## **Basic Java Interview Questions**

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

|  |  |  |
| --- | --- | --- |
| **JDK vs JRE vs JVM** | | |
| **JDK** | **JRE** | **JVM** |
| It stands for Java Development Kit. | It stands for Java Runtime Environment. | It stands for Java Virtual Machine. |
| It is the tool necessary to compile, document and package Java programs. | JRE refers to a runtime environment in which Java bytecode can be executed. | It is an abstract machine. It is a specification that provides a run-time environment in which Java bytecode can be executed. |
| It contains JRE + development tools. | It’s an implementation of the JVM which physically exists. | JVM follows three notations: Specification, **Implementation,**and **Runtime Instance**. |

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

main() in Java is the entry point for any Java program. It is always written as **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. main() is made static in Java so that it can be accessed without creating the instance of a Class. In case, main is not made static then the compiler will throw an error as **main**() is called by the JVM before any objects are made and only static methods can be directly invoked via the 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?****

Java is called platform independent 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 data types such as boolean, byte, char, int, float, double, long, short which are not objects.

### ****Q5. What are wrapper classes in Java?****

Wrapper classes convert 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. Refer to the below image which displays different primitive type, wrapper class and constructor argument.

### ****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:

1. **Default Constructor:** In Java, a default constructor is the one which does not take any inputs. In other words, default constructors are the no argument constructors which will be created by default in case you no other constructor is defined by the user. Its main purpose is to initialize the instance variables with the default values. Also, it is majorly used for object creation.
2. **Parameterized Constructor:** The parameterized constructor in Java, is the constructor which is capable of initializing the instance variables with the provided values. In other words, the constructors which take the arguments are called parameterized constructors.

### ****Q7. What is singleton class in Java 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 in Java?****

|  |  |
| --- | --- |
| **ArrayList** | **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. | Vector can use both Enumeration and Iterator for traversing. |

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

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.

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

The major difference between Heap and Stack memory are:

|  |  |  |
| --- | --- | --- |
| **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 the generation associated with 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. |

### ****Q11. What is a package in Java? List down various advantages of packages.****

Packages in Java, are the collection of related classes and interfaces which are bundled together. By using packages, developers can easily modularize the code and optimize its reuse. Also, the code within the packages can be imported by other classes and reused. Below I have listed down a few of its advantages:

* Packages help in avoiding name clashes
* They provide easier access control on the code
* Packages can also contain hidden classes which are not visible to the outer classes and only used within the package
* Creates a proper hierarchical structure which makes it easier to locate the related classes

### ****Q12. Why pointers are not used in Java?****

Java doesn’t use pointers because they are unsafe and increases the complexity of the program. Since, Java is known for its simplicity of code, adding the concept of pointers will be contradicting. Moreover, since JVM is responsible for implicit memory allocation, thus in order to avoid direct access to memory by the user,  pointers are discouraged in Java.

### ****Q13. What is JIT compiler in Java?****

JIT stands for Just-In-Time compiler in Java. It is a program that helps in converting the Java bytecode into instructions that are sent directly to the processor. By default, the JIT compiler is enabled in Java and is activated whenever a Java method is invoked. The JIT compiler then compiles the bytecode of the invoked method into native machine code, compiling it “just in time” to execute. Once the method has been compiled, the JVM summons the compiled code of that method directly rather than interpreting it. This is why it is often responsible for the performance optimization of Java applications at the run time.

### ****Q14. What are access modifiers in Java?****

In Java, access modifiers are special keywords which are used to restrict the access of a class, constructor, data member and method in another class. Java supports four types of access modifiers:

1. Default
2. Private
3. Protected
4. Public

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Modifier** | **Default** | **Private** | **Protected** | **Public** |
| Same class | YES | YES | YES | YES |
| Same Package subclass | YES | NO | YES | YES |
| Same Package non-subclass | YES | NO | YES | YES |
| Different package subclass | NO | NO | YES | YES |
| Different package non-subclass | NO | NO | NO | YES |

### ****Q15. Define a Java Class.****

A class in Java is a blueprint which includes all your data.  A class contains fields (variables) and methods to describe the behavior of an object. Let’s have a look at the syntax of a class.

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

### ****Q16. What is an object in Java and how is it created?****

An object is a real-world entity that has a state and behavior. An object has three characteristics:

1. State
2. Behavior
3. Identity

An object is created using the ‘new’ keyword. For example:

ClassName obj = new ClassName();

### ****Q17. What is Object Oriented Programming?****

Object-oriented programming or popularly known as OOPs is a programming model or approach where the programs are organized around objects rather than logic and functions. In other words, OOP mainly focuses on the objects that are required to be manipulated instead of logic. This approach is ideal for the programs large and complex codes and needs to be actively updated or maintained.

### ****Q18. What are the main concepts of OOPs in Java?****

Object-Oriented Programming or OOPs is a programming style that is associated with concepts like:

1. Inheritance: Inheritance is a process where one class acquires the properties of another.
2. Encapsulation: Encapsulation in Java is a mechanism of wrapping up the data and code together as a single unit.
3. Abstraction: Abstraction is the methodology of hiding the implementation details from the user and only providing the functionality to the users.
4. Polymorphism: Polymorphism is the ability of a variable, function or object to take multiple forms.

### ****Q19. What is the difference between a local variable and an instance variable?****

In Java, a **local variable** is typically used inside a method, constructor, or a **block** and has only local scope. Thus, this variable can be used only within the scope of a block. The best benefit of having a local variable is that other methods in the class won’t be even aware of that variable.

#### **Example**

|  |
| --- |
| if(x > 100)  {  String test = "Edureka";  } |

Whereas, an **instance variable** in Java, is a variable which is bounded to its object itself. These variables are declared within a **class**, but outside a method. Every object of that class will create it’s own copy of the variable while using it. Thus, any changes made to the variable won’t reflect in any other instances of that class and will be bound to that particular instance only.

|  |
| --- |
| class Test{  public String EmpName;  public int empAge;  } |

### ****Q20. Differentiate between the constructors and methods in Java?****

|  |  |
| --- | --- |
| **Methods** | **Constructors** |
| 1. Used to represent the behavior of an object | 1. Used to initialize the state of an object |
| 2. Must have a return type | 2. Do not have any return type |
| 3. Needs to be invoked explicitly | 3. Is invoked implicitly |
| 4. No default method is provided by the compiler | 4. A default constructor is provided by the compiler if the class has none |
| 5. Method name may or may not be same as class name | 5. Constructor name must always be the same as the class name |

### ****Q21. What is final keyword in Java?****

**final**is a special keyword in Java that is used as a non-access modifier. A final variable can be used in different contexts such as:

* **final variable**

When the final keyword is used with a variable then its value can’t be changed once assigned. In case the no value has been assigned to the final variable then using only the class constructor a value can be assigned to it.

#### **final method**

When a method is declared final then it can’t be overridden by the inheriting class.

#### **final class**

When a class is declared as final in Java, it can’t be extended by any subclass class but it can extend other class.

### ****Q22. What is the difference between break and continue statements?****

|  |  |
| --- | --- |
| **break** | **continue** |
| 1. Can be used in switch and loop (for, while, do while) statements | 1. Can be only used with loop statements |
| 2. It causes the switch or loop statements to terminate the moment it is executed | 2. It doesn’t terminate the loop but causes the loop to jump to the next iteration |
| 3. It terminates the innermost enclosing loop or switch immediately | 3. A continue within a loop nested with a switch will cause the next loop iteration to execute |

***Example break:***

|  |
| --- |
| for (int i = 0; i < 5; i++)  {  if (i == 3)  {  break;  }  System.out.println(i);  } |

***Example continue:***

|  |
| --- |
| for (int i = 0; i < 5; i++)  {  if(i == 2)  {  continue;  }  System.out.println(i);  } |

### ****Q23.What is an infinite loop in Java? Explain with an example.****

An infinite loop is an instruction sequence in Java that loops endlessly when a functional exit isn’t met. This type of loop can be the result of a programming error or may also be a deliberate action based on the application behavior. An infinite loop will terminate automatically once the application exits.

For example:

|  |
| --- |
| public class InfiniteForLoopDemo  {  public static void main(String[] arg) {  for(;;)  System.out.println("Welcome to Edureka!");  // To terminate this program press ctrl + c in the console.  }  } |

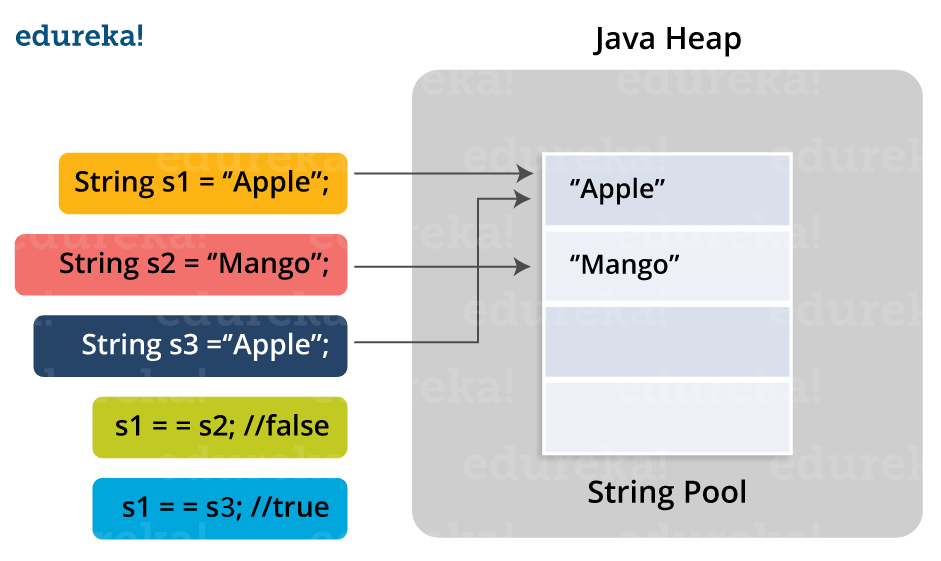
### ****Q24. What is the difference between this() and super() in Java?****

In Java, super() and this(), both are special keywords that are used to call the constructor.

|  |  |
| --- | --- |
| **this()** | **super()** |
| 1. this() represents the current instance of a class | 1. super() represents the current instance of a parent/base class |
| 2. Used to call the default constructor of the same class | 2. Used to call the default constructor of the parent/base class |
| 3. Used to access methods of the current class | 3. Used to access methods of the base class |
| 4.  Used for pointing the current class instance | 4. Used for pointing the superclass instance |
| 5. Must be the first line of a block | 5. Must be the first line of a block |

### ****Q25. What is Java String Pool?****

Java String pool refers to a 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 the same reference is returned to the variable else new object will be created in the String pool and the respective reference will be returned.

****

### ****Q26. Differentiate between static and non-static methods in Java.****

|  |  |
| --- | --- |
| **Static Method** | **Non-Static Method** |
| 1. *The static* keyword must be used before the method name | 1. No need to use the static keyword before the method name |
| 2. It is called using the class (className.methodName) | 2. It is can be called like any general method |
| 3. They can’t access any non-static instance variables or methods | 3. It can access any static method and any static variable without creating an instance of the class |

### ****Q27. What is constructor chaining in Java?****

In Java, constructor chaining is the process of calling one constructor from another with respect to the current object. Constructor chaining is possible only through legacy where a subclass constructor is responsible for invoking the superclass’ constructor first. There could be any number of classes in the constructor chain. Constructor chaining can be achieved in two ways:

1. Within the same class using this()
2. From base class using super()

**Q28. Difference between String, StringBuilder, and StringBuffer.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Factor** | **String** | **StringBuilder** | **StringBuffer** |
| Storage Area | Constant String Pool | Heap Area | Heap Area |
| Mutability | Immutable | Mutable | Mutable |
| Thread Safety | Yes | No | Yes |
| Performance | Fast | More efficient | Less efficient |

### ****Q29. What is a classloader in Java?****

The **Java ClassLoader** is a subset of JVM (Java Virtual Machine) that is responsible for loading the class files. Whenever a Java program is executed it is first loaded by the classloader. Java provides three built-in classloaders:

1. Bootstrap ClassLoader
2. Extension ClassLoader
3. System/Application ClassLoader

### ****Q30. Why Java Strings are immutable in nature?****

In Java, string objects are immutable in nature which simply means once the String object is created its state cannot be modified. Whenever you try to update the value of that object instead of updating the values of that particular object, Java creates a new string object. Java String objects are immutable as String objects are generally cached in the String pool. Since String literals are usually shared between multiple clients, action from one client might affect the rest. It enhances security, caching, synchronization, and performance of the application.

### ****Q31. What is the difference between an array and an array list?****

|  |  |
| --- | --- |
| **Array** | **ArrayList** |
| Cannot contain values of different data types | Can contain values of different data types. |
| Size must be defined at the time of declaration | Size can be dynamically changed |
| Need to specify the index in order to add data | No need to specify the index |
| Arrays are not type parameterized | Arraylists are type |
| Arrays can contain primitive data types as well as objects | Arraylists can contain only objects, no primitive data types are allowed |

### ****Q32. What is a Map in Java?****

In Java, Map is an interface of Util package which maps unique keys to values. The Map interface is not a subset of the main Collection interface and thus it behaves little different from the other collection types. Below are a few of the characteristics of Map interface:

1. Map doesn’t contain duplicate keys.
2. Each key can map at max one value.

### ****Q33. What is collection class in Java? List down its methods and interfaces.****

In Java, the collection is a framework that acts as an architecture for storing and manipulating a group of objects. Using Collections you can perform various tasks like searching, sorting, insertion, manipulation, deletion, etc. Java collection framework includes the following:

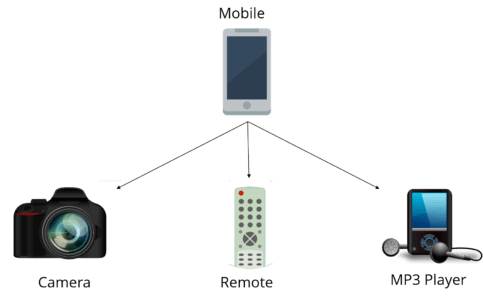
* Interfaces
* Classes
* Methods

The below image shows the complete hierarchy of the Java Collection.



## **OOPS Java Interview Questions**

### ****Q1. What is Polymorphism?****

Polymorphism is briefly described as “one interface, many implementations”. Polymorphism is a characteristic of being able to assign a different meaning or usage to something in different contexts – specifically, to allow an entity such as a variable, a function, or an object to have more than one form. There are two types of polymorphism:

1. Compile time polymorphism
2. Run time polymorphism

Compile time polymorphism is method overloading whereas Runtime time polymorphism is done using inheritance and interface.

### ****Q2. What is runtime polymorphism or dynamic method dispatch?****

In Java, runtime polymorphism or dynamic method dispatch is a process in which a call to an overridden method is resolved at runtime rather than at compile-time. In this process, an overridden method is called through the reference variable of a superclass. Let’s take a look at the example below to understand it better.

|  |
| --- |
| class Car {  void run()  {  System.out.println(“car is running”);  }  }  class Audi extends Car {  void run()  {  System.out.prinltn(“Audi is running safely with 100km”);  }  public static void main(String args[])  {  Car b= new Audi();    //upcasting  b.run();  }  } |

### ****Q3. What is abstraction in Java?****

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. Thus you can say that abstraction in Java is the process of hiding the implementation details from the user and revealing only the functionality to them. Abstraction can be achieved in two ways:

1. **Abstract Classes** (0-100% of abstraction can be achieved)
2. **Interfaces** (100% of abstraction can be achieved)

### ****Q4. What do you mean by an interface in Java?****

An 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. Thus, interface basically is a group of related methods with empty bodies. Example:

public interface Animal {

  public void eat();

  public void sleep();

  public void run();

}

### ****Q5. 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 the case of an 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 the corresponding method in the actual class |

**Q6. What is inheritance in Java?**

Inheritance in Java is the 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. Inheritance is performed between two types of classes:

1. Parent class (Super or Base class)
2. Child class (Subclass or Derived class)

A class which inherits the properties is known as Child Class whereas a class whose properties are inherited is known as Parent class.

### ****Q7. What are the different types of inheritance in Java?****

Java supports four types of inheritance which are:

1. **Single Inheritance:** In single inheritance, one class inherits the properties of another i.e there will be only one parent as well as one child class.
2. **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.
3. **Hierarchical Inheritance:**When a class has more than one child classes (subclasses) or in other words, more than one child classes have the same parent class, then such kind of inheritance is known as hierarchical.
4. **Hybrid Inheritance:**Hybrid inheritance is a combination of two*or more types* of inheritance.

### ****Q8. What is method overloading and method overriding?****

#### **Method Overloading :**

* In Method Overloading, Methods of the same class shares the same name but each method must have a different number of parameters or parameters having different types and order.
* Method Overloading is to “add” or “extend” more to the method’s behavior.
* It is a compile-time polymorphism.
* The methods must have a different signature.
* It may or may not need inheritance in Method Overloading.

Let’s take a look at the example below to understand it better.

|  |
| --- |
| 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));  }} |

#### **Method Overriding:**

* In Method Overriding, the subclass has the same method with the same name and exactly the same number and type of parameters and same return type as a superclass.
* Method Overriding is to “Change” existing behavior of the method.
* It is a run time polymorphism.
* The methods must have the same signature.
* It always requires inheritance in Method Overriding.

Let’s take a look at the example below to understand it better.

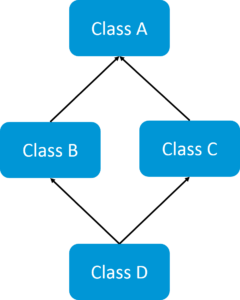
|  |
| --- |
| class Car {  void run(){  System.out.println(“car is running”);  }  Class Audi extends Car{  void run()  {  System.out.prinltn("Audi is running safely with 100km");  }  public static void main( String args[])  {  Car b=new Audi();  b.run();  }  } |

### ****Q9. 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 the same return type and same method arguments in child class then it will hide the superclass method; this is known as method hiding. Similarly, you cannot override a private method in subclass because it’s not accessible there. What you can do is create another private method with the same name in the child class. Let’s take a look at the example below to understand it better.

|  |
| --- |
| 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();  }  } |

### ****Q10. What is multiple inheritance? Is it supported by Java?****



If a child class inherits the property from multiple classes is known as multiple inheritance. Java does not allow to extend multiple classes.

The problem with multiple inheritance is that if multiple parent classes have the same method name, then at runtime it becomes difficult for the compiler to decide which method to execute from the child class.

Therefore, Java doesn’t support multiple inheritance. The problem is commonly referred to as Diamond Problem.

In case you are facing any challenges with these java interview questions, please comment on your problems in the section below.

### ****Q11. What is encapsulation in Java?****

Encapsulation is a mechanism where you bind your data(variables) and code(methods) together as a single unit. Here, the data is hidden from the outer world and can be accessed only via current class methods. This helps in protecting the data from any unnecessary modification. 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 values of the variables.

### ****Q12. What is an association?****

Association is a relationship where all object have their own lifecycle and there is no owner. Let’s take the 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 relationships can be one to one, one to many, many to one and many to many.

### ****Q13. What do you mean by aggregation?****

An aggregation is a specialized form of Association where all object has their own lifecycle but there is ownership and child object can not belong to another parent object. Let’s take an example of Department and teacher. A single teacher can not belong to multiple departments, but if we delete the department teacher object will not destroy.

### ****Q14. What is composition in Java?****

Composition is again a 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 a relationship between House and rooms. House can contain multiple rooms there is no independent life of room and any room can not belongs to two different houses if we delete the house room will automatically delete.

**Q15. What is a marker interface?**

A Marker interface can be defined as the interface having no data member and member functions. In simpler terms, an empty interface is called the Marker interface. The most common examples of Marker interface in Java are Serializable, Cloneable etc. The marker interface can be declared as follows.

|  |
| --- |
| public interface Serializable{  } |

**Q16. What is object cloning in Java?**

Object cloning in Java is the process of creating an exact copy of an object. It basically means the ability to create an object with a similar state as the original object. To achieve this, Java provides a method **clone()** to make use of this functionality. This method creates a new instance of the class of the current object and then initializes all its fields with the exact same contents of corresponding fields. To object clone(), the marker interface **java.lang.Cloneable**must be implemented to avoid any runtime exceptions. One thing you must note is Object clone() is a protected method, thus you need to override it.

### ****Q17. What is a copy constructor in Java?****

Copy constructor is a member function that is used to initialize an object using another object of the same class. Though there is no need for copy constructor in Java since all objects are passed by reference. Moreover, Java does not even support automatic pass-by-value.

### ****Q18. What is a constructor overloading in Java?****

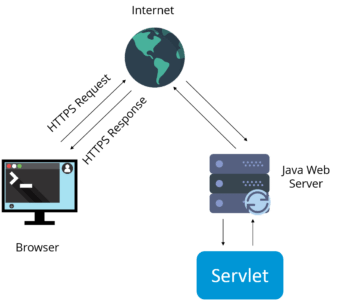
In Java, constructor overloading is a technique of adding any number of constructors to a class each having a different parameter list. The compiler uses the number of parameters and their types in the list to differentiate the overloaded constructors.

|  |
| --- |
| class Demo  {  int i;  public Demo(int a)  {  i=k;  }  public Demo(int a, int b)  {  //body  }  } |

## **Servlets – Java Interview Questions**

### ****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 thats 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?****

RequestDispatcher 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:

1.void forward()

2.void include()

## ForwardMethod - Java Interview Questions - Edureka

## IncludeMethod - Java Interview Questions - Edureka

### ****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?****

There are 5 stages in the lifecycle of a servlet:

****

1. Servlet is loaded
2. Servlet is instantiated
3. Servlet is initialized
4. Service the request
5. 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?****

The difference between ServletContext and ServletConfig in Servlets JSP is in below tabular format.

|  |  |
| --- | --- |
| **ServletConfig** | **ServletContext** |
| Servlet config object represent single servlet | It represent whole web application running on particular JVM and common for all the servlet |
| Its like local parameter associated with particular servlet | Its 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:

1. User Authentication
2. HTML Hidden Field
3. Cookies
4. URL Rewriting
5. Session Management API

## SessionManagement - Java Interview Questions - Edureka

In case you are facing any challenges with these java interview questions, please comment your problems in the section below. Apart from this Java Interview Questions Blog, if you want to get trained from professionals on this technology, you can opt for a structured training from edureka! Click below to know more.

## **JDBC – Java Interview Questions**

### ****1. 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:

1. JDBC-ODBC bridge driver
2. Native-API driver (partially java driver)
3. Network Protocol driver (fully java driver)
4. Thin driver (fully java driver)

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

* Registering the driver class
* Creating connection
* Creating statement
* Executing queries
* Closing connection

### ****3. 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.

### ****4. 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.

### ****5. 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.

## ConnectionInterface - Java Interview Questions - Edureka

In case you are facing any challenges with these java interview questions, please comment on your problems in the section below.

### ****6.  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.

### ****7. What is JDBC ResultSetMetaData interface?****

The ResultSetMetaData interface returns the information of table such as total number of columns, column name, column type etc.

### ****8. 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.

### ****9. 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.

### ****10. 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 can not 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.

### ****Q11. What do you understand by JDBC Statements?****

JDBC statements are basically the statements which are used to send SQL commands to the database and retrieve data back from the database. Various methods like execute(), executeUpdate(), executeQuery, etc. are provided by JDBC to interact with the database.

JDBC supports 3 types of statements:

1. Statement: Used for general purpose access to the database and executes a static SQL query at runtime.
2. PreparedStatement: Used to provide input parameters to the query during execution.
3. CallableStatement: Used to access the database stored procedures and helps in accepting runtime parameters.

## **Spring Framework – Java Interview Questions**

### ****Q1. What is Spring?****

Wikipedia defines 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.****

Some of the important Spring Framework modules are:

* 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.

### SpringFramework - Java Interview Questions - Edureka

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

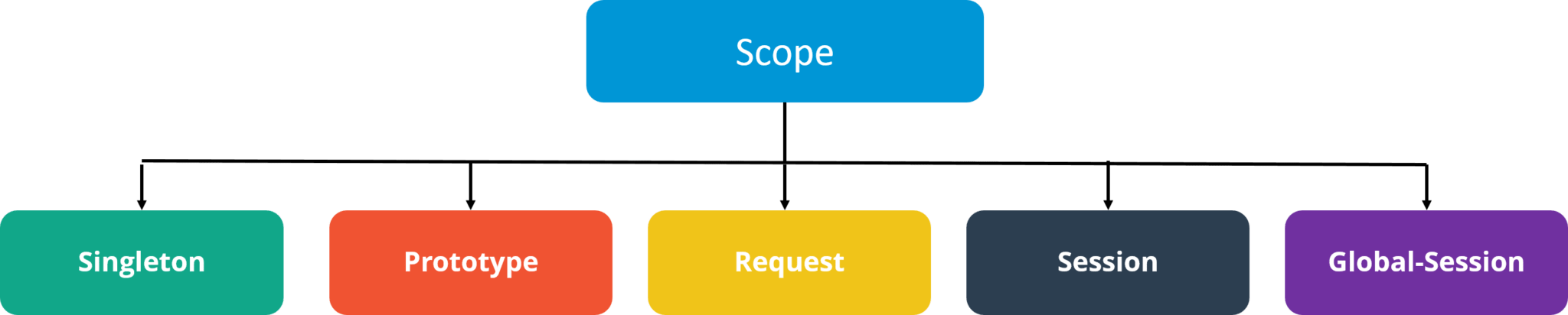
The important annotations are:

* @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.

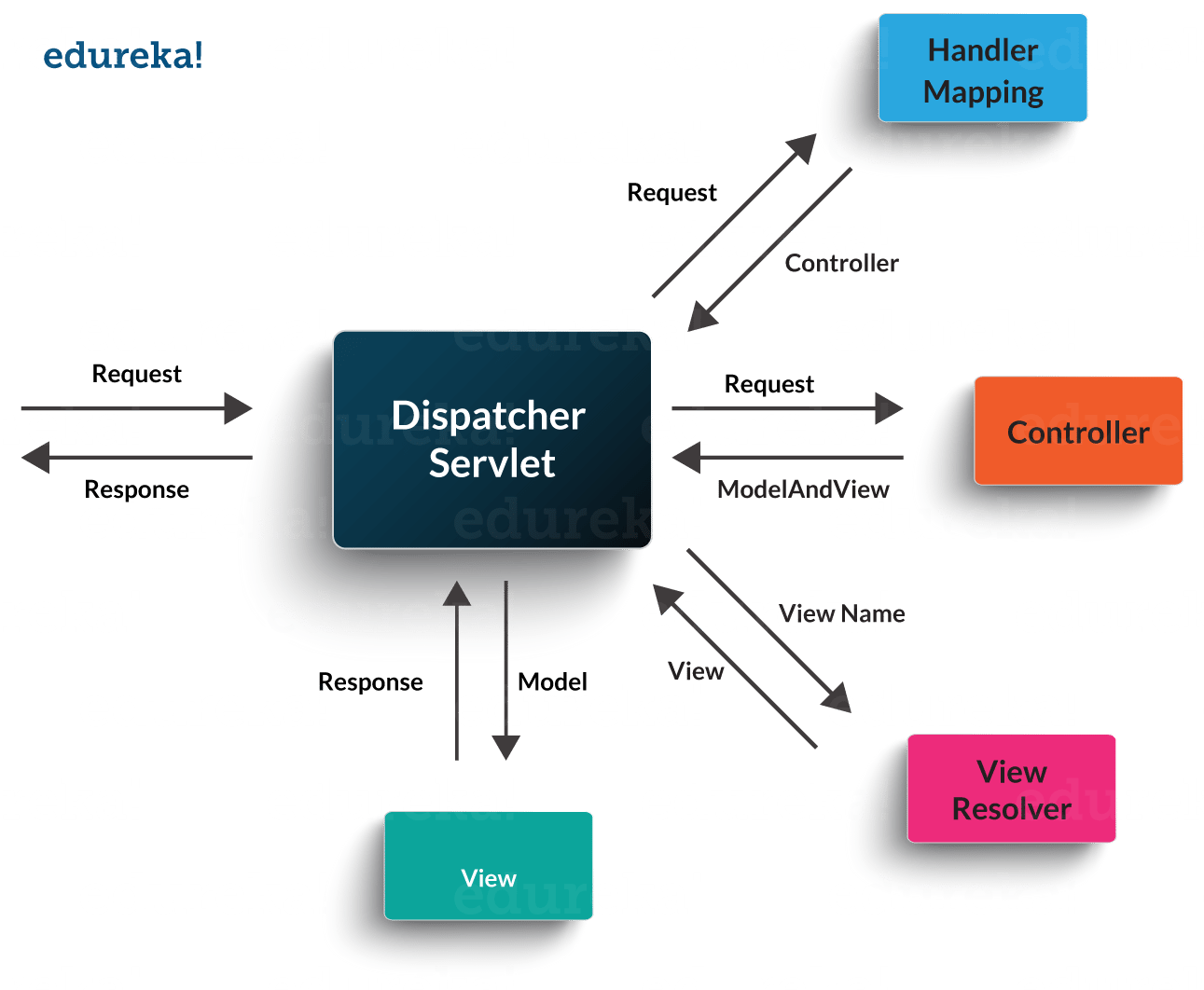


* **Singleton**: Only one instance of the bean will be created for each container. This is the default scope for the spring beans. While using this scope, make sure spring bean doesn’t have shared instance variables otherwise it might lead to data inconsistency issues because it’s not thread-safe.
*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

In case you are facing any challenges with these java interview questions, please comment on your problems in the section below.

### ****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.

### ContextLoader - Java Interview Questions - Edureka

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

|  |  |  |
| --- | --- | --- |
| **No.** | **Constructor Injection** | **Setter Injection** |
| 1) | No Partial Injection | Partial Injection |
| 2) | Doesn’t override the setter property | Overrides the constructor property if both are defined. |
| 3) | Creates a new instance if any modification occurs | Doesn’t create a new instance if you change the property value |
| 4) | Better for too many properties | Better for a 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.

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

The autowiring modes are given below:

|  |  |  |
| --- | --- | --- |
| **No.** | **Mode** | **Description** |
| 1) | no | this is the default mode, it means autowiring is not enabled. |
| 2) | byName | Injects the bean based on the property name. It uses setter method. |
| 3) | byType | Injects the bean based on the property type. It uses setter method. |
| 4) | constructor | It injects the bean using constructor |

### ****Q8. How to handle exceptions in Spring MVC Framework?****

Spring MVC Framework provides the 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?****

Some of the Spring annotations that I have used in my project are:

**@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 the 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 using HibernateTemplate 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.

### ****Q11. Name the types of transaction management that Spring supports.****

Two types of transaction management are supported by Spring. They are:

1. **Programmatic transaction management:**In this, the transaction is managed with the help of programming. It provides you extreme flexibility, but it is very difficult to maintain.
2. **Declarative transaction management:**In this, transaction management is separated from the business code. Only annotations or XML based configurations are used to manage the transactions.

## **Hibernate – Java Interview Questions**

### ****1. 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 a framework for mapping application domain objects to the relational database tables and vice versa.

Hibernate provides a reference implementation of Java Persistence API, that makes it a great choice as ORM tool with benefits of loose coupling. We can use the 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.

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

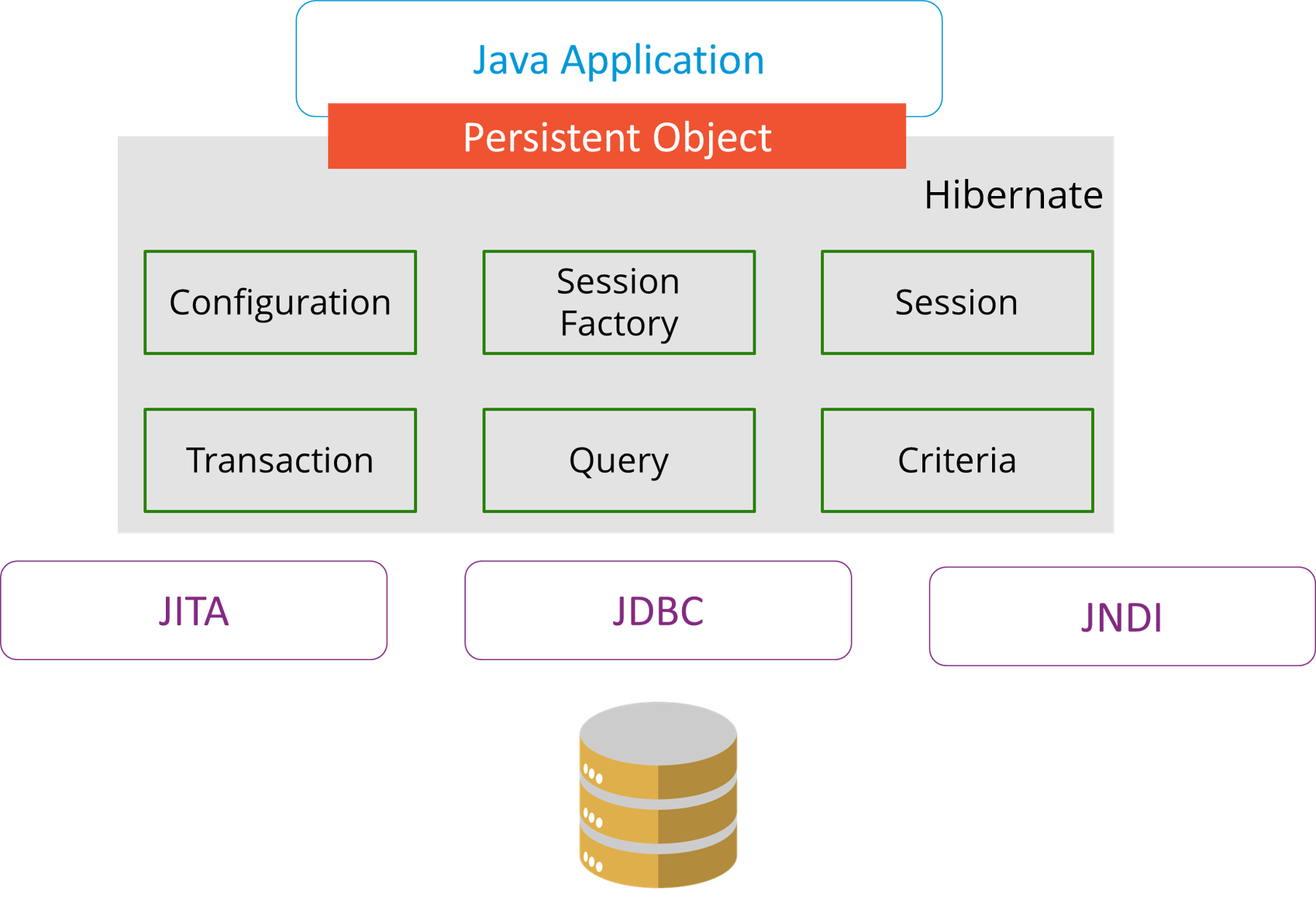
Some of the important benefits of using hibernate framework are:

1. Hibernate eliminates all the boiler-plate code that comes with JDBC and takes care of managing resources, so we can focus on business logic.
2. Hibernate framework provides support for XML as well as JPA annotations, that makes our code implementation independent.
3. 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.
4. 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 documentation and help is easily available in forums.
5. 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.
6. Hibernate supports lazy initialization using proxy objects and perform actual database queries only when it’s required.
7. Hibernate cache helps us in getting better performance.
8. For database vendor specific feature, hibernate is suitable because we can also execute native sql queries.

Overall hibernate is the best choice in current market for ORM tool, it contains all the features that you will ever need in an ORM tool.

### ****3. Explain Hibernate architecture.****

Hibernate has a layered architecture which helps the user to operate without having to know the underlying APIs. Hibernate makes use of the database and configuration data to provide persistence services (and persistent objects) to the application. It includes many objects such as persistent object, session factory, transaction factory, connection factory, session, transaction etc.



The Hibernate architecture is categorized in four layers.

* Java application layer
* Hibernate framework layer
* Backhand API layer
* Database layer

### ****4. What are the differences between get and load methods?****

The differences between get() and load() methods are given below.

|  |  |  |
| --- | --- | --- |
| **No.** | **get()** | **load()** |
| 1) | Returns null if object is not found. | Throws ObjectNotFoundException if an object is not found. |
| 2) | get() method always hit the database. | load() method doesn’t hit the database. |
| 3) | It returns a real object, not a proxy. | It returns a proxy object. |
| 4) | It should be used if you are not sure about the existence of instance. | It should be used if you are sure that the instance exists. |

### ****5. What are the advantages of Hibernate over JDBC?****

Some of the important advantages of Hibernate framework over JDBC are:

1. Hibernate removes a lot of boiler-plate code that comes with JDBC API, the code looks cleaner and readable.
2. Hibernate supports inheritance, associations, and collections. These features are not present with JDBC API.
3. 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.
4. 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 HibernateException un-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.
5. Hibernate Query Language (HQL) is more object-oriented and close to Java programming language. For JDBC, we need to write native SQL queries.
6. Hibernate supports caching that is better for performance, JDBC queries are not cached hence performance is low.
7. Hibernate provides option through which we can create database tables too, for JDBC tables must exist in the database.
8. Hibernate configuration helps us in using JDBC like connection as well as JNDI DataSource for the connection pool. This is a very important feature in enterprise application and completely missing in JDBC API.
9. Hibernate supports JPA annotations, so the code is independent of the implementation and easily replaceable with other ORM tools. JDBC code is very tightly coupled with the application.

In case you are facing any challenges with these Java interview questions, please comment on your problems in the section below. Apart from this Java Interview Questions Blog, if you want to get trained from professionals on this technology, you can opt for structured training from edureka!

## **JSP – Java Interview Questions**

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

|  |  |
| --- | --- |
| **Methods** | **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. |

### ****2. 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 |

### ****3. 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. |

### ****4. 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  
%**>**

### ****5. What are the different tags provided in JSTL?****

There are 5 type of JSTL tags.

1. core tags
2. sql tags
3. xml tags
4. internationalization tags
5. functions tags

### ****6. How to disable session in JSP?****

1. **<**%@ page session=“false” %**>**

### ****7.  How to delete a Cookie in a JSP?****

The following code explains 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 ); |

### ****8. 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.

### ****9.  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 an HTML template file. It provides the framework for the development of a Web Application.

### ****10. 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.

### ****11. How will you use JSP EL in order to get the HTTP method name?****

Using pageContext JSP EL implicit object you can get the request object reference and make use of the dot operator to retrieve the HTTP method name in the JSP page. The JSP EL code for this purpose will look like ${pageContext.request.method}.

## **Exception and Thread Java Interview Questions**

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

An error is an irrecoverable condition occurring at runtime. Such as OutOfMemory error. These JVM errors you cannot repair them at runtime. Though error can be caught in the 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 (probably by giving the user feedback for entering proper values etc.

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

There are five keywords used to handle exceptions in Java:

1. try
2. catch
3. finally
4. throw
5. 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 do the keywords final, finally, and finalize fulfill?****

#### **Final:**

Final is used to apply restrictions on class, method, and variable. A final class can’t be inherited, final method can’t be overridden and final variable value can’t be changed. Let’s take a look at the example below to understand it better.

|  |
| --- |
| 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 the exception is handled or not. Let’s take a look at the example below to understand it better.

|  |
| --- |
| 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 the object is garbage collected. Let’s take a look at the example below to understand it better.

|  |
| --- |
| 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();  }  } |

### 

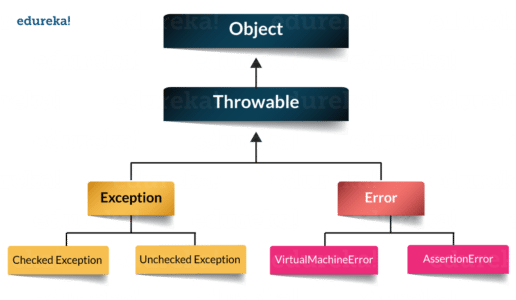
### ****Q5. What are the differences between throw and throws?****

|  |  |
| --- | --- |
| **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 IOException { }             // 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.

1. **String getMessage()** – This method returns the message String of Throwable and the message can be provided while creating the exception through it’s constructor.
2. **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.
3. **Synchronized Throwable getCause()** – This method returns the cause of the exception or null id the cause is unknown.
4. **String toString()** – This method returns the information about Throwable in String format, the returned String contains the name of Throwable class and localized message.
5. **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 an argument to write the stack trace information to the file or stream.

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

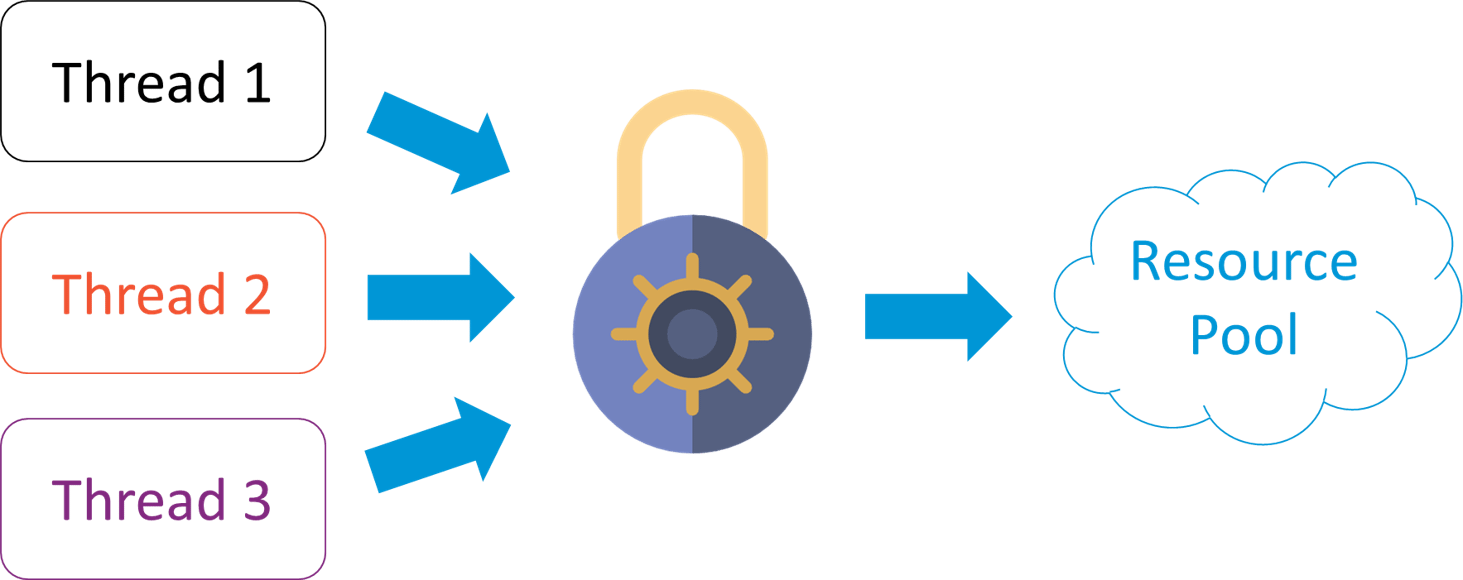
|  |  |  |
| --- | --- | --- |
|  | **Process** | **Thread** |
| **Definition** | An executing instance of a program is called a process. | A thread is a subset of the process. |
| **Communication** | Processes must use inter-process communication to communicate with sibling processes. | Threads can 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 executes 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. Let’s understand this with a programmatic example.

|  |
| --- |
| 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");  }  } |

**Q13. What are the important methods of Java Exception Class?**

Methods are defined in the base class Throwable. Some of the important methods of Java exception class are stated below.

1. **String getMessage()** – This method returns the message String about the exception. The message can be provided through its constructor.
2. **public StackTraceElement[] getStackTrace() –**This method returns an array containing each element on the stack trace. The element at index 0 represents the top of the call stack whereas the last element in the array represents the method at the bottom of the call stack.
3. **Synchronized Throwable getCause()** – This method returns the cause of the exception or null id as represented by a Throwable object.
4. **String toString()** – This method returns the information in String format. The returned String contains the name of Throwable class and localized message.
5. **void printStackTrace()** – This method prints the stack trace information to the standard error stream.

**Q14. What is OutOfMemoryError in Java?**

OutOfMemoryError is the subclass of java.lang.Error which generally occurs when our JVM runs out of memory.

**Q15. What is a Thread?**

A thread is the smallest piece of programmed instructions which can be executed independently by a scheduler. In Java, all the programs will have at least one thread which is known as the main thread. This main thread is created by the JVM when the program starts its execution. The main thread is used to invoke the main() of the program.

**Q16. What are the two ways to create a thread?**

In Java, threads can be created in the following two ways:-

* By implementing the Runnable interface.
* By extending the Thread

**Q17. What are the different types of garbage collectors in Java?**

Garbage collection in Java a program which helps in implicit memory management. Since in Java, using the new keyword you can create objects dynamically, which once created will consume some memory. Once the job is done and there are no more references left to the object, Java using garbage collection destroys the object and relieves the memory occupied by it. Java provides four types of garbage collectors:

* Serial Garbage Collector
* Parallel Garbage Collector
* CMS Garbage Collector
* G1 Garbage Collector

## What is ORM?

ORM stands for Object Relational mapping. It is programming paradigm which is used to persist java objects  to database tables.

## 2. What is Hibernate?

Hibernate is pure ORM tool which is used to save old java objects to database tables. The main goal of hibernate to avoid old JDBC code and focus more on business logic. You need to write less code with it.

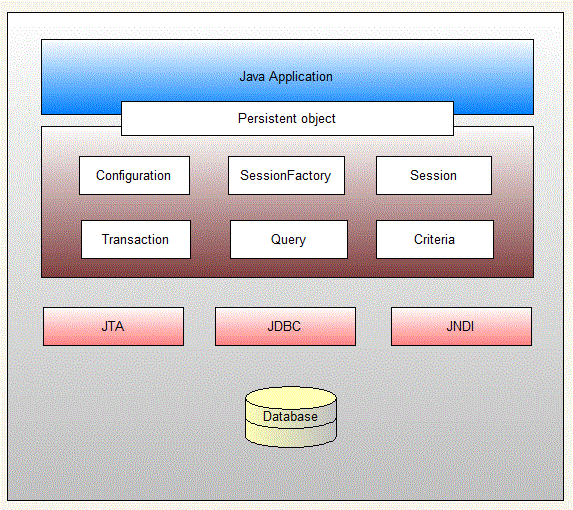
## 3. What are advantages of Hibernate?

Advantages of Hibernate are:

* Lazy Loading
* Caching
* You do not need to maintain JDBC code , Hibernate takes care of it.
* You need to write less code
* It provides high level object oriented API

## 4. Explain architecture of Hibernate?

Following is detail architecture of hibernate with core classes.

[](https://2.bp.blogspot.com/-ak2gCuZuf8I/UQlJ2RcnbTI/AAAAAAAADxk/EWUnClpuY2EfA-NYZrw0Q-uFOJBULZdoQCKgB/s1600/HibernateArchitecture.gif)

Hibernate is like a bridge between java code and relation database and provide object oriented API to deal with JDBC tasks.

## 5. What are some core interfaces of hibernate?

* Session
* SessionFactory
* Configuration
* Transaction
* Query and Criteria interface.

## 6. Explain brief about Session interface used in hibernate?

Session interface is primarily used by hibernate application. Session is light weight,short lived objects which are inexpensive to create and destroy. It allows you to create query objects to retrieve persistent objects.It wraps JDBC connection Factory for Transaction.It holds a mandatory (first-level) cache of persistent objects, used when navigating the object graph or looking up objects by identifier .

## 7. Are session objects thread safe?

Session objects are not thread safe and need to be used in single thread. Every thread should have their own session object and close it once it is done with the work.

## 8. Explain brief about SessionFactory object used in hibernate?

SessionFactory is heavy weight object and it should be created one per database. SessionFactory object is shared by multiple sessions.

## 9. What is Configuration class in hibernate?

Configuration class is used to load required hibernate configuration. It is used to bootstrap hibernate and it is used to locate to hibernate mapping file.

## 10. Explain brief about Criteria API in Hibernate?

Criteria API is a API for retrieving entities by composing Criterion objects also referred as Criterion query.

Criteria API is elegant way for building dynamic queries on the persistence database.

Let’s understand with the help of example.  
You have Employee class with two attributes i.e. name and age.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50 | package org.arpit.java2blog;    /\*  \* This is our model class and it corresponds to Employee table in database  \*/  @Entity  @Table(name="EMPLOYEE")  public class Employee {        @Id      @Column(name="id")      @GeneratedValue(strategy=GenerationType.IDENTITY)      int id;        @Column(name="name")      String name;        @Column(name="age")      int age;        public Employee(String name) {          this.name = name;      }        public int getId() {          return id;      }        public void setId(int id) {          this.id = id;      }        public String getName() {          return name;      }        public void setName(String name) {          this.name = name;      }        public int getAge() {          return age;      }        public void setAge(int age) {          this.age = age;      }  } |

You want to get list of employees whose name start with A and age is greater than 30.

You can write query as below.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | Criteria criteria = session.createCriteria(Employee.class);  criteria.add(Restrictions.like("name","A%");  criteria.add(Restrictions.gt("age",30);  List<Employee> employeeList = criteria.list(); |

## 11. Explain brief about Query interface in Hibernate?

Query interface is object oriented representation of Hibernate Query. You can get query object bu calling Session.createQuery() method.

Here is simple example to execute Native query using Query APIs.

|  |
| --- |
| SQLQuery query = session.createSQLQuery("select name, age from Employee");  List<Object[]> rows = query.list();  for(Object[] row : rows){      Employee e = new Employee();      e.setName(row[0].toString());      e.setAge(Integer.parseInt(row[1].toString()));      System.out.println(e);  } |

## 12. What are differences between openSession and getCurrentSession in hibernate?

|  |  |  |
| --- | --- | --- |
| **Parameter** | **openSession** | **getCurrentSession** |
| Session object | It always create new Session object | It creates a new Session if not exists , else use same session which is in current hibernate context |
| Flush and close | You need to explicitly flush and close session objects | You do not need to flush and close session objects, it will be automatically taken care by Hibernate internally |
| Performance | In single threaded environment , It is slower than getCurrentSession | In single threaded environment , It is faster than getOpenSession |
| Configuration | You do not need to configure any property to call this method | You need to configure additional property “hibernate.current\_session\_context\_class” to call getCurrentSession method, otherwise it will throw exceptions. |

## 13. What are differences between get and load methods in hibernate?

This is one of the most asked hibernate interview questions.

|  |  |  |
| --- | --- | --- |
| **Parameter** | **get** | **load** |
| Database retrieval | It always hits the database | It does not hit database |
| If null | If it does not get the object with id, it returns null | If it does get the object with id, it throws ObjectNotFoundException |
| Proxy | It returns real object | It returns proxy object |
| Use | If you are not sure if object with id exists or not, you can use get | If you are sure about existence of object, you can use load |

## 14. Can you declare Entity class as final in hibernate?

Yes, you can declare entity class as final but it is not considered as a good practice because hibernate uses proxy pattern for lazy initialisation, If you declare it as final then hibernate won’t be able to create sub class and won’t be able to use proxy pattern, so it will limit performance and improvement options.

## 15. Differentiate between first level cache and second level cache?

This is one of most important hibernate interview questions.

|  |  |  |
| --- | --- | --- |
| **Parameter** | **First level Cache** | **Second level Cache** |
| Association | It is associated at Session level. | It is associated at SessionFactory level and is generally exists one per application. |
| Default | It is enabled by default | It is not enabled by default, you need to enable it explicitly |

## 16. What are states of object in hibernate?

**Transient :** If object is in transient state, it means it was never associated with session and just created.

**Persistent :** If object is in persistent state, it means it is associated with session and you just saved or retrieved object from database.

**Detached :**If object is in detached state, it means session is closed and object is no more part of session. If you call merge or update, object goes back to persistent state.

## 17. Does entity class in hibernate require no arg constructor?

Yes, Entity class in hibernate requires no arg constructor because Hibernate use reflection to create instance of entity class and it mandates no arg constructor in Entity class.

## 18. What is HQL?

HQL stands for Hibernate query language. It is very simple, efficient and object oriented query languages which simplifies complex SQL queries. Instead of table, you use object to write queries.

## 19. What are differences between save and saveOrUpdate method of session object?

save(): stores object in database. It generates identifier for tQhe object and returns it. If object already exists in database, it will throw an error.

saveOrUpdate(): SaveOrUpdate method save the object if identifies does not exist. If it exists , it calls update method.

## 20. What are two types of Collections in hibernate?

* Sorted Collection
* Ordered Collection

## 21. What are differences between sorted and ordered collection in hibernate?

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Sorted Collection** | **Ordered Collection** |
| Sorting | Sorted collection uses java’s sorting API to sort the collection. | Ordered Collections uses order by clause while retrieval of objects |
| Default | It is enabled by default | It is not enabled by default, you need to enable it explicitly |

## 22. What is lazy loading in hibernate?

It is used for performance improvement. Lazy loading means when you load parent, child objects won’t get loaded until requested. Infact hibernate loads child objects automatically when you access them. It improves performance as you are not loading whole object at once.

## 23. What is LazyInitializationException in Hibernate? Why do you get it?

This exception generally occurs when you use lazy loading and child objects are accessed after closing session.

That’s all about hibernate interview questions.

## 24. What are different collection types available in hibernate?

There are 5 collection types available in Hibernate for one to many relationship mappings.

* Bag
* Set
* List
* Map
* Array

## 25. How can you log sql queries executed by Hibernate?

You can set hibernate.show\_sql to true for logging sql queries in Hibernate configuration file.

|  |
| --- |
| <property name="hibernate.show\_sql">true</property> |

## 26. Can you execute native sql in hibernate?

Yes, you can use execute native sql with the help of SQLQuery object in Hibernate.

Here is the example to fetch list of employees from database and create corrresponding Employee objects.

|  |
| --- |
| // Crate session object  SessionFactory sf = HibernateUtil.getSessionFactory();  Session session = sf.getCurrentSession();    // Get list of employees with help of SQLQuery.  Transaction tx = session.beginTransaction();  SQLQuery sqlQuery = session.createSQLQuery("select id, name, age from Employee");  List<Object[]> rows = sqlQuery.list();  for(Object[] row : rows){      Employee employee = new Employee();      employee.setId(Long.parseLong(row[0].toString()));      employee.setName(row[1].toString());      employee.setAge(Integer.parseInt(row[2].toString()));      System.out.println(employee);  } |

## 27. What are inheritance mapping strategies are available in Hibernate?

There are three strategies supported by Hibernate. You can use xml files or JPA annotation to implement them.

* Table per Hierarchy
* Table per concrete class
* Table per subclass

## 28. How to make an class immutable in Hibernate?

You can mark class as mutable=false and the class will become immutable.

## 29. What is autamatic dirty checking in Hibernate?

if object is modified in the transaction, then its state will be updated automatically when you committ the transaction.

Here is an example:

|  |
| --- |
| SessionFactory factory = cfg.buildSessionFactory();  Session session = factory.openSession();  Transaction tx=session.beginTransaction();    Employee e1 = (Employee) session.get(Employee.class, Integer.valueOf(10001));    e1.setAge(32);    tx.commit();  session.close(); |

Here, we are updating age of employe after getting employee instance and this is updated automatically when we have committed the transaction.

## 30. What do you understand by Hibernate tuning?

Hibernate tuning is process of optimizing performance of Hibernate application.

Some of the performance tuning strategies are:

* Data Caching
* Session management
* SQL optimization

## 31. What types of joins can you use in Hibernate?

There are multiple ways to use join in Hibernate.

* Using relationships such as one-to-one, one-to-many or many-to-many
* Using joins in native SQL query
* Using joins in HQL

## 32. What is dialect in Hibernate?

Dialect specifies type of database used in Hibernate, so that Hibernate can generate type of SQL statements.

For example:  
Dialect for mysql database: org.hibernate.dialect.MySQL5Dialect Dialect for sqlserver database: org.hibernate.dialect.SQLServer2005Dialect

## 33. Can you share some of the databases supported by Hibernate?

Here are some of the databases supported by Hibernate:

* MySQL
* SQLServer
* PostgreSQL
* FrontBaase
* Oracle
* Sybase SQL Server

## 34. What is Named queries in Hibernate?

Named queries helps you to group HQL/SQL statements at single location. You can refer it by name in code when you want to use them. It helps you to avoid code mess that can happend because of scattered queriest through the project.

|  |
| --- |
| package org.arpit.java2blog;  @NamedQueries(      {          @NamedQuery(          name = "getEmployeeByName",          query = "from Employee e where e.name = :name"          )      }  )    /\*  \* This is our model class and it corresponds to Employee table in database  \*/  @Entity  @Table(name="EMPLOYEE")  public class Employee {        @Id      @Column(name="id")      @GeneratedValue(strategy=GenerationType.IDENTITY)      int id;        @Column(name="name")      String name;        @Column(name="age")      int age;        public Employee(String name) {          this.name = name;      }        public int getId() {          return id;      }        public void setId(int id) {          this.id = id;      }        public String getName() {          return name;      }        public void setName(String name) {          this.name = name;      }        public int getAge() {          return age;      }        public void setAge(int age) {          this.age = age;      }  } |

You can execute the query in main class as below:

|  |
| --- |
| TypedQuery query = session.getNamedQuery("getEmployeeByName");      query.setParameter("name","John");        List<Employee> employees=query.getResultList();      System.out.println(employees); |

## 35. What is Query cache in Hibernate?

If you have queries that run over and over again,with same parameters, then query caching can you performance improvements in the application.

Benifit from caching query results is very limited and dependent on the usage of the application. This is reason Hibernate disables query level cache by default.

To enable, you need to do following:

1. Set hibernate.cache.use\_query\_cache in hibernate config file.

|  |
| --- |
| <property name="hibernate.cache.use\_query\_cache">true</property> |

1. You need to enable query cache for specific queries. Here is an example:

|  |
| --- |
| List<Employees> employee = session.createQuery("from Employee e where e.name = :name")          .setEntity("employee", employee)          .setMaxResults(15)          .setCacheable(true)          .setCacheRegion("employeeReg")          .list(); |

## 36. What are benefits of Criteria API in Hibernate?

Criteria API can be used to fetch entity from database using object oriented apporach.  
Here are the advantages of criteria APIs.

* Criteria API is cleanm nice and Object oriented API.
* Criteria API provides Projection that can be used for aggregate functions like sum(), min() etc.
* You can write more flexible and dynamic queries as compared to HQL.
* It has addOrder() method which can be used for ordering the results.

## 37. What is cascading in Hiberante and can you list types of cascading?

Most of the time, if Entity relationships depend on existence of other entity.  
For example:  
In case of Employee-Address relationship, if Employee is removed from database, then Address does not make sense of its own. So when you remove Employee from database, then its associated Address should also be removed.

You can use Cascading to achieve this. When you perform an action on entity, same action can be performed on associated entity.

Here are the Cascading types supported by Hibernate:

|  |  |
| --- | --- |
| **Cascade Operation** | **Description** |
| ALL | All the operations will be applied to parent entity’s associated entity. All operation includes DETACH, MERGE, PERSIST, REFRESH, REMOVE etc. |
| DETACH | If parent entitiy is detached from context, then the associated entity will also be detached. |
| MERGE | If parent entitiy is merged into the context, then the associated entity will also be merged. |
| PERSIST | If parent entitiy is persisted into the context, then the associated entity will also be persisted. |
| REFRESH | If parent entitiy is refreshed in the curent persistence context, then the associated entity will also be persisted. |
| REMOVE | If parent entitiy is removed from the curent persistence context, then the associated entity will also be removed. |

## 38. How to integrate log4j with Hibernate?

For log4j configuration, you can follow following steps:

1. Add log4j dependencies for maven project. If it is not maven project, then add required log4j jars to classpath.
2. Create log4j.xml or log4j.properties and put in the classpath.
3. Use DOMConfigurator or PropertyConfigurator to configure log4j in static block for standalone application. If you have web application, then you can use ServletContextListener to configure it.

## 39. What is hibernate configuration file?

Hibernate configuration file contains database configurations such as database url, username, password and dialect etc and is used to initialize SessionFactory. It also contains mapping files and entity class details.

## 40. Can you list down important annotations used for Hibernate mapping?

Here are some important annotations that can be used for Hibernate mapping.

@Entity : It is used to define class as entity bean.  
@Table : It is used to define table name in database corrresponding to Entity.  
@Id : It is used to define primary key in the entity bean.  
@Column: It is used to define column properties in database corrresponding to entity bean property.  
@OneToOne, @ManyToOne, @ManyToMany: These annotations are used to define relationships between various entities.  
@JoinColumn : It is used to specify a mapped column for joining an entity association.

Here is an example:  
We are using two entities Country and Capital with one to one relationship.

|  |
| --- |
| package org.arpit.java2blog;  import javax.persistence.Column;  import javax.persistence.Entity;  import javax.persistence.Id;  import javax.persistence.JoinColumn;  import javax.persistence.OneToOne;  import javax.persistence.Table;    @Entity  @Table(name="COUNTRY")  public class Country {        @Id      @Column(name="Country\_Name")      String countryName ;        @OneToOne      @JoinColumn(name="Capital\_Name")      Capital capital;        @Column(name="Country\_Population")      long countryPopulation;        public Country()      {        }        public Country(String countryName, long countryPopulation) {    this.countryName = countryName;    this.countryPopulation = countryPopulation;  }        public long getCountryPopulation() {    return countryPopulation;  }        public void setCountryPopulation(long countryPopulation) {    this.countryPopulation = countryPopulation;  }        public String getCountryName() {          return countryName;      }        public void setCountryName(String countryName) {          this.countryName = countryName;      }        public Capital getCapital() {    return capital;  }        public void setCapital(Capital capital) {    this.capital = capital;  }  } |

Capital.java

|  |
| --- |
| package org.arpit.javapostsforlearning;import javax.persistence.Column;  import javax.persistence.Entity;  import javax.persistence.Id;  import javax.persistence.Table;    @Entity  @Table(name="CAPITAL")  public class Capital {         @Id       @Column(name="Capital\_Name")       String capitalName;         @Column(name="Capital\_Population")       long capitalPopulation;         public Capital()       {         }       public Capital(String capitalName, long capitalPopulation) {        super();        this.capitalName = capitalName;        this.capitalPopulation = capitalPopulation;       }         public String getCapitalName() {        return capitalName;       }         public void setCapitalName(String capitalName) {        this.capitalName = capitalName;       }         public long getCapitalPopulation() {        return capitalPopulation;       }         public void setCapitalPopulation(long capitalPopulation) {        this.capitalPopulation = capitalPopulation;       }    } |

## 41. What are the design patterns used in Hibernate?

Here are some of the design patterns used in Hibernate:

* [Proxy pattern](https://java2blog.com/proxy-design-pattern-in-java) for lazy loading
* Factory pattern in SessionFactory
* Query object for Criteria API
* Data Mapper: A pattern in which layer of matters that flows data between object and databases while keeping them independent of each other
* Domain Model pattern: An object model of the domain that incorporates both data and behavior.

## 42. Given a Customer class, you need to persist customer data in the database with customer ID as primary key. Please list down the changes you need to make?

Here is definition of table CUSTOMER.

|  |
| --- |
| CREATE TABLE CUSTOMER (  id int(11) NOT NULL AUTO\_INCREMENT,  Customer\_Name varchar(255) DEFAULT NULL,  email varchar(255) DEFAULT NULL,  PRIMARY KEY (id)  ) |

Here is the definition of Customer class

|  |
| --- |
| package org.arpit.java2blog.model;    /\*  \* This is our model class and it corresponds to Customer table in database  \*/  =  public class Customer{        int id;        String customerName;        String email;        public Customer() {          super();      }      public Customer(String customerName,String email) {          super();          this.customerName=customerName;          this.email=email;      }      public String getCustomerName() {          return customerName;      }      public void setCustomerName(String customerName) {          this.customerName = customerName;      }      public String getEmail() {          return email;      }      public void setEmail(String email) {          this.email = email;      }      public int getId() {          return id;      }      public void setId(int id) {          this.id = id;      }    } |

Here are the changes that needs to be done:

1. Annotate the Customer class with @Entity to declare it as Hibernate entity
2. Use @Table annotation to map class to table name in databse
3. Use @id and @GeneratedValue annotation to make id as primary key
4. Use @Column to map attribute to corresponding database column

Here is the code:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| package org.arpit.java2blog.model;    import javax.persistence.Column;  import javax.persistence.Entity;  import javax.persistence.GeneratedValue;  import javax.persistence.GenerationType;  import javax.persistence.Id;  import javax.persistence.Table;    /\*  \* This is our model class and it corresponds to Customer table in database  \*/  @Entity  @Table(name="CUSTOMER")  public class Customer{        @Id      @Column(name="id")      @GeneratedValue(strategy=GenerationType.IDENTITY)      int id;        @Column(name="Customer\_Name")      String customerName;        @Column(name="email")      String email;        public Customer() {          super();      }      public Customer(String customerName,String email) {          super();          this.customerName=customerName;          this.email=email;      }      public String getCustomerName() {          return customerName;      }      public void setCustomerName(String customerName) {          this.customerName = customerName;      }      public String getEmail() {          return email;      }      public void setEmail(String email) {          this.email = email;      }      public int getId() {          return id;      }      public void setId(int id) {          this.id = id;      }    }  **Serialization**  Java provides a mechanism, called object serialization where an object can be represented as a sequence of bytes that includes the object's data as well as information about the object's type and the types of data stored in the object.  After a serialized object has been written into a file, it can be read from the file and deserialized that is, the type information and bytes that represent the object and its data can be used to recreate the object in memory.  Most impressive is that the entire process is JVM independent, meaning an object can be serialized on one platform and deserialized on an entirely different platform.  Classes **ObjectInputStream** and **ObjectOutputStream** are high-level streams that contain the methods for serializing and deserializing an object.  The ObjectOutputStream class contains many write methods for writing various data types, but one method in particular stands out −  public final void writeObject(Object x) throws IOException  The above method serializes an Object and sends it to the output stream. Similarly, the ObjectInputStream class contains the following method for deserializing an object −  public final Object readObject() throws IOException, ClassNotFoundException  This method retrieves the next Object out of the stream and deserializes it. The return value is Object, so you will need to cast it to its appropriate data type.  To demonstrate how serialization works in Java, I am going to use the Employee class that we discussed early on in the book. Suppose that we have the following Employee class, which implements the Serializable interface −  Example  public class Employee implements java.io.Serializable {  public String name;  public String address;  public transient int SSN;  public int number;    public void mailCheck() {  System.out.println("Mailing a check to " + name + " " + address);  }  }  Notice that for a class to be serialized successfully, two conditions must be met −   * The class must implement the java.io.Serializable interface. * All of the fields in the class must be serializable. If a field is not serializable, it must be marked **transient**.   If you are curious to know if a Java Standard Class is serializable or not, check the documentation for the class. The test is simple: If the class implements java.io.Serializable, then it is serializable; otherwise, it's not.  Serializing an Object  The ObjectOutputStream class is used to serialize an Object. The following SerializeDemo program instantiates an Employee object and serializes it to a file.  When the program is done executing, a file named employee.ser is created. The program does not generate any output, but study the code and try to determine what the program is doing.  **Note** − When serializing an object to a file, the standard convention in Java is to give the file a **.ser** extension.  Example  import java.io.\*;  public class SerializeDemo {  public static void main(String [] args) {  Employee e = new Employee();  e.name = "Reyan Ali";  e.address = "Phokka Kuan, Ambehta Peer";  e.SSN = 11122333;  e.number = 101;    try {  FileOutputStream fileOut =  new FileOutputStream("/tmp/employee.ser");  ObjectOutputStream out = new ObjectOutputStream(fileOut);  out.writeObject(e);  out.close();  fileOut.close();  System.out.printf("Serialized data is saved in /tmp/employee.ser");  } catch (IOException i) {  i.printStackTrace();  }  }  }  Deserializing an Object  The following DeserializeDemo program deserializes the Employee object created in the SerializeDemo program. Study the program and try to determine its output −  Example  import java.io.\*;  public class DeserializeDemo {  public static void main(String [] args) {  Employee e = null;  try {  FileInputStream fileIn = new FileInputStream("/tmp/employee.ser");  ObjectInputStream in = new ObjectInputStream(fileIn);  e = (Employee) in.readObject();  in.close();  fileIn.close();  } catch (IOException i) {  i.printStackTrace();  return;  } catch (ClassNotFoundException c) {  System.out.println("Employee class not found");  c.printStackTrace();  return;  }    System.out.println("Deserialized Employee...");  System.out.println("Name: " + e.name);  System.out.println("Address: " + e.address);  System.out.println("SSN: " + e.SSN);  System.out.println("Number: " + e.number);  }  }  This will produce the following result −  Output  Deserialized Employee...  Name: Reyan Ali  Address:Phokka Kuan, Ambehta Peer  SSN: 0  Number:101  Here are following important points to be noted −   * The try/catch block tries to catch a ClassNotFoundException, which is declared by the readObject() method. For a JVM to be able to deserialize an object, it must be able to find the bytecode for the class. If the JVM can't find a class during the deserialization of an object, it throws a ClassNotFoundException. * Notice that the return value of readObject() is cast to an Employee reference. * The value of the SSN field was 11122333 when the object was serialized, but because the field is transient, this value was not sent to the output stream. The SSN field of the deserialized Employee object is 0.   **Externalization**  Externalization is used whenever we need to customize serialization mechanism. If a class implements an Externalizable interface then, object serialization will be done using writeExternal() method. Whereas at receiver's end when an Externalizable object is a reconstructed instance will be created using no argument constructor and then the readExternal() method is called.  If a class implements only Serializable interface object serialization will be done using ObjectoutputStream. At the receiver's end, the serializable object is reconstructed using ObjectInputStream.  Below example showcases usage of Externalizable interface.  import java.io.Externalizable;  import java.io.FileInputStream;  import java.io.FileOutputStream;  import java.io.IOException;  import java.io.ObjectInput;  import java.io.ObjectInputStream;  import java.io.ObjectOutput;  import java.io.ObjectOutputStream;  public class Tester {  public static void main(String[] args) {  Employee e = new Employee();  e.name = "Reyan Ali";  e.age = 30;  try (  FileOutputStream fileOut = new FileOutputStream("test.txt");  ObjectOutputStream out = new ObjectOutputStream(fileOut);  ) {  out.writeObject(e);  }catch (IOException i) {  System.out.println(i.getMessage());  }  try (  FileInputStream fileIn = new FileInputStream("test.txt");  ObjectInputStream in = new ObjectInputStream(fileIn);  ) {  e = (Employee)in.readObject();  System.out.println(e.name);  System.out.println(e.age);  } catch (IOException i) {  System.out.println(i.getMessage());  } catch (ClassNotFoundException e1) {  System.out.println(e1.getMessage());  }  }  }  class Employee implements Externalizable {  public Employee(){}  String name;  int age;  public void writeExternal(ObjectOutput out) throws IOException {  out.writeObject(name);  out.writeInt(age);  }  public void readExternal(ObjectInput in) throws IOException,  ClassNotFoundException {  name = (String)in.readObject();  age = in.readInt();  }  }  **Difference Between Serializable and Externalizable**   |  |  | | --- | --- | | **SERIALIZABLE** | **EXTERNALIZABLE** | | Serializable is a marker interface i.e. does not contain any method. | Externalizable interface contains two methods writeExternal() and readExternal() which implementing classes MUST override. | | Serializable interface pass the responsibility of serialization to JVM and it’s default algorithm. | Externalizable provides control of serialization logic to programmer – to write custom logic. | | Mostly, default serialization is easy to implement, but has higher performance cost. | Serialization done using Externalizable, add more responsibility to programmer but often result in better performance. | | It’s hard to analyze and modify class structure because any change may break the serialization. | It’s more easy to analyze and modify class structure because of complete control over serialization logic. | | Default serialization does not call any class constructor. | A public no-arg constructor is required while using Externalizable interface. |   Please note that Externalizable interface is child interface of Serializable i.e. Externalizable extends Serializable. So if any class implements Externalizable interface and override it’s writeExternal() and readExternal() methods then first preference is given to these methods over default serialization mechanism provided by JVM.  **Question** **1**: Why would it be more secure to store sensitive data (such as a password, social security number, etc.) in a character array rather than in a String?  In Java, Strings are immutable and are stored in the String pool. What this means is that, once a String is created, it stays in the pool in memory until being garbage collected. Therefore, even after you’re done processing the string value (e.g., the password), it remains available in memory for an indeterminate period of time thereafter (again, until being garbage collected) which you have no real control over. Therefore, anyone having access to a memory dump can potentially extract the sensitive data and exploit it.  In contrast, if you use a mutable object like a character array, for example, to store the value, you can set it to blank once you are done with it with confidence that it will no longer be retained in memory.  **Question** **2**: What is the ThreadLocal class? How and why would you use it?  A single ThreadLocal instance can store different values for each thread independently. Each thread that accesses the get() or set() method of a ThreadLocal instance is accessing its own, independently initialized copy of the variable. ThreadLocal instances are typically private static fields in classes that wish to associate state with a thread (e.g., a user ID or transaction ID).  **Question** **3**: What is the volatile keyword? How and why would you use it?  In Java, each thread has its own stack, including its own copy of variables it can access. When the thread is created, it copies the value of all accessible variables into its own stack. The volatile keyword basically says to the JVM “Warning, this variable may be modified in another Thread”.  In all versions of Java, the volatile keyword guarantees global ordering on reads and writes to a variable. This implies that every thread accessing a volatile field will read the variable’s current value instead of (potentially) using a cached value.  In Java 5 or later, volatile reads and writes establish a happens-before relationship, much like acquiring and releasing a mutex.  Using volatile may be faster than a lock, but it will not work in some situations. The range of situations in which volatile is effective was expanded in Java 5; in particular, double-checked locking now works correctly.  The volatile keyword is also useful for 64-bit types like long and double since they are written in two operations. Without the volatile keyword you risk stale or invalid values.  One common example for using volatile is for a flag to terminate a thread. If you’ve started a thread, and you want to be able to safely interrupt it from a different thread, you can have the thread periodically check a flag (i.e., to stop it, set the flag to true). By making the flag volatile, you can ensure that the thread that is checking its value will see that it has been set to true without even having to use a synchronized block. For example:  public class Foo extends Thread {  private volatile boolean close = false;  public void run() {  while(!close) {  // do work  }  }  public void close() {  close = true;  // interrupt here if needed  }  }  Question: How can you swap the values of two numeric variables without using any other variables?  You can swap two values a and b without using any other variables as follows:  a = a + b;  b = a - b;  a = a - b;  **Question 4**: What is the Java Classloader? List and explain the purpose of the three types of class loaders.  The Java Classloader is the part of the Java runtime environment that loads classes on demand (lazy loading) into the JVM (Java Virtual Machine). Classes may be loaded from the local file system, a remote file system, or even the web.  When the JVM is started, three class loaders are used: 1. Bootstrap Classloader: Loads core java API file rt.jar from folder. 2. Extension Classloader: Loads jar files from folder. 3. System/Application Classloader: Loads jar files from path specified in the CLASSPATH environment variable.  **Question** **5**: Why String is immutable or final in Java  There are several benefits of String because it’s immutable and final.   * String Pool is possible because String is immutable in java. * It increases security because any hacker can’t change its value and it’s used for storing sensitive information such as database username, password etc. * Since String is immutable, it’s safe to use in multi-threading and we don’t need any synchronization. * Strings are used in java classloader and immutability provides security that correct class is getting loaded by Classloader.   **Question 6**: Why Char array is preferred over String for storing password?  String is immutable in Java and stored in String pool. Once it’s created it stays in the pool until unless garbage collected, so even though we are done with password it’s available in memory for longer duration and there is no way to avoid it. It’s a security risk because anyone having access to memory dump can find the password as clear text.  If we use a char array to store password, we can set it to blank once we are done with it. So we can control for how long it’s available in memory that avoids the security threat with String  **Question 7:** Why Java is not pure Object Oriented language?  Java is not said to be pure object-oriented because it supports primitive types such as int, byte, short, long, etc. I believe it brings simplicity to the language while writing our code. Java could have wrapper objects for the primitive types but just for the representation, they would not have provided any benefit.  As we know, for all the primitive types we have wrapper classes such as Integer, Long etc that provides some additional methods.  **Question 7:** What is difference between user Thread and daemon Thread?  When we create a Thread in java program, it’s known as user thread. A daemon thread runs in background and doesn’t prevent JVM from terminating. When there are no user threads running, JVM shutdown the program and quits. A child thread created from daemon thread is also a daemon thread  **Question 8:** What is context-switching in multi-threading?  Context Switching is the process of storing and restoring of CPU state so that Thread execution can be resumed from the same point at a later point of time. Context Switching is the essential feature for multitasking operating system and support for multi-threaded environment.  **Question 9:** How can we make sure main() is the last thread to finish in Java Program?  We can use Thread join() method to make sure all the threads created by the program is dead before finishing the main function.  **Question 10:** Why thread communication methods wait(), notify() and notifyAll() are in Object class?  In Java every Object has a monitor and wait, notify methods are used to wait for the Object monitor or to notify other threads that Object monitor is free now. There is no monitor on threads in java and synchronization can be used with any Object, that’s why it’s part of Object class so that every class in java has these essential methods for inter thread communication**.**  **Question 11:** Why wait(), notify() and notifyAll() methods have to be called from synchronized method or block?  When a Thread calls wait() on any Object, it must have the monitor on the Object that it will leave and goes in wait state until any other thread call notify() on this Object. Similarly when a thread calls notify() on any Object, it leaves the monitor on the Object and other waiting threads can get the monitor on the Object. Since all these methods require Thread to have the Object monitor, that can be achieved only by synchronization, they need to be called from synchronized method or block.  **Question 12:** Why Thread sleep() and yield() methods are static?  Thread sleep() and yield() methods work on the currently executing thread. So there is no point in invoking these methods on some other threads that are in wait state. That’s why these methods are made static so that when this method is called statically, it works on the current executing thread and avoid confusion to the programmers who might think that they can invoke these methods on some non-running threads  **Question 13:** What is volatile keyword in Java  When we use volatile keyword with a variable, all the threads read it’s value directly from the memory and don’t cache it. This makes sure that the value read is the same as in the memory.  **Question 14:** How to create daemon thread in Java?  Thread class setDaemon(true) can be used to create daemon thread in java. We need to call this method before calling start() method else it will throw IllegalThreadStateException.  **Question 15:** What is ThreadLocal?  Java ThreadLocal is used to create thread-local variables. We know that all threads of an Object share it’s variables, so if the variable is not thread safe, we can use synchronization but if we want to avoid synchronization, we can use ThreadLocal variables.  Every thread has its own ThreadLocal variable and they can use it gets () and set() methods to get the default value or change it’s value local to Thread. ThreadLocal instances are typically private static fields in classes that wish to associate the state with a thread. Check this post for small example program showing ThreadLocal Example.  **Question 16:** What is the difference between JDK and JVM?  Java Development Kit (JDK) is for development purpose and JVM is a part of it to execute the java programs.  JDK provides all the tools, executables and binaries required to compile, debug and execute a Java Program. The execution part is handled by JVM to provide machine independence  **Question 17:** What is the difference between JVM and JRE?  Java Runtime Environment (JRE) is the implementation of JVM. JRE consists of JVM and java binaries and other classes to execute any program successfully. JRE doesn’t contain any development tools like java compiler, debugger, etc. If you want to execute any java program, you should have JRE installed**.**  **Question 18:** What is difference between path and classpath variables?  PATH is an environment variable used by the operating system to locate the executables. That’s why when we install Java or want any executable to be found by OS, we need to add the directory location in the PATH variable.  Classpath is specific to Java and used by java executables to locate class files. We can provide the classpath location while running java application and it can be a directory, ZIP files, JAR files, etc.  **Question 19:** What is the importance of main method in Java?  main() method is the entry point of any standalone java application. The syntax of main method is public static void main(String args[]).  Java’s main method is public and static so that Java runtime can access it without initializing the class. The input parameter is an array of String through which we can pass runtime arguments to the java program.  **Question 20:** Can we have multiple public classes in a java source file?  We can’t have more than one public class in a single java source file. A single source file can have multiple classes that are not public**.**  **Question 21:** What is Java Package and which package is imported by default?  Java package is the mechanism to organize the java classes by grouping them. The grouping logic can be based on functionality or modules based. A java class fully classified name contains package and class name. For example, java.lang.Object is the fully classified name of Object class that is part of java.lang package.  java.lang package is imported by default and we don’t need to import any class from this package explicitly.  **Question 22:** What is finally and finalize in java?  The finally block is used with try-catch to put the code that you want to get executed always, even if an exception is thrown by the try-catch block. finally block is mostly used to release resources created in the try block.  finalize() is a special method in Object class that we can override in our classes. This method gets called by the garbage collector when the object is getting garbage collected. This method is usually overridden to release system resources when the object is garbage collected.  **Question 23:** Can we declare a class as static?  We can’t declare a top-level class as static however an inner class can be declared as static. If the inner class is declared as static, it’s called a static nested class.  The static nested class is the same as any other top-level class and is nested for only packaging convenience.  **Question 24:** What is static import?  If we have to use any static variable or method from other class, usually we import the class and then use the method/variable with class name.  import java.lang.Math;  //inside class  double test = Math.PI \* 5;  We can do the same thing by importing the static method or variable only and then use it in the class as if it belongs to it.  import static java.lang.Math.PI;  //no need to refer class now  double test = PI \* 5;  Use of static import can cause confusion, so it’s better to avoid it. Overuse of static import can make your program unreadable and unmaintainable.  **Question 25:** What is try-with-resources in java?  One of the Java 7 features is the try-with-resources statement for automatic resource management. Before Java 7, there was no auto resource management and we should explicitly close the resource. Usually, it was done in the finally block of a try-catch statement. This approach used to cause memory leaks when we forgot to close the resource.  From Java 7, we can create resources inside try block and use it. Java takes care of closing it as soon as try-catch block gets finished.  package com.journaldev.util;  import java.io.BufferedReader;  import java.io.FileReader;  import java.io.IOException;  public class Java7ResourceManagement {  public static void main(String[] args) {  try (BufferedReader br = new BufferedReader(new FileReader(  "C:\\journaldev.txt"))) {  System.out.println(br.readLine());  } catch (IOException e) {  e.printStackTrace();  }  }  }  **Question 26:** What is multi-catch block in java?  Java 7 one of the improvement was a multi-catch block where we can catch multiple exceptions in a single catch block. This makes our code shorter and cleaner when every catch block has a similar code.  If a catch block handles multiple exceptions, you can separate them using a pipe (|) and in this case, the exception parameter (ex) is final, so you can’t change it.  catch(IOException | SQLException ex){  logger.error(ex);  throw new MyException(ex.getMessage());  }  If a catch block handles multiple exceptions, you can separate them using a pipe (|) and in this case, exception parameter (ex) is final, so you can’t change it. The byte code generated by this feature is smaller and reduce code redundancy.  **Question 27:** What is static block?  Java static block is the group of statements that gets executed when the class is loaded into memory by Java ClassLoader. It is used to initialize static variables of the class. Mostly it’s used to create static resources when class is loaded.  **Question 28:** What are Wrapper classes?  Java wrapper classes are the Object representation of eight primitive types in java. All the wrapper classes in java are immutable and final. Java 5 autoboxing and unboxing allows easy conversion between primitive types and their corresponding wrapper classes**.**  **Question 29:** What is Enum in Java?  Enum was introduced in Java 1.5 as a new type whose fields consist of a fixed set of constants. For example, in Java, we can create Direction as an enum with fixed fields as EAST, WEST, NORTH, SOUTH.  enum is the keyword to create an enum type and similar to the class. Enum constants are implicitly static and final  **Question 30:** What is Java Reflection API? Why it’s so important to have?  Java Reflection API provides the ability to inspect and modify the runtime behavior of java application. We can inspect a java class, interface, enum and get their methods and field details. Reflection API is an advanced topic and we should avoid it in normal programming. Reflection API usage can break the design pattern such as Singleton pattern by invoking the private constructor i.e violating the rules of access modifiers.  Even though we don’t use Reflection API in normal programming, it’s very important to have. We can’t have any frameworks such as Spring, Hibernate or servers such as Tomcat, JBoss without Reflection API. They invoke the appropriate methods and instantiate classes through reflection API and use it a lot for other processing.  **Question 31:** What is inner class in java?  We can define a class inside a class and they are called nested classes. Any non-static nested class is known as an inner class. Inner classes are associated with the object of the class and they can access all the variables and methods of the outer class. Since inner classes are associated with the instance, we can’t have any static variables in them**.**  **Question 32:** What is anonymous inner class?  A local inner class without a name is known as an anonymous inner class. An anonymous class is defined and instantiated in a single statement. Anonymous inner class always extend a class or implement an interface.  Since an anonymous class has no name, it is not possible to define a constructor for an anonymous class. Anonymous inner classes are accessible only at the point where it is defined.  **Question 33:** What is default constructor?  No argument constructor of a class is known as default constructor. When we don’t define any constructor for the class, java compiler automatically creates the default no-args constructor for the class. If there are other constructors defined, then compiler won’t create default constructor for us.  **Question 34:** Can we have try without catch block?  Yes, we can have try-finally statement and hence avoiding catch block.  **Question 35:** What is Garbage Collection?  Garbage Collection is the process of looking at heap memory, identifying which objects are in use and which are not, and deleting the unused objects. In Java, the process of deallocating memory is handled automatically by the garbage collector.  We can run the garbage collector with code Runtime.getRuntime().gc() or use utility method System.gc()  **Question 36:** What is the use of System class?  Java System Class is one of the core classes. One of the easiest ways to log information for debugging is System.out.print() method.  System class is final so that we can’t subclass and override its behavior through inheritance. System class doesn’t provide any public constructors, so we can’t instantiate this class and that’s why all of its methods are static.  Some of the utility methods of System class are for array copy, get the current time, reading environment variables.  **Question 37:** Java is Pass by Value or Pass by Reference?  This is a very confusing question, we know that object variables contain the reference to the Objects in heap space. When we invoke any method, a copy of these variables is passed and gets stored in the stack memory of the method**.**  **Question 38:** What is difference between Heap and Stack Memory?  Major difference between Heap and Stack memory are as follows:  Heap memory is used by all the parts of the application whereas stack memory is used only by one thread of execution.  Whenever an object is created, it’s always stored in the Heap space and stack memory contains the reference to it. Stack memory only contains local primitive variables and reference variables to objects in heap space.  Memory management in the stack is done in a LIFO manner whereas it’s more complex in Heap memory because it’s used globally.  **Question 39:** static method invocation  package com.journaldev.util;  public class Test {  public static String foo(){  System.out.println("Test foo called");  return "";  }    public static void main(String args[]){  Test obj = null;  System.out.println(obj.foo());  }  }  Answer: Well this is a strange situation. We all have seen NullPointerException when we invoke a method on the object that is NULL. But here this program will work and prints “Test foo called”.  The reason for this is the java compiler code optimization. When the java code is compiled to produced byte code, it figures out that foo() is a static method and should be called using class. So it changes the method call obj.foo() to Test.foo() and hence no NullPointerException.  **Question 40:** What is Thread Pool? How can we create Thread Pool in Java?  A thread pool manages the pool of worker threads, it contains a queue that keeps tasks waiting to get executed.  A thread pool manages the collection of Runnable threads and worker threads execute Runnable from the queue.  java.util.concurrent.Executors provide implementation of java.util.concurrent.Executor interface to create the thread pool in java. Thread Pool Example program shows how to create and use Thread Pool in java. Or read ScheduledThreadPoolExecutor Example to know how to schedule tasks after certain delay**.**    Java thread pool manages the pool of worker threads. It contains a queue that keeps tasks waiting to get executed. We can use ThreadPoolExecutor to create thread pool in Java.  Java thread pool manages the collection of Runnable threads. The worker threads execute Runnable threads from the queue. java.util.concurrent.Executors provide factory and support methods for java.util.concurrent.Executor interface to create the thread pool in java.  Executors is a utility class that also provides useful methods to work with ExecutorService, ScheduledExecutorService, ThreadFactory, and Callable classes through various factory methods.  Let’s write a simple program to explain it’s working.  First, we need to have a Runnable class, named WorkerThread.java  package com.journaldev.threadpool;  public class WorkerThread implements Runnable {    private String command;    public WorkerThread(String s){  this.command=s;  }  @Override  public void run() {  System.out.println(Thread.currentThread().getName()+" Start. Command = "+command);  processCommand();  System.out.println(Thread.currentThread().getName()+" End.");  }  private void processCommand() {  try {  Thread.sleep(5000);  } catch (InterruptedException e) {  e.printStackTrace();  }  }  @Override  public String toString(){  return this.command;  }  }  **Table of Contents:**  1 ExecutorService Example  2 ThreadPoolExecutor Example  **ExecutorService Example**  Here is the test program class SimpleThreadPool.java, where we are creating fixed thread pool from Executors framework.  package com.journaldev.threadpool;  import java.util.concurrent.ExecutorService;  import java.util.concurrent.Executors;  public class SimpleThreadPool {  public static void main(String[] args) {  ExecutorService executor = Executors.newFixedThreadPool(5);  for (int i = 0; i < 10; i++) {  Runnable worker = new WorkerThread("" + i);  executor.execute(worker);  }  executor.shutdown();  while (!executor.isTerminated()) {  }  System.out.println("Finished all threads");  }  }  In the above program, we are creating a fixed-size thread pool of 5 worker threads. Then we are submitting 10 jobs to this pool, since the pool size is 5, it will start working on 5 jobs and other jobs will be in wait state, as soon as one of the job is finished, another job from the wait queue will be picked up by worker thread and get’s executed.  Here is the output of the above program.  pool-1-thread-2 Start. Command = 1  pool-1-thread-4 Start. Command = 3  pool-1-thread-1 Start. Command = 0  pool-1-thread-3 Start. Command = 2  pool-1-thread-5 Start. Command = 4  pool-1-thread-4 End.  pool-1-thread-5 End.  pool-1-thread-1 End.  pool-1-thread-3 End.  pool-1-thread-3 Start. Command = 8  pool-1-thread-2 End.  pool-1-thread-2 Start. Command = 9  pool-1-thread-1 Start. Command = 7  pool-1-thread-5 Start. Command = 6  pool-1-thread-4 Start. Command = 5  pool-1-thread-2 End.  pool-1-thread-4 End.  pool-1-thread-3 End.  pool-1-thread-5 End.  pool-1-thread-1 End.  Finished all threads  The output confirms that there are five threads in the pool named from “pool-1-thread-1” to “pool-1-thread-5” and they are responsible to execute the submitted tasks to the pool.  **ThreadPoolExecutor Example**  Executors class provide simple implementation of ExecutorService using ThreadPoolExecutor but ThreadPoolExecutor provides much more feature than that. We can specify the number of threads that will be alive when we create ThreadPoolExecutor instance and we can limit the size of thread pool and create our own RejectedExecutionHandler implementation to handle the jobs that can’t fit in the worker queue.  Here is our custom implementation of RejectedExecutionHandler interface.  package com.journaldev.threadpool;  import java.util.concurrent.RejectedExecutionHandler;  import java.util.concurrent.ThreadPoolExecutor;  public class RejectedExecutionHandlerImpl implements RejectedExecutionHandler {  @Override  public void rejectedExecution(Runnable r, ThreadPoolExecutor executor) {  System.out.println(r.toString() + " is rejected");  }  }  **ThreadPoolExecutor** provides several methods using which we can find out the current state of the executor, pool size, active thread count and task count. So I have a monitor thread that will print the executor information at a certain time interval.  package com.journaldev.threadpool;  import java.util.concurrent.ThreadPoolExecutor;  public class MyMonitorThread implements Runnable  {  private ThreadPoolExecutor executor;  private int seconds;  private boolean run=true;  public MyMonitorThread(ThreadPoolExecutor executor, int delay)  {  this.executor = executor;  this.seconds=delay;  }  public void shutdown(){  this.run=false;  }  @Override  public void run()  {  while(run){  System.out.println(  String.format("[monitor] [%d/%d] Active: %d, Completed: %d, Task: %d, isShutdown: %s, isTerminated: %s",  this.executor.getPoolSize(),  this.executor.getCorePoolSize(),  this.executor.getActiveCount(),  this.executor.getCompletedTaskCount(),  this.executor.getTaskCount(),  this.executor.isShutdown(),  this.executor.isTerminated()));  try {  Thread.sleep(seconds\*1000);  } catch (InterruptedException e) {  e.printStackTrace();  }  }    }  }  Here is the thread pool implementation example using ThreadPoolExecutor.  package com.journaldev.threadpool;  import java.util.concurrent.ArrayBlockingQueue;  import java.util.concurrent.Executors;  import java.util.concurrent.ThreadFactory;  import java.util.concurrent.ThreadPoolExecutor;  import java.util.concurrent.TimeUnit;  public class WorkerPool {  public static void main(String args[]) throws InterruptedException{  //RejectedExecutionHandler implementation  RejectedExecutionHandlerImpl rejectionHandler = new RejectedExecutionHandlerImpl();  //Get the ThreadFactory implementation to use  ThreadFactory threadFactory = Executors.defaultThreadFactory();  //creating the ThreadPoolExecutor  ThreadPoolExecutor executorPool = new ThreadPoolExecutor(2, 4, 10, TimeUnit.SECONDS, new ArrayBlockingQueue<Runnable>(2), threadFactory, rejectionHandler);  //start the monitoring thread  MyMonitorThread monitor = new MyMonitorThread(executorPool, 3);  Thread monitorThread = new Thread(monitor);  monitorThread.start();  //submit work to the thread pool  for(int i=0; i<10; i++){  executorPool.execute(new WorkerThread("cmd"+i));  }    Thread.sleep(30000);  //shut down the pool  executorPool.shutdown();  //shut down the monitor thread  Thread.sleep(5000);  monitor.shutdown();    }  }  Notice that while initializing the ThreadPoolExecutor, we are keeping initial pool size as 2, maximum pool size to 4 and work queue size as 2. So if there are 4 running tasks and more tasks are submitted, the work queue will hold only 2 of them and the rest of them will be handled by RejectedExecutionHandlerImpl.  Here is the output of the above program that confirms the above statement.  pool-1-thread-1 Start. Command = cmd0  pool-1-thread-4 Start. Command = cmd5  cmd6 is rejected  pool-1-thread-3 Start. Command = cmd4  pool-1-thread-2 Start. Command = cmd1  cmd7 is rejected  cmd8 is rejected  cmd9 is rejected  [monitor] [0/2] Active: 4, Completed: 0, Task: 6, isShutdown: false, isTerminated: false  [monitor] [4/2] Active: 4, Completed: 0, Task: 6, isShutdown: false, isTerminated: false  pool-1-thread-4 End.  pool-1-thread-1 End.  pool-1-thread-2 End.  pool-1-thread-3 End.  pool-1-thread-1 Start. Command = cmd3  pool-1-thread-4 Start. Command = cmd2  [monitor] [4/2] Active: 2, Completed: 4, Task: 6, isShutdown: false, isTerminated: false  [monitor] [4/2] Active: 2, Completed: 4, Task: 6, isShutdown: false, isTerminated: false  pool-1-thread-1 End.  pool-1-thread-4 End.  [monitor] [4/2] Active: 0, Completed: 6, Task: 6, isShutdown: false, isTerminated: false  [monitor] [2/2] Active: 0, Completed: 6, Task: 6, isShutdown: false, isTerminated: false  [monitor] [2/2] Active: 0, Completed: 6, Task: 6, isShutdown: false, isTerminated: false  [monitor] [2/2] Active: 0, Completed: 6, Task: 6, isShutdown: false, isTerminated: false  [monitor] [2/2] Active: 0, Completed: 6, Task: 6, isShutdown: false, isTerminated: false  [monitor] [2/2] Active: 0, Completed: 6, Task: 6, isShutdown: false, isTerminated: false  [monitor] [0/2] Active: 0, Completed: 6, Task: 6, isShutdown: true, isTerminated: true  [monitor] [0/2] Active: 0, Completed: 6, Task: 6, isShutdown: true, isTerminated: true  Notice the change in active, completed and total completed task count of the executor. We can invoke shutdown() method to finish execution of all the submitted tasks and terminate the thread pool.  If you want to schedule a task to run with delay or periodically then you can use ScheduledThreadPoolExecutor class. Read more about them at Java Schedule Thread Pool Executor.  Welcome to the Java Scheduler Example. Today we will look into ScheduledExecutorService and it’s implementation class ScheduledThreadPoolExecutor example.  Table of Contents  1 Java Scheduler ScheduledExecutorService  1.1 Java Scheduler Example  1.2 ScheduledExecutorService scheduleAtFixedRate(Runnable command,long initialDelay,long period,TimeUnit unit)  1.3 ScheduledExecutorService scheduleWithFixedDelay(Runnable command,long initialDelay,long delay,TimeUnit unit)  **Java Scheduler ScheduledExecutorService**  Sometimes we need to execute a task periodically or after specific delay. Java provides Timer Class through which we can achieve this but sometimes we need to run similar tasks in parallel. So creating multiple Timer objects will be an overhead to the system and it’s better to have a thread pool of scheduled tasks.  Java provides scheduled thread pool implementation through ScheduledThreadPoolExecutor class that implements ScheduledExecutorService interface. ScheduledExecutorService defines the contract methods to schedule a task with different options.  Sometime back I wrote a post about Java ThreadPoolExecutor where I was using Executors class to create the thread pool. Executors class also provide factory methods to create ScheduledThreadPoolExecutor where we can specify the number of threads in the pool.  Java Scheduler Example  Let’s say we have a simple Runnable class like below.  WorkerThread.java  package com.journaldev.threads;  import java.util.Date;  public class WorkerThread implements Runnable{  private String command;    public WorkerThread(String s){  this.command=s;  }  @Override  public void run() {  System.out.println(Thread.currentThread().getName()+" Start. Time = "+new Date());  processCommand();  System.out.println(Thread.currentThread().getName()+" End. Time = "+new Date());  }  private void processCommand() {  try {  Thread.sleep(5000);  } catch (InterruptedException e) {  e.printStackTrace();  }  }  @Override  public String toString(){  return this.command;  }  }  It’s a simple Runnable class that takes around 5 seconds to execute its task.  Let’s see a simple example where we will schedule the worker thread to execute after 10 seconds delay. We will use Executors class newScheduledThreadPool(int corePoolSize) method that returns instance of ScheduledThreadPoolExecutor. Here is the code snippet from Executors class.  public static ScheduledExecutorService newScheduledThreadPool(int corePoolSize) {  return new ScheduledThreadPoolExecutor(corePoolSize);  }  Below is our java scheduler example program using ScheduledExecutorService and ScheduledThreadPoolExecutor implementation.  package com.journaldev.threads;  import java.util.Date;  import java.util.concurrent.Executors;  import java.util.concurrent.ScheduledExecutorService;  import java.util.concurrent.TimeUnit;  public class ScheduledThreadPool {  public static void main(String[] args) throws InterruptedException {  ScheduledExecutorService scheduledThreadPool = Executors.newScheduledThreadPool(5);      //schedule to run after sometime  System.out.println("Current Time = "+new Date());  for(int i=0; i<3; i++){  Thread.sleep(1000);  WorkerThread worker = new WorkerThread("do heavy processing");  scheduledThreadPool.schedule(worker, 10, TimeUnit.SECONDS);  }    //add some delay to let some threads spawn by scheduler  Thread.sleep(30000);    scheduledThreadPool.shutdown();  while(!scheduledThreadPool.isTerminated()){  //wait for all tasks to finish  }  System.out.println("Finished all threads");  }  }  When we run above java scheduler example program, we get following output that confirms that tasks are running with 10 seconds delay.  Current Time = Tue Oct 29 15:10:03 IST 2013  pool-1-thread-1 Start. Time = Tue Oct 29 15:10:14 IST 2013  pool-1-thread-2 Start. Time = Tue Oct 29 15:10:15 IST 2013  pool-1-thread-3 Start. Time = Tue Oct 29 15:10:16 IST 2013  pool-1-thread-1 End. Time = Tue Oct 29 15:10:19 IST 2013  pool-1-thread-2 End. Time = Tue Oct 29 15:10:20 IST 2013  pool-1-thread-3 End. Time = Tue Oct 29 15:10:21 IST 2013  Finished all threads  Note that all the schedule() methods return instance of ScheduledFuture that we can use to get the thread state information and delay time for the thread.  ScheduledFuture extends Future interface, read more about them at Java Callable Future Example.  There are two more methods in ScheduledExecutorService that provide option to schedule a task to run periodically.  **ScheduledExecutorService scheduleAtFixedRate(Runnable command,long initialDelay,long period,TimeUnit unit)**  We can use ScheduledExecutorService scheduleAtFixedRate method to schedule a task to run after initial delay and then with the given period.  The time period is from the start of the first thread in the pool, so if you are specifying period as 1 second and your thread runs for 5 second, then the next thread will start executing as soon as the first worker thread finishes it’s execution.  For example, if we have code like this:  for (int i = 0; i < 3; i++) {  Thread.sleep(1000);  WorkerThread worker = new WorkerThread("do heavy processing");  // schedule task to execute at fixed rate  scheduledThreadPool.scheduleAtFixedRate(worker, 0, 10,  TimeUnit.SECONDS);  }  Then we will get output like below.  Current Time = Tue Oct 29 16:10:00 IST 2013  pool-1-thread-1 Start. Time = Tue Oct 29 16:10:01 IST 2013  pool-1-thread-2 Start. Time = Tue Oct 29 16:10:02 IST 2013  pool-1-thread-3 Start. Time = Tue Oct 29 16:10:03 IST 2013  pool-1-thread-1 End. Time = Tue Oct 29 16:10:06 IST 2013  pool-1-thread-2 End. Time = Tue Oct 29 16:10:07 IST 2013  pool-1-thread-3 End. Time = Tue Oct 29 16:10:08 IST 2013  pool-1-thread-1 Start. Time = Tue Oct 29 16:10:11 IST 2013  pool-1-thread-4 Start. Time = Tue Oct 29 16:10:12 IST 2013  **ScheduledExecutorService scheduleWithFixedDelay(Runnable command,long initialDelay,long delay,TimeUnit unit)**  ScheduledExecutorService scheduleWithFixedDelay method can be used to start the periodic execution with initial delay and then execute with given delay. The delay time is from the time thread finishes it’s execution. So if we have code like below:  for (int i = 0; i < 3; i++) {  Thread.sleep(1000);  WorkerThread worker = new WorkerThread("do heavy processing");  scheduledThreadPool.scheduleWithFixedDelay(worker, 0, 1,  TimeUnit.SECONDS);  }  Then we will get output like below.  Current Time = Tue Oct 29 16:14:13 IST 2013  pool-1-thread-1 Start. Time = Tue Oct 29 16:14:14 IST 2013  pool-1-thread-2 Start. Time = Tue Oct 29 16:14:15 IST 2013  pool-1-thread-3 Start. Time = Tue Oct 29 16:14:16 IST 2013  pool-1-thread-1 End. Time = Tue Oct 29 16:14:19 IST 2013  pool-1-thread-2 End. Time = Tue Oct 29 16:14:20 IST 2013  pool-1-thread-1 Start. Time = Tue Oct 29 16:14:20 IST 2013  pool-1-thread-3 End. Time = Tue Oct 29 16:14:21 IST 2013  pool-1-thread-4 Start. Time = Tue Oct 29 16:14:21 IST 2013  **Question 41:** What is BlockingQueue?  java.util.concurrent.BlockingQueue is a Queue that supports operations that wait for the queue to become non-empty when retrieving and removing an element, and wait for space to become available in the queue when adding an element.  BlockingQueue doesn’t accept null values and throw NullPointerException if you try to store null value in the queue.  BlockingQueue implementations are thread-safe. All queuing methods are atomic in nature and use internal locks or other forms of concurrency control.  BlockingQueue interface is part of the Java collections framework and it’s primarily used for implementing the producer-consumer problem.  **Question 42:** What is Callable and Future?  Java 5 introduced java.util.concurrent.Callable interface in concurrency package that is similar to Runnable interface but it can return any Object and able to throw Exception.  The Callable interface uses Generics to define the return type of Object. Executors class provide useful methods to execute Callable in a thread pool. Since callable tasks run in parallel, we have to wait for the returned Object. Callable tasks return java.util.concurrent.Future object. Using Future we can find out the status of the Callable task and get the returned Object. It provides the get() method that can wait for the Callable to finish and then return the result.  **Question 43:** What is FutureTask Class?  FutureTask is the base implementation class of Future interface and we can use it with Executors for asynchronous processing. Most of the time we don’t need to use FutureTask class but it comes real handy if we want to override some of the methods of Future interface and want to keep most of the base implementation. We can just extend this class and override the methods according to our requirements  **Question 44:** What are Concurrent Collection Classes?  Java Collection classes are fail-fast which means that if the Collection will be changed while some thread is traversing over it using iterator, the iterator.next() will throw ConcurrentModificationException.  Concurrent Collection classes support full concurrency of retrievals and adjustable expected concurrency for updates.  Major classes are ConcurrentHashMap, CopyOnWriteArrayList and CopyOnWriteArraySet  **Question 45:** Why String is popular HashMap key in Java?  Since String is immutable, its hashcode is cached at the time of creation and it doesn’t need to be calculated again. This makes it a great candidate for the key in a Map and it’s processing is fast than other HashMap key objects. This is why String is mostly used Object as HashMap keys**.**  **Question 46:** What are Collection related features in Java 8?  Java 8 has brought major changes in the Collection API. Some of the changes are:  Java Stream API for collection classes for supporting sequential as well as parallel processing  Iterable interface is extended with forEach() default method that we can use to iterate over a collection. It is very helpful when used with lambda expressions because its argument Consumer is a function interface.  Miscellaneous Collection API improvements such as forEachRemaining(Consumer action) method in Iterator interface, Map replaceAll(), compute(), merge() methods.  **Question 47:** What is the benefit of Generics in Collections Framework?  Java 1.5 came with Generics and all collection interfaces and implementations use it heavily. Generics allow us to provide the type of Object that a collection can contain, so if you try to add any element of other type it throws compile time error.  This avoids ClassCastException at Runtime because you will get the error at compilation. Also Generics make code clean since we don’t need to use casting and instanceof operator.  **Question 48:** What is difference between Enumeration and Iterator interface?  Enumeration is twice as fast as Iterator and uses very little memory. Enumeration is very basic and fits basic needs. But the Iterator is much safer as compared to Enumeration because it always denies other threads to modify the collection object which is being iterated by it.  Iterator takes the place of Enumeration in the Java Collections Framework. Iterators allow the caller to remove elements from the underlying collection that is not possible with Enumeration. Iterator method names have been improved to make its functionality clear.  **Question 49:** Why there is not method like Iterator.add() to add elements to the collection?  The semantics are unclear, given that the contract for Iterator makes no guarantees about the order of iteration. Note, however, that ListIterator does provide an add operation, as it does guarantee the order of the iteration.  **Question 50:** What do you understand by iterator fail-fast property?  Iterator fail-fast property checks for any modification in the structure of the underlying collection everytime we try to get the next element. If there are any modifications found, it throws ConcurrentModificationException. All the implementations of Iterator in Collection classes are fail-fast by design except the concurrent collection classes like ConcurrentHashMap and CopyOnWriteArrayList  **Question 51:** What is difference between fail-fast and fail-safe?  Iterator fail-safe property work with the clone of underlying collection, hence it’s not affected by any modification in the collection. By design, all the collection classes in java.util package are fail-fast whereas collection classes in java.util.concurrent are fail-safe.  Fail-fast iterators throw ConcurrentModificationException whereas fail-safe iterator never throws ConcurrentModificationException**.**  **Question 52:** How to avoid ConcurrentModificationException while iterating a collection?  We can use concurrent collection classes to avoid ConcurrentModificationException while iterating over a collection, for example CopyOnWriteArrayList instead of ArrayList**.**  **Question 53:** What is the importance of hashCode() and equals() methods?  HashMap uses the Key object hashCode() and equals() method to determine the index to put the key-value pair. These methods are also used when we try to get value from HashMap. If these methods are not implemented correctly, two different Key’s might produce the same hashCode() and equals() output and in that case, rather than storing it at a different location, HashMap will consider the same and overwrite them.  Similarly all the collection classes that doesn’t store duplicate data use hashCode() and equals() to find duplicates, so it’s very important to implement them correctly. The implementation of equals() and hashCode() should follow these rules.  If o1.equals(o2), then o1.hashCode() == o2.hashCode()should always be true.  If o1.hashCode() == o2.hashCode is true, it doesn’t mean that o1.equals(o2) will be true  **Question 54:** Which collection classes provide random access of its elements?  ArrayList, HashMap, TreeMap, Hashtable, and Vector classes provide random access to its elements.  **Question 55:** Which collection classes are thread-safe?  Vector, Hashtable, Properties and Stack are synchronized classes, so they are thread-safe and can be used in multi-threaded environment. Java 1.5 Concurrent API included some collection classes that allows modification of collection while iteration because they work on the clone of the collection, so they are safe to use in multi-threaded environment**.**  **Question 56:** Explain Java Exception Hierarchy?  Java Exceptions are hierarchical and inheritance is used to categorize different types of exceptions. Throwable is the parent class of Java Exceptions Hierarchy and it has two child objects – Error and Exception. Exceptions are further divided into checked exceptions and runtime exceptions.  Errors are exceptional scenarios that are out of the scope of application and it’s not possible to anticipate and recover from them, for example, hardware failure, JVM crash, or out-of-memory error.  Checked Exceptions are exceptional scenarios that we can anticipate in a program and try to recover from it, for example, FileNotFoundException. We should catch this exception and provide a useful message to the user and log it properly for debugging purposes. Exception is the parent class of all Checked Exceptions.  Runtime Exceptions are caused by bad programming, for example, trying to retrieve an element from the Array. We should check the length of the array first before trying to retrieve the element otherwise it might throw ArrayIndexOutOfBoundException at runtime. RuntimeException is the parent class of all runtime exceptions.  exception handling in java interview questions, exception handling interview questions, java exception interview questions  **Question 57:**  **Question 58:**  **Question 59:**  **Question 60:**  **Question 61:** |
|  |
|  |