering.

**Methods of Java Queue interface:**

|  |  |
| --- | --- |
| **Method** | **Description** |
| boolean add(object) | Inserts the specified element into the queue and returns true if it is a success. |
| boolean offer(object) | Inserts the specified element into this queue. |
| Object remove() | Retrieves and removes the head of the queue. |
| Object poll() | Retrieves and removes the head of the queue, or returns null if the queue is empty. |
| Object element() | Retrieves, but does not remove the head of the queue. |
| Object peek() | Retrieves, but does not remove the head of this queue, or returns null if the queue is empty. |

**Java Collections: Sets: -** A Set refers to a collection that cannot contain duplicate elements. It is mainly used to model the mathematical set abstraction. Set has its implementation in various classes such as HashSet, TreeSet and LinkedHashSet.

**HashSet**: Java HashSet class creates a collection that use a hash table for storage. HashSet only contain unique elements and it inherits the AbstractSet class and implements Set interface. Also, it uses a mechanism ***hashing*** to store the elements.    
**Methods of Java HashSet class:**

|  |  |
| --- | --- |
| **Method** | **Description** |
| boolean add(Object o) | Adds the specified element to this set if it is not already present. |
| boolean contains(Object o) | Returns true if the set contains the specified element. |
| void clear() | Removes all the elements from the set. |
| boolean isEmpty() | Returns true if the set contains no elements. |
| boolean remove(Object o) | Remove the specified element from the set. |
| Object clone() | Returns a shallow copy of the HashSet instance: the elements themselves are not cloned. |
| Iterator iterator() | Returns an iterator over the elements in this set. |
| int size() | Return the number of elements in the set. |

**Linked HashSet:** Java LinkedHashSet class is a Hash table and Linked list implementation of the set interface. It contains only unique elements like HashSet. Linked HashSet also provides all optional set operations and maintains insertion order.

**TreeSet:** TreeSet class implements the Set interface that uses a tree for storage. The objects of this class are stored in the ascending order. Also, it inherits AbstractSet class and implements NavigableSet interface. It contains only unique elements like HashSet. In TreeSet class, access and retrieval time are faster.  
**Methods of Java TreeSet class:**

|  |  |
| --- | --- |
| **Method** | **Description** |
| boolean addAll(Collection c) | Add all the elements in the specified collection to this set. |
| boolean contains(Object o) | Returns true if the set contains the specified element. |
| boolean isEmpty() | Returns true if this set contains no elements. |
| boolean remove(Object o) | Remove the specified element from the set. |
| void add(Object o) | Add the specified element to the set. |
| void clear() | Removes all the elements from the set. |
| Object clone() | Return a shallow copy of this TreeSet instance. |
| Object first() | Return the first element currently in the sorted set. |
| Object last() | Return the last element currently in the sorted set. |
| int size() | Return the number of elements in the set. |

**What is the difference between all these sets? HashSet** stores elements in random order whereas **LinkedHashSet** stores elements according to insertion order and **TreeHashSet** stores according to natural ordering.

**Q1. Explain JVM, JRE and JDK?**

**JVM (Java Virtual Machine):**It is an abstract machine. It is a specification that provides run-time environment in which java bytecode can be executed. It follows three notations:

**Specification**: It is a document that describes the implementation of the Java virtual machine. It is provided by Sun and other companies.

**Implementation**: It is a program that meets the requirements of JVM specification.

**Runtime Instance**: An instance of JVM is created whenever you write a java command on the command prompt and run the class.

**JRE (Java Runtime Environment):**JRE refers to a runtime environment in which java bytecode can be executed. It implements the JVM (Java Virtual Machine) and provides all the class libraries and other support files that JVM uses at runtime. So JRE is a software package that contains what is required to run a Java program. Basically, it’s an implementation of the JVM which physically exists.

**JDK (Java Development Kit) :** It is the tool necessary to:-

Compile

Document

Package Java programs.

The JDK completely includes JRE which contains tools for Java programmers. The Java Development Kit is provided free of charge. Along with JRE, it includes an interpreter/loader, a compiler (javac), an archiver (jar), a documentation generator (javadoc) and other tools needed in Java development. In short, it contains JRE + development tools.

**Q2. Explain public static void main (String args []).**

**public:** Public is an access modifier, which is used to specify who can access this method. Public means that this Method will be accessible by any Class.

**static:** It is a keyword in java which identifies it is class based i.e it can be accessed without creating the instance of a Class.

**void:** It is the return type of the method. Void defines the method which will not return any value.

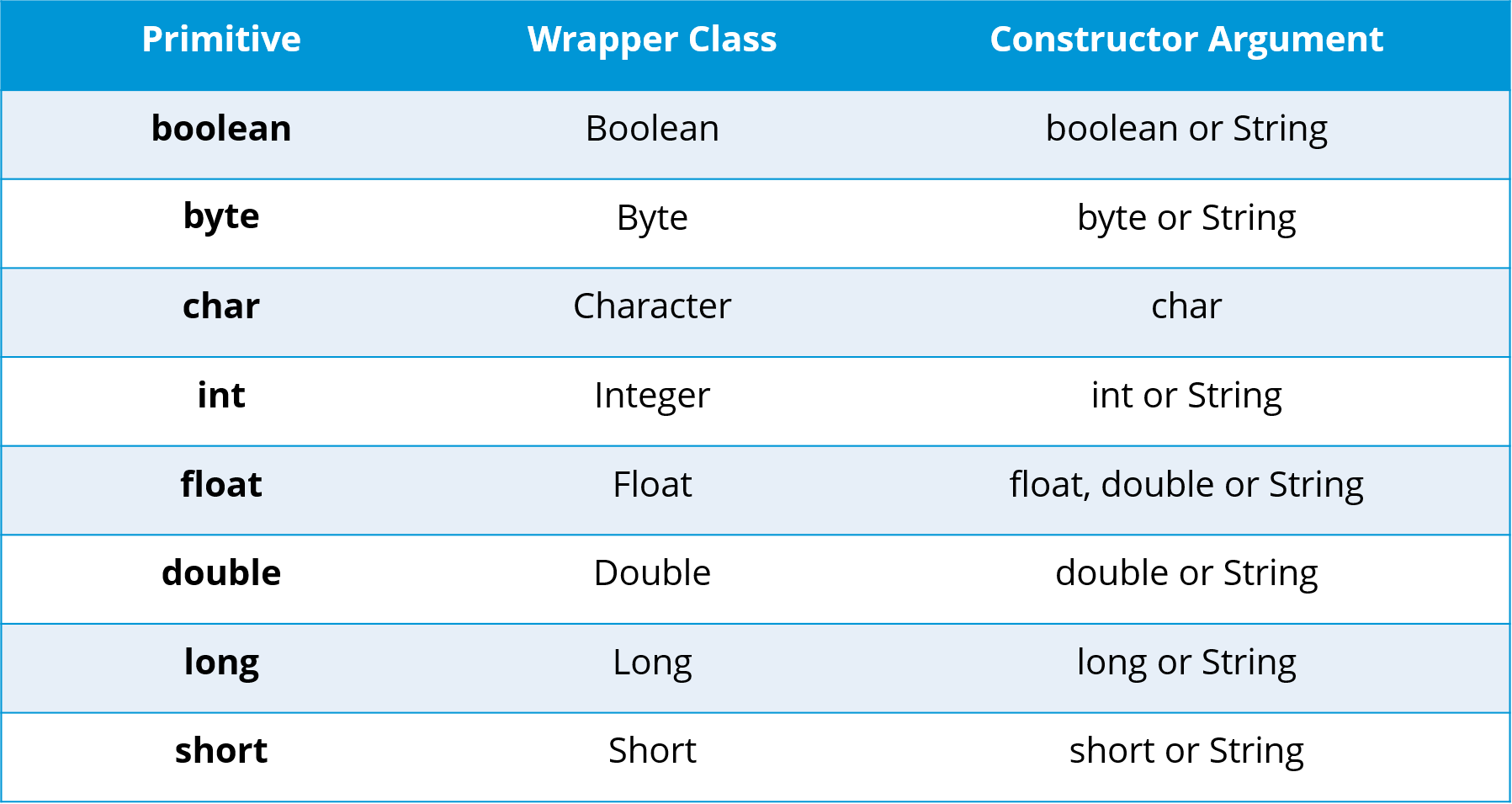
**main:** It is the name of the method which is searched by JVM as a starting point for an application with a particular signature only. It is the method where the main execution occurs.

**String args[] :** It is the parameter passed to the main method.

**Q3. Why Java is platform independent?** Platform independent practically means “write once run anywhere”. Java is called so because of its byte codes which can run on any system irrespective of its underlying operating system.

**Q4. Why java is not 100% Object-oriented?** Java is not 100% Object-oriented because it makes use of eight primitive datatypes such as boolean, byte, char, int, float, double, long, short which are not objects.

**Q5. What are wrapper classes?** Wrapper classes converts the java primitives into the reference types (objects). Every primitive data type has a class dedicated to it. These are known as wrapper classes because they “wrap” the primitive data type into an object of that class.



**Q6. What are constructors in Java?** In Java, constructor refers to a block of code which is used to initialize an object. It must have the same name as that of the class. Also, it has no return type and it is automatically called when an object is created. There are two types of constructors:

Default constructor

Parameterized constructor

**Q7. What is singleton class and how can we make a class singleton?** Singleton class is a class whose only one instance can be created at any given time, in one JVM. A class can be made singleton by making its constructor private.

**Q8. What is the difference between Array list and vector?**

|  |  |
| --- | --- |
| **Array List** | **Vector** |
| Array List is not synchronized. | Vector is synchronized. |
| Array List is fast as it’s non-synchronized. | Vector is slow as it is thread safe. |
| If an element is inserted into the Array List, it increases its Array size by 50%. | Vector defaults to doubling size of its array. |
| Array List does not define the increment size. | Vector defines the increment size. |
| Array List can only use Iterator for traversing an Array List. | Except Hash table, Vector is the only other class which uses both Enumeration and Iterator. |

**Q9. What is the difference between equals () and ==?**

Equals () method is defined in Object class in Java and used for checking equality of two objects defined by business logic.   
“==” or equality operator in Java is a binary operator provided by Java programming language and used to compare primitives and objects. **Public Boolean equals (Object o)** is the method provided by the Object class. The default implementation uses == operator to compare two objects. For example: method can be overridden like String class. **Equals ()** method is used to compare the values of two objects.

|  |
| --- |
| public class Equaltest {  public static void main(String[] args) {  String str1= new String(“ABCD”);  String str2= new String(“ABCD”);  if(Str1 == str2){  System.out.println("String 1 == String 2 is true");  }else{  System.out.println("String 1 == String 2 is false");  String Str3 = Str2;  if( Str2 == Str3){  System.out.println("String 2 == String 3 is true");  }else{  System.out.println("String 2 == String 3 is false");  }  if(Str1.equals(str2)){  System.out.println("String 1 equals string 2 is true");  }else{  System.out.println("String 1 equals string 2 is false");  }  }} |

**Q10. What are the differences between Heap and Stack Memory?**

|  |  |  |
| --- | --- | --- |
| **Features** | **Stack** | **Heap** |
| Memory | Stack memory is used only by one thread of execution. | Heap memory is used by all the parts of the application. |
| Access | Stack memory can’t be accessed by other threads. | Objects stored in the heap are globally accessible. |
| Memory Management | Follows LIFO manner to free memory. | Memory management is based on generation associated to each object. |
| Lifetime | Exists until the end of execution of the thread. | Heap memory lives from the start till the end of application execution. |
| Usage | Stack memory only contains local primitive and reference variables to objects in heap space. | Whenever an object is created, it’s always stored in the Heap space. |

**Q3. What is the difference between abstract classes and interfaces?**

|  |  |
| --- | --- |
| **Abstract Class** | **Interfaces** |
| **An abstract class can provide complete, default code and/or just the details that have to be overridden.** | **An interface cannot provide any code at all, just the signature.** |
| **In case of abstract class, a class may extend only one abstract class.** | **A Class may implement several interfaces.** |
| **An abstract class can have non-abstract methods.** | **All methods of an Interface are abstract.** |
| **An abstract class can have instance variables.** | **An Interface cannot have instance variables** |
| **An abstract class can have any visibility: public, private, protected.** | **An Interface visibility must be public (or) none.** |
| **If we add a new method to an abstract class then we have the option of providing default implementation and therefore all the existing code might work properly** | **If we add a new method to an Interface then we have to track down all the implementations of the interface and define implementation for the new method** |
| **An abstract class can contain constructors** | **An Interface cannot contain constructors** |
| **Abstract classes are fast** | **Interfaces are slow as it requires extra indirection to find corresponding method in the actual class** |

**Q5. Can you override a private or static method in Java?** You cannot override a private or static method in Java. If you create a similar method with same return type and same method arguments in child class then it will hide the super class method; this is known **as method hiding**. Similarly, you cannot override a private method in sub class because it’s not accessible there. What you can do is create another private method with the same name in the child class.

|  |
| --- |
| class Base {  private static void display() {  System.out.println("Static or class method from Base");  }  public void print() {  System.out.println("Non-static or instance method from Base");  }  class Derived extends Base {  private static void display() {  System.out.println("Static or class method from Derived");  }  public void print() {  System.out.println("Non-static or instance method from Derived");  }  public class test {  public static void main(String args[])  {  Base obj= new Derived();  obj1.display();  obj1.print();  }  } |

**Q7. What is association?** Association is a relationship where all object have their own lifecycle and there is no owner. Let’s take an **example of Teacher and Student.** Multiple students can associate with a single teacher and a single student can associate with multiple teachers but there is no ownership between the objects and both have their own lifecycle. **These relationship can be one to one, one to many, many to one and many to many.**

**Q8. What do you mean by aggregation?** Aggregation is a specialized form of Association where all object have their own lifecycle but there is ownership and child object cannot belongs to another parent object. Let’s take an **example of** **Department and teacher.** A single teacher cannot belongs to multiple departments, but if we delete the department teacher object will not destroy.

**Q9. What is composition in Java?** Composition is again specialized form of Aggregation and we can call this as a “death” relationship. It is a strong type of Aggregation. Child object does not have their lifecycle and if parent object deletes all child object will also be deleted. Let’s take again an **example of relationship between House and rooms.** House can contain multiple rooms there is no independent life of room and any room cannot belongs to two different house if we delete the house room will automatically delete.

**Q1. What is a servlet?**

Java Servlet is server side technologies to extend the capability of web servers by providing support for dynamic response and data persistence.

The javax.servlet and javax.servlet.http packages provide interfaces and classes for writing our own servlets.

All servlets must implement the javax.servlet.Servlet interface, which defines servlet lifecycle methods. When implementing a generic service, we can extend the GenericServlet class provided with the Java Servlet API. The HttpServlet class provides methods, such as doGet () and doPost (), for handling HTTP-specific services.

Most of the times, web applications are accessed using HTTP protocol and that’s why we mostly extend HttpServlet class. Servlet API hierarchy is shown in below image.

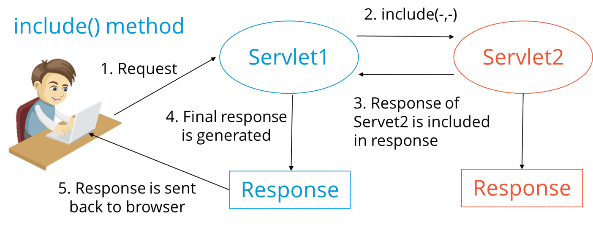
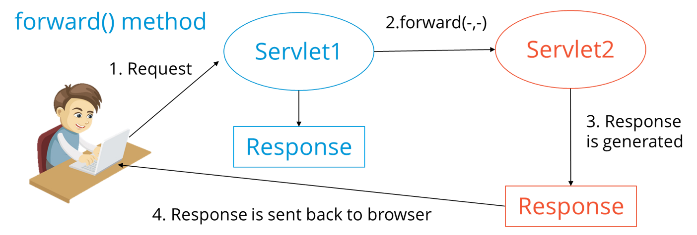
**Q2. What are the differences between Get and Post methods?**

|  |  |
| --- | --- |
| **Get** | **Post** |
| Limited amount of data can be sent because data is sent in header. | Large amount of data can be sent because data is sent in body. |
| Not Secured because data is exposed in URL bar. | Secured because data is not exposed in URL bar. |
| Can be bookmarked | Cannot be bookmarked |
| Idempotent | Non-Idempotent |
| It is more efficient and used than Post | It is less efficient and used |

**Q3. What is Request Dispatcher?** Request Dispatcher interface is used to forward the request to another resource that can be HTML, JSP or another servlet in same application. We can also use this to include the content of another resource to the response. There are two methods defined in this interface:

void forward()

void include()



**Q4. What are the differences between forward () method and sendRedirect () methods?**

|  |  |
| --- | --- |
| **Forward() method** | **SendRedirect() method** |
| forward () sends the same request to another resource. | sendRedirect () method sends new request always because it uses the URL bar of the browser. |
| forward () method works at server side. | sendRedirect () method works at client side. |
| forward () method works within the server only. | sendRedirect () method works within and outside the server. |

**Q5. What is the life-cycle of a servlet?**

Servlet is loaded

Servlet is instantiated

Servlet is initialized

Service the request

Servlet is destroyed

**Q6. How does cookies work in Servlets?**

Cookies are text data sent by server to the client and it gets saved at the client local machine.

Servlet API provides cookies support through javax.servlet.http.Cookie class that implements Serializable and Cloneable interfaces.

HttpServletRequest getCookies() method is provided to get the array of Cookies from request, since there is no point of adding Cookie to request, there are no methods to set or add cookie to request.

Similarly HttpServletResponse addCookie (Cookie c) method is provided to attach cookie in response header, there are no getter methods for cookie.

**Q7. What are the differences between ServletContext vs ServletConfig?**

|  |  |
| --- | --- |
| **ServletConfig** | **ServletContext** |
| Servlet config object represent single servlet | It represent whole web application running on particular JVM and common for all the servlet |
| It’s like local parameter associated with particular servlet | It’s like global parameter associated with whole application |
| It’s a name value pair defined inside the servlet section of web.xml file so it has servlet wide scope | ServletContext has application wide scope so define outside of servlet tag in web.xml file. |
| getServletConfig() method is used to get the config object | getServletContext () method is  used to get the context object. |
| for example shopping cart of a user is a specific to particular user so here we can use servlet config | To get the MIME type of a file or application session related information is stored using servlet context object. |

**Q8. What are the different methods of session management in servlets?** Session is a conversational state between client and server and it can consists of multiple request and response between client and server. Since HTTP and Web Server both are stateless, the only way to maintain a session is when some unique information about the session (session id) is passed between server and client in every request and response. **Some of the common ways of session management in servlets are:**

User Authentication

HTML Hidden Field

Cookies

URL Rewriting

Session Management API

**Q1. What is JDBC Driver?** JDBC Driver is a software component that enables java application to interact with the database. **There are 4 types of JDBC drivers:**

JDBC-ODBC bridge driver

Native-API driver (partially java driver)

Network Protocol driver (fully java driver)

Thin driver (fully java driver)

**Q2. What are the steps to connect to a database in java?**

Registering the driver class

Creating connection

Creating statement

Executing queries

Closing connection

**Q3. What are the JDBC API components?** The java.sql package contains interfaces and classes for JDBC API.

**Interfaces:**

Connection

Statement

PreparedStatement

ResultSet

ResultSetMetaData

DatabaseMetaData

CallableStatement etc.

**Classes:**

DriverManager

Blob

Clob

Types

SQLException etc.

**Q4. What is the role of JDBC DriverManager class?** The DriverManager class manages the registered drivers. It can be used to register and unregister drivers. It provides factory method that returns the instance of Connection.

**Q5. What is JDBC Connection interface?** The Connection interface maintains a session with the database. It can be used for transaction management. It provides factory methods that returns the instance of Statement, PreparedStatement, CallableStatement and DatabaseMetaData.



**Q6.  What is the purpose of JDBC ResultSet interface?** The ResultSet object represents a row of a table. It can be used to change the cursor pointer and get the information from the database.

**Q7. What is JDBC ResultSetMetaData interface?** The ResultSetMetaData interface returns the information of table such as total number of columns, column name, column type etc.

**Q8. What is JDBC DatabaseMetaData interface?** The DatabaseMetaData interface returns the information of the database such as username, driver name, driver version, number of tables, number of views etc.

**Q9. What do you mean by batch processing in JDBC?** Batch processing helps you to group related SQL statements into a batch and execute them instead of executing a single query. By using batch processing technique in JDBC, you can execute multiple queries which makes the performance faster.

**Q10. What is the difference between execute, executeQuery, executeUpdate?**

Statement **execute(String query)** is used to execute any SQL query and it returns TRUE if the result is an ResultSet such as running Select queries. The output is FALSE when there is no ResultSet object such as running Insert or Update queries. We can use *getResultSet ()* to get the ResultSet and *getUpdateCount ()* method to retrieve the update count.

Statement **executeQuery (String query)** is used to execute Select queries and returns the ResultSet. ResultSet returned is never null even if there are no records matching the query. When executing select queries we should use executeQuery method so that if someone tries to execute insert/update statement it will throw java.sql.SQLException with message “executeQuery method cannot be used for update”.

Statement **executeUpdate (String query)** is used to execute Insert/Update/Delete (DML) statements or DDL statements that returns nothing. The output is int and equals to the row count for SQL Data Manipulation Language (DML) statements. For DDL statements, the output is 0.

**You should use execute () method only when you are not sure about the type of statement else use executeQuery or executeUpdate method.**

**Q1. What is a Spring?** The spring framework as “an application framework and inversion of control container for the Java platform. The framework’s core features can be used by any Java application, but there are extensions for building web applications on top of the Java EE platform.” Spring is essentially a lightweight, integrated framework that can be used for developing enterprise applications in java.

**Q2. Name the different modules of the Spring framework.**

Spring Context – for dependency injection.

Spring AOP – for aspect oriented programming.

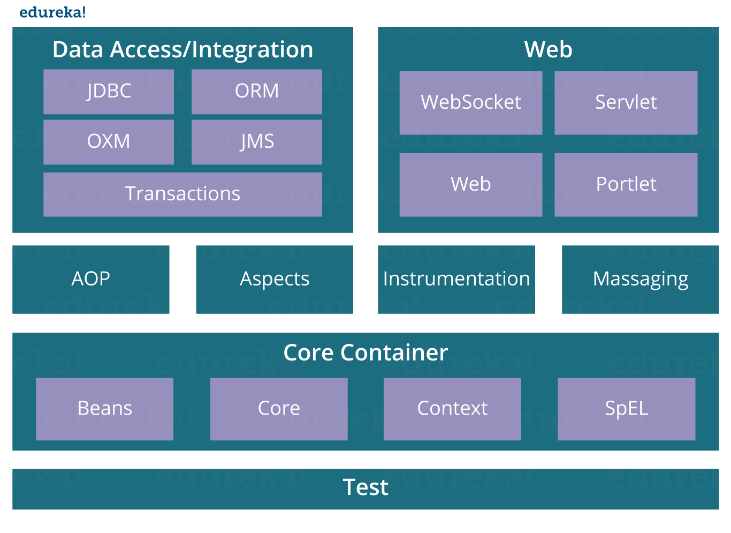
Spring DAO – for database operations using DAO pattern

Spring JDBC – for JDBC and DataSource support.

Spring ORM – for ORM tools support such as Hibernate

Spring Web Module – for creating web applications.

Spring MVC – Model-View-Controller implementation for creating web applications, web services etc.



**Q3. List some of the important annotations in annotation-based Spring configuration.**

@Required

@Autowired

@Qualifier

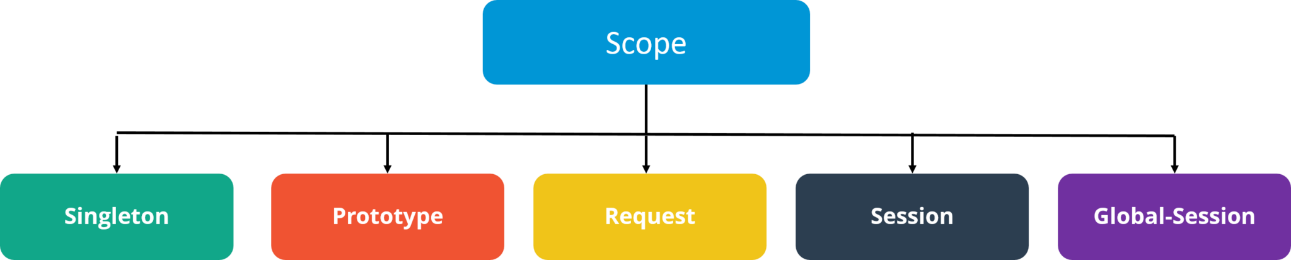
@Resource

@PostConstruct

@PreDestroy

**Q4. Explain Bean in Spring and List the different Scopes of Spring bean.** Beans are objects that form the backbone of a Spring application. They are managed by the Spring IoC container. In other words, a bean is an object that is instantiated, assembled, and managed by a Spring IoC container.

**There are five Scopes defined in Spring beans.**



**Singleton:** Only one instance of the bean will be created for each container. This is the default scope for the spring beans. While using this scope, make sure spring bean doesn’t have shared instance variables otherwise it might lead to data inconsistency issues because it’s not thread-safe.

**Prototype**: A new instance will be created every time the bean is requested.

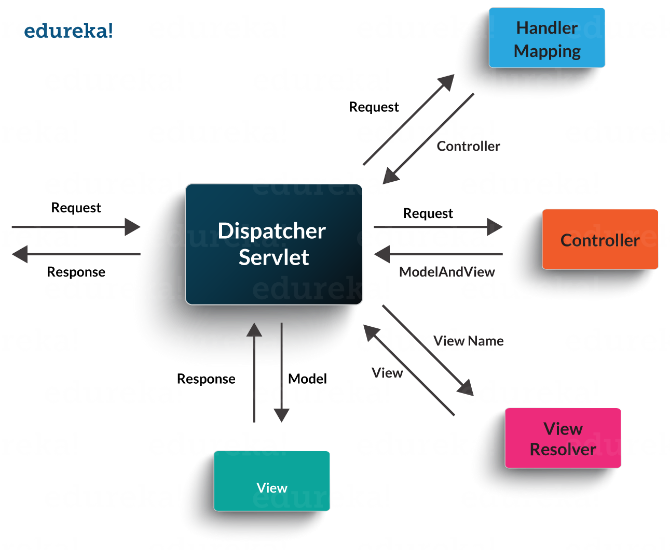
**Request:** This is same as prototype scope, however it’s meant to be used for web applications. A new instance of the bean will be created for each HTTP request.

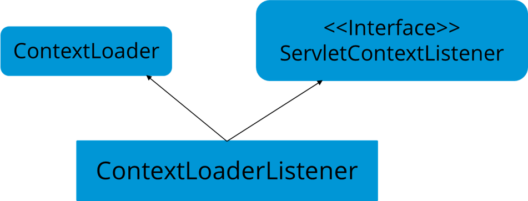
**Session:** A new bean will be created for each HTTP session by the container.

**Global-session:** This is used to create global session beans for Portlet applications.

**Q5. Explain the role of DispatcherServlet and ContextLoaderListener.**

**DispatcherServlet** is basically the front controller in the Spring MVC application as it loads the spring bean configuration file and initializes all the beans that have been configured. If annotations are enabled, it also scans the packages to configure any bean annotated with @Component, @Controller, @Repository or @Service annotations.

****

**ContextLoaderListener,** on the other hand, is the listener to start up and shut down the WebApplicationContext in Spring root. Some of its important functions includes tying up the lifecycle of Application Context to the lifecycle of the ServletContext and automating the creation of ApplicationContext.

**Q6. What are the differences between constructor injection and setter injection?**

|  |  |
| --- | --- |
| **Constructor Injection** | **Setter Injection** |
| No Partial Injection | Partial Injection |
| Desn’t override the setter property | Overrides the constructor property if both are defined. |
| Creates new instance if any modification occurs | Doesn’t create new instance if you change the property value |
| Better for too many properties | Better for few properties. |

**Q7. What is autowiring in Spring? What are the autowiring modes?** Autowiring enables the programmer to inject the bean automatically. We don’t need to write explicit injection logic. Let’s see the code to inject bean using dependency injection.

**<bean id=“emp” class=“com.javatpoint.Employee” autowire=“byName” />**

**The autowiring modes are given below:**

|  |  |
| --- | --- |
| **Mode** | **Description** |
| no | this is the default mode, it means autowiring is not enabled. |
| byName | Injects the bean based on the property name. It uses setter method. |
| byType | Injects the bean based on the property type. It uses setter method. |
| constructor | It injects the bean using constructor |

**Q8. How to handle exceptions in Spring MVC Framework?** Spring MVC Framework provides following ways to help us achieving robust exception handling.

**Controller Based:** We can define exception handler methods in our controller classes. All we need is to annotate these methods with @ExceptionHandler annotation.

**Global Exception Handler:** Exception Handling is a cross-cutting concern and Spring provides @ControllerAdvice annotation that we can use with any class to define our global exception handler.

**HandlerExceptionResolver implementation:**  For generic exceptions, most of the times we serve static pages. Spring Framework provides HandlerExceptionResolver interface that we can implement to create global exception handler. The reason behind this additional way to define global exception handler is that Spring framework also provides default implementation classes that we can define in our spring bean configuration file to get spring framework exception handling benefits.

**Q9. What are some of the important Spring annotations which you have used?**

**@Controller** – for controller classes in Spring MVC project.

**@RequestMapping** – for configuring URI mapping in controller handler methods. This is a very important annotation, so you should go through Spring MVC RequestMapping Annotation Examples

**@ResponseBody** – for sending Object as response, usually for sending XML or JSON data as response.

**@PathVariable** – for mapping dynamic values from the URI to handler method arguments.

**@Autowired** – for autowiring dependencies in spring beans.

**@Qualifier** – with @Autowired annotation to avoid confusion when multiple instances of bean type is present.

**@Service** – for service classes.

**@Scope** – for configuring scope of the spring bean.

**@Configuration, @ComponentScan and @Bean** – for java based configurations.

**AspectJ annotations for configuring aspects and advices, @Aspect, @Before, @After, @Around, @Pointcut etc.**

**Q10. How to integrate Spring and Hibernate Frameworks?**

We can use Spring ORM module to integrate Spring and Hibernate frameworks, if you are using Hibernate 3+ where SessionFactory provides current session, then you should avoid usingHibernateTemplate or HibernateDaoSupport classes and better to use DAO pattern with dependency injection for the integration.

Also Spring ORM provides support for using Spring declarative transaction management, so you should utilize that rather than going for hibernate boiler-plate code for transaction management.

**Q1. What is Hibernate Framework?**

Object-relational mapping or ORM is the programming technique to map application domain model objects to the relational database tables. Hibernate is java based ORM tool that provides framework for mapping application domain objects to the relational database tables and vice versa.

Hibernate provides reference implementation of Java Persistence API, that makes it a great choice as ORM tool with benefits of loose coupling. We can use Hibernate persistence API for CRUD operations. Hibernate framework provide option to map plain old java objects to traditional database tables with the use of JPA annotations as well as XML based configuration.

Similarly hibernate configurations are flexible and can be done from XML configuration file as well as programmatically.

**Q2. What are the important benefits of using Hibernate Framework?**

Hibernate eliminates all the boiler-plate code that comes with JDBC and takes care of managing resources, so we can focus on business logic.

Hibernate framework provides support for XML as well as JPA annotations, that makes our code implementation independent.

Hibernate provides a powerful query language (HQL) that is similar to SQL. However, HQL is fully object-oriented and understands concepts like inheritance, polymorphism and association.

Hibernate is an open source project from Red Hat Community and used worldwide. This makes it a better choice than others because learning curve is small and there are tons of online documentations and help is easily available in forums.

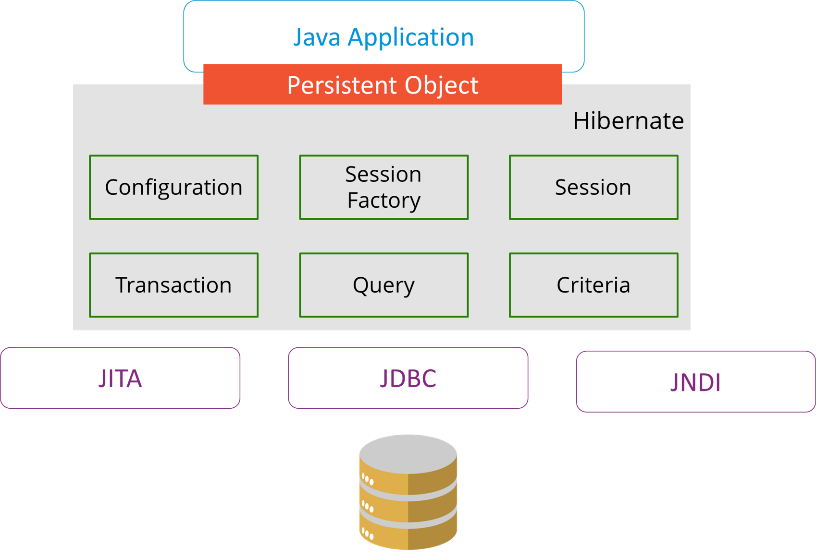
Hibernate is easy to integrate with other Java EE frameworks, it’s so popular that Spring Framework provides built-in support for integrating hibernate with Spring applications.

Hibernate supports lazy initialization using proxy objects and perform actual database queries only when it’s required.

Hibernate cache helps us in getting better performance.

For database vendor specific feature, hibernate is suitable because we can also execute native sql queries.

**Q3. Explain Hibernate architecture?**



**Q4 what are the differences between get and load methods?**

|  |  |
| --- | --- |
| **get()** | **load()** |
| Returns null if object is not found. | Throws ObjectNotFoundException if object is not found. |
| get() method always hit the database. | load() method doesn’t hit the database. |
| It returns real object not proxy. | It returns proxy object. |
| It should be used if you are not sure about the existence of instance. | It should be used if you are sure that instance exists. |

**Q5 What are the advantages of Hibernate over JDBC?**

Hibernate removes a lot of boiler-plate code that comes with JDBC API, the code looks more cleaner and readable.

Hibernate supports inheritance, associations and collections. These features are not present with JDBC API.

Hibernate implicitly provides transaction management, in fact most of the queries can’t be executed outside transaction. In JDBC API, we need to write code for transaction management using commit and rollback.

JDBC API throws SQLException that is a checked exception, so we need to write a lot of try-catch block code. Most of the times it’s redundant in every JDBC call and used for transaction management. Hibernate wraps JDBC exceptions and throw JDBCException or HibernateExceptionun-checked exception, so we don’t need to write code to handle it. Hibernate built-in transaction management removes the usage of try-catch blocks.

Hibernate Query Language (HQL) is more object oriented and close to java programming language. For JDBC, we need to write native sql queries.

Hibernate supports caching that is better for performance, JDBC queries are not cached hence performance is low.

Hibernate provide option through which we can create database tables too, for JDBC tables must exist in the database.

Hibernate configuration helps us in using JDBC like connection as well as JNDI DataSource for connection pool. This is very important feature in enterprise application and completely missing in JDBC API.

Hibernate supports JPA annotations, so code is independent of implementation and easily replaceable with other ORM tools. JDBC code is very tightly coupled with the application.

**Q1. What are the life-cycle methods for a jsp?**

|  |  |
| --- | --- |
| **Method** | **Description** |
| public void jspInit() | It is invoked only once, same as init method of servlet. |
| public void \_jspService(ServletRequest request,ServletResponse)throws ServletException,IOException | It is invoked at each request, same as service() method of servlet. |
| public void jspDestroy() | It is invoked only once, same as destroy() method of servlet. |

**Q2. What are the JSP implicit objects?**

JSP provides 9 implicit objects by default. They are as follows:

|  |  |
| --- | --- |
| **Object** | **Type** |
| 1) out | JspWriter |
| 2) request | HttpServletRequest |
| 3) response | HttpServletResponse |
| 4) config | ServletConfig |
| 5) session | HttpSession |
| 6) application | ServletContext |
| 7) pageContext | PageContext |
| 8) page | Object |
| 9) exception | Throwable |

**Q3. What are the differences between include directive and include action?**

|  |  |
| --- | --- |
| **include directive** | **include action** |
| The include directive includes the content at page translation time. | The include action includes the content at request time. |
| The include directive includes the original content of the page so page size increases at runtime. | The include action doesn’t include the original content rather invokes the include() method of Vendor provided class. |
| It’s better for static pages. | It’s better for dynamic pages. |

**Q4. How to disable caching on back button of the browser?**

**<**%  
response.setHeader(“Cache-Control”,”no-store”);  
response.setHeader(“Pragma”,”no-cache”);  
response.setHeader (“Expires”, “0”);                    //prevents caching at the proxy server  
%**>**

**Q5. What are the different tags provided in JSTL?**

There are 5 type of JSTL tags.

core tags

sql tags

xml tags

internationalization tags

functions tags

**Q6. How to disable session in JSP?**

**<**%@ page session=“false” %**>**

**Q7.  How to delete a Cookie in a JSP?**

The following code explain how to delete a Cookie in a JSP :

|  |
| --- |
| Cookie mycook = new Cookie("name1","value1");    response.addCookie(mycook1);    Cookie killmycook = new Cookie("mycook1","value1");    killmycook . set MaxAge ( 0 );    killmycook . set Path ("/");    killmycook . addCookie ( killmycook 1 ); |

**Q8. Explain the jspDestroy() method.**

jspDestry() method is invoked from javax.servlet.jsp.JspPage interface whenever a JSP page is about to be destroyed. Servlets destroy methods can be easily overridden to perform cleanup, like when closing a database connection.

**Q9.  How is JSP better than Servlet technology?**

JSP is a technology on the server’s side to make content generation simple. They are document centric, whereas servlets are programs. A Java server page can contain fragments of Java program, which execute and instantiate Java classes. However, they occur inside HTML template file. It provides the framework for development of a Web Application.

**Q10. Why should we not configure JSP standard tags in web.xml?**

We don’t need to configure JSP standard tags in web.xml because when container loads the web application and find TLD files, it automatically configures them to be used directly in the application JSP pages. We just need to include it in the JSP page using taglib directive.

**Q1. What is difference between Error and Exception?**

An error is an irrecoverable condition occurring at runtime. Such as OutOfMemory error. These JVM errors you can not repair them at runtime.Though error can be caught in catch block but the execution of application will come to a halt and is not recoverable.

While exceptions are conditions that occur because of bad input or human error etc. e.g. FileNotFoundException will be thrown if the specified file does not exist. Or a NullPointerException will take place if you try using a null reference. In most of the cases it is possible to recover from an exception

**Q2. How can you handle Java exceptions?**

There are five keywords used to handle exceptions in java:

try

catch

finally

throw

throws

**Q3. What are the differences between Checked Exception and Unchecked Exception?**

Checked Exception

The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions.

Checked exceptions are checked at compile-time.

Example: IOException, SQLException etc.

Unchecked Exception

The classes that extend RuntimeException are known as unchecked exceptions.

Unchecked exceptions are not checked at compile-time.

Example: ArithmeticException, NullPointerException etc.

**Q4. What purpose does the keywords final, finally, and finalize fulfill?**

**Final:**

Final is used to apply restrictions on class, method and variable. Final class can’t be inherited, final method can’t be overridden and final variable value can’t be changed.

|  |
| --- |
| class FinalVarExample {  public static void main( String args[])  {  final int a=10;   // Final variable  a=50;             //Error as value can't be changed  } |

**Finally**

Finally is used to place important code, it will be executed whether exception is handled or not.

|  |
| --- |
| class FinallyExample {  public static void main(String args[]){  try {  int x=100;  }  catch(Exception e) {  System.out.println(e);  }  finally {  System.out.println("finally block is executing");}  }}  } |

**Finalize**

Finalize is used to perform clean up processing just before object is garbage collected.

|  |
| --- |
| class FinalizeExample {  public void finalize() {  System.out.println("Finalize is called");  }  public static void main(String args[])  {  FinalizeExample f1=new FinalizeExample();  FinalizeExample f2=new FinalizeExample();  f1= NULL;  f2=NULL;  System.gc();  }  } |

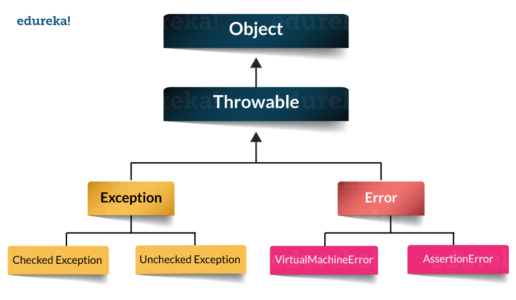
**Q5. What are the differences between throw and throws?**

|  |  |
| --- | --- |
| **throw keyword** | **throws keyword** |
| Throw is used to explicitly throw an exception. | Throws is used to declare an exception. |
| Checked exceptions can not be propagated with throw only. | Checked exception can be propagated with throws. |
| Throw is followed by an instance. | Throws is followed by class. |
| Throw is used within the method. | Throws is used with the method signature. |
| You cannot throw multiple exception | You can declare multiple exception e.g. public void method()throws IOException,SQLException. |

**Q6. What is exception hierarchy in java?**

The hierarchy is as follows:

Throwable is a parent class of all Exception classes. There are two types of Exceptions: Checked exceptions and UncheckedExceptions or RunTimeExceptions. Both type of exceptions extends Exception class whereas errors are further classified into Virtual Machine error and Assertion error.



**Q7. How to create a custom Exception?**

To create you own exception extend the Exception class or any of its subclasses.

class New1Exception extends Exception { }               // this will create Checked Exception

class NewException extends IOExcpetion { }             // this will create Checked exception

class NewException extends NullPonterExcpetion { }  // this will create UnChecked exception

**Q8. What are the important methods of Java Exception Class?**

Exception and all of it’s subclasses doesn’t provide any specific methods and all of the methods are defined in the base class Throwable.

**String getMessage()** – This method returns the message String of Throwable and the message can be provided while creating the exception through it’s constructor.

**String getLocalizedMessage(**) – This method is provided so that subclasses can override it to provide locale specific message to the calling program. Throwable class implementation of this method simply use getMessage() method to return the exception message.

**Synchronized Throwable getCause()** – This method returns the cause of the exception or null id the cause is unknown.

**String toString()** – This method returns the information about Throwable in String format, the returned String contains the name of Throwable class and localized message.

**void printStackTrace()** – This method prints the stack trace information to the standard error stream, this method is overloaded and we can pass PrintStream or PrintWriter as argument to write the stack trace information to the file or stream.

**Q9. What are the differences between processes and threads?**

|  |  |  |
| --- | --- | --- |
|  | **Process** | **Thread** |
| **Definition** | An executing instance of a program is called a process. | A thread is a subset of the process. |
| **Communication** | Processes must use inter-process communication to communicate with sibling processes. | Threads can directly communicate with other threads of its process. |
| **Control** | Processes can only exercise control over child processes. | Threads can exercise considerable control over threads of the same process. |
| **Changes** | Any change in the parent process does not affect child processes. | Any change in the main thread may affect the behavior of the other threads of the process. |
| **Memory** | Run in separate memory spaces. | Run in shared memory spaces. |
| **Controlled by** | Process is controlled by the operating system. | Threads are controlled by programmer in a program. |
| **Dependence** | Processes are independent. | Threads are dependent. |

**Q10. What is a finally block? Is there a case when finally will not execute?**

Finally block is a block which always execute a set of statements. It is always associated with a try block regardless of any exception that occurs or not.   
Yes, finally will not be executed if the program exits either by calling System.exit() or by causing a fatal error that causes the process to abort.

**Q11. What is synchronization?**

Synchronization refers to multi-threading. A synchronized block of code can be executed by only one thread at a time. As Java supports execution of multiple threads, two or more threads may access the same fields or objects. Synchronization is a process which keeps all concurrent threads in execution to be in sync. Synchronization avoids memory consistency errors caused due to inconsistent view of shared memory. When a method is declared as synchronized the thread holds the monitor for that method’s object. If another thread is executing the synchronized method the thread is blocked until that thread releases the monitor.

**Q12. Can we write multiple catch blocks under single try block?**

Yes we can have multiple catch blocks under single try block but the approach should be from specific to general.

|  |
| --- |
| public class Example {  public static void main(String args[]) {  try {  int a[]= new int[10];  a[10]= 10/0;  }  catch(ArithmeticException e)  {  System.out.println("Arithmetic exception in first catch block");  }  catch(ArrayIndexOutOfBoundsException e)  {  System.out.println("Array index out of bounds in second catch block");  }  catch(Exception e)  {  System.out.println("Any exception in third catch block");  }  } |