1. **Do we need changes and reforms in Java? Disadvantages**

Yes, we do need changes and reforms/Disadvantages in Java language because of the following reasons.

1. To make the best and efficient use of multi-million CPUs that are deployed at work throughout the world.
2. To meet with the changes and up-gradation made in the technology hardware and software.
3. To support the running of various kinds of applications.
4. To create highly concurrent and scalable applications.
5. To make use of all the [functional programming](https://hackr.io/blog/functional-programming) features which are an integral part of the latest Java SE 8 computer language.
6. **Which are the most popular and well-known latest features of Java SE 8?**

The most popular and well-known latest features of Java SE 8 include the following.

1. Functional Interfaces.
2. Collections API Enhancements.
3. Lambda Expressions.
4. Spliterator.
5. Stream API and more.
6. **What makes Java SE 8 superior over others?**

Java SE 8 has the following features that make it superior over others.

1. It writes the parallel code.
2. It offers even more useable codes.
3. It has improved performance applications.
4. It has more readable and concise codes.
5. It supports writing database including promotions.
6. **Define a Lambda Expression in Java SE 8?**

A Lambda Expression is Java SE 8 is a name given to an anonymous function that helps in accepting a different set of inputs parameters and offers a variety of results outcomes.

1. **Why is Lambda Expression coined as a block of code?**

Lambda Expression is coined as a block of code because it has no name and can be with or without the parameters and results.

1. **What is the distinct feature of the Block of Code?**

A Block of Code has the distinct feature of getting executed on only demand.

1. **Define the different parts of Lambda Expression?**

There are three different parts of Lambda Expression which include the following.

1. Parameter List.
2. Lambda Body Expression.
3. Lambda Arrow Operator.
4. **How is the Parameter List of Lambda Expression different from the Lambda Arrow Operator?**

Lambda Expression can carry zero, one or even more parameters at one time. On the other hand, the Lambda Arrow Operator separates these parameters from the list and body using the icon "->".

1. **What are the distinct features of Lambda Expression?**

The distinct features of Lambda Expression include the following.

1. It has no specific parameters.
2. It returns any results.
3. Its type is “Java.lang.Runnable”.
4. **How is Lambda Expression highly useful?**

Lambda Expression is highly useful because it offers a functional interface.

1. **Define the functional interface?**

The functional interface is referred to as the interface which carries only one abstract method. The code example can be used for explaining the functional interface.

// Java program to demonstrate Implementation **of**  
// functional interface using lambda expressions  
class Test  
{  
  public static void main(String args[])  
  {  
    // lambda expression to create the object  
    **new** Thread(()->  
       ).start();  
  }  
}

1. **Is there any difference between the Functional interface and SAM interface?**

No, there is no difference between Functional interface and SAM interface. The SAM interface or Single Abstract Method interface is a kind of Functional interface defined in Java SE 8 API.

1. **Can we define our functional interface? If yes, how?**

Yes, we can define our functional interface. They can be defined using Java SE 8’s  @Functionalinterface annotation to mark an interface as a functional interface.

1. **What are the guidelines that are needed to be followed in Functional Interface?**

There are several guidelines stated below which are needed to be followed in Functional Interface.

1. The interface should be defined with only one abstract method.
2. Not more than one abstract can be defined.
3. Making use of @Functionalinterface annotation in the interface definition.
4. The override of the Java.lang.object class’s method will not be considered as an abstract method.
5. Any methods can be used for defining a number.
6. **Which are the two most popular methods that can be used for defining any number in a Functional Interface?**

The two most popular methods used for defining any number in a Functional Interface include Static methods and Default methods.

1. **Is it mandatory to define a Functional Interface with @Functionalinterface annotation?**

Not it is not mandatory to define a Functional Interface with @Functionalinterface annotation in case we don’t want it and hence omitting this annotation can be followed.

1. **What is the compulsion in the Functional Interface definition?**

The compulsion in the functional interface definition is that the Java Compiler forces to make use of one abstract inside the interface.

1. **What is the link between Lambda Expressions and Functional Interface?**

When we are using Lambda Expressions then that means that we are using a Functional interface. Therefore they are both interrelated. This means that Lambda Expressions are a part of the Functional interface which is a bigger platform carrying various other features and expressions.

1. **How is Collection API different from Stream API?**

The difference between Collection API and Stream API can be stated as under.

|  |  |
| --- | --- |
| Collection API | Stream API |
| It is available since the introduction of Java 1.2 | It is made available after the introduction of Java SE 8 |
| It helps in storing Data which are set of objects. | It helps in computing data wherein computations of a set of data objects are made. |
| It can be used for storing a limited number of elements. | It can be used for storing an unlimited number of elements. |
| The construction of the Collection object is done Eagerly. | The construction of the Stream object is done Lazily. |

1. **Define a Spliterator in Java SE 8?**

A Spliterator in Java SE 8 is one of the latest iterator interface introduced by [Oracle Corporation](https://www.oracle.com/corporate/) as part of Java SE 8.

1. **How is Spliterator a different Iterator?**

The following are the differences between Spliterator and Iterator.

|  |  |
| --- | --- |
| Spliterator | Iterator |
| It is introduced along with Java SE 8 | It was introduced long ago with Java 1.2 |
| It is defined as a Spliterator iterator. | It is defined as a non-spilterator Iterator. |
| It can be used in Stream API. | It can be used in Collection API. |
| It helps to iterate streams in parallel and sequential order. | It helps in iterate collections only in sequential order. |
| The examples include tryAdvance() | The examples include, next(), hasNext(). |

1. **Define Optional in Java SE 8?**

Optional is defined as a final class that is introduced as an integral part of Java SE8. It is a java. util package.

1. **What is the use of Optional in Java SE 8?**

Optional in Java SE 8 is used for representing optional values that exist or does not exist. It does not support too many null checks and NullPointerException. In addition to that, it avoids the runtime NullPointerExceptions and helps in developing cleaner Java APIs.

1. **How many values can Optional have in Java SE 8?**

Optional in Java SE 8 can have either one value or zero value.

1. **What are the advantages of Optional?**

Optional has the following advantages.

1. It helps in avoiding the null checks.
2. It is used for avoiding “NullPointerException”.
3. **Define Type Inference in Java SE 8?**

Type Interface in Java 8 is referred to as determining the Type by the compiler at the compile time.

1. **Was Type Inference available before Java SE 8?**

Yes, Type Inference was available before Java SE 8 in Java 7 and even earlier [Java languages](https://hackr.io/blog/what-is-java).

1. **What are the common types of Functional Interfaces in the Standard Library?**

The common types of Functional Interfaces in the Standard Library include the following.

1. Predicate
2. Function
3. Consumer
4. Supplier
5. BiFunction
6. BinaryOperator
7. UnaryOperator
8. **What is the Default Method?**

A Default Method is found in an interface and can be used for the implementation purpose of new functionality added to the interface.

1. **Define Nashorn in Java SE 8?**

Nashorn is the latest Javascript processing engine that is used on the Java platform of Java 8.

1. **What was used as a Javascript processing engine before Java SE 8?**

 Earlier to Nashorn, Mozilla Rhino was used as a Javascript processing engine.

1. **What is the use of the JJS command-line tool?**

JSS is the latest executable command-line tool in Java 8 which helps in executing the Javascript code at the console.

1. **What is the major difference between Map and FlatMap stream operation?**

The major difference between Map and FlatMap stream operation is that the earlier wraps its return value inside its ordinal type while the latter does not.

1. **What is the similarity between Map and Flat map stream operation?**

Both the Map and FlatMap stream operation is intermediate stream operations that receive a function and also apply these functions to different elements of the stream.

1. **Define Stream Pipelining?**

Stream Pipelining in Java 8 is used for chaining operations together by splitting the operations that can happen on one stream.

1. **What are the two categories of Stream Pipelining?**

The two categories of Stream Pipelining are Intermediate operations and Terminal operations.

1. **What is compulsion in using Stream Pipeline?**

The compulsion in using Stream Pipeline is the presence of a terminal operation which helps in returning the final value and supports termination of the pipeline.

1. **What is the role of the New Date and Time API?**

New Date and Time API is designed in Java SE 8 under the package java time so that the problems and issues related to JDK or Java.util.date can be avoided.

1. **Can we create our functional interface?**

Yes, we can create our functional interface by using the code “Printable”.

1. **Define Predicate? Display its Code?**

Predicate is a single argument function that gives the outcome as true or false. Its code is <T>.

1. **Define Function? Display its Code?**

Function is a single argument function that gives outcomes in the form of an object. Its code is <T, R>.

1. **What are the similarities between Predicate and Function?**

Predicate and Function are both functional interfaces.

1. **What are the Core API classes for Java SE 8?**

The Core API classes for Java SE 8 include LocalDate, LocalTime, and LocalDateTime.

1. **Define PermGen? What is its current status?**

PermGen is used to store classes. It has been removed in Java SE 8 and has been replaced with MetaSpace.

1. **What is the advantage of Metaspace over PermGen?**

PerGen was fixed in size and could not grow dynamically, while, Metaspace can grow dynamically and does have any type of size constraint.

1. **What are the examples of Intermediate Operations?**

Examples of Intermediate Operations include the following.

1. Limit(long n)
2. skip (long n)
3. Distinct ()
4. Filter (Predicate)
5. Map (Function)
6. **What are the examples of Terminal Operations?**

The examples of Terminal Operations are as follows.

1. Max.
2. Min.
3. AnyMatch.
4. AllMatch.
5. Reduce.
6. ToArray.
7. Count.
8. **Can we list the numbers and remove the duplicate elements in the list using Java SE 8 features?**

Yes, we can list the numbers and remove the duplicate elements in the list by applying stream and then collecting it to set using Collections.toSet() method.

1. Explain JDK, JRE and 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**. |

1. 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.

1. 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.

1. 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.
2. 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.

1. 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.
3. 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.

1. 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. |

1. 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.

1. 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. |

1. 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

1. 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.

1. 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.

1. 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:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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 |

1. What is an object in Java and how is it created?

An object is a real-world entity that has a state and behaviour. 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();

1. 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.

1. What are the main concepts of OOPs in Java?

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

*Inheritance:*Inheritance is a process where one class acquires the properties of another.

*Encapsulation:*Encapsulation in Java is a mechanism of wrapping up the data and code together as a single unit.

*Abstraction:*Abstraction is the methodology of hiding the implementation details from the user and only providing the functionality to the users.

*Polymorphism:*Polymorphism is the ability of a variable, function or object to take multiple forms.

1. 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.

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.

1. 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 |

1. 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.

1. 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:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | for (int i = 0; i < 5; i++)  {  if (i == 3)  {  break;  }  System.out.println(i);  } |

**Example continue:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | for (int i = 0; i < 5; i++)  {  if(i == 2)  {  continue;  }  System.out.println(i);  } |

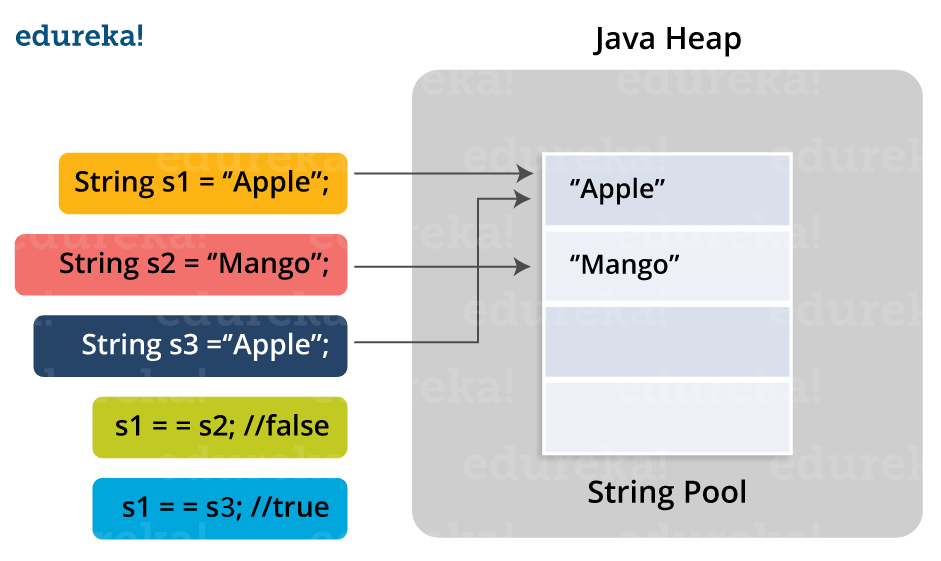
1. What is the difference between this() and super() in Java?

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 |

1. 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.

****

1. 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 |

1. 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()
3. **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 |

1. 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
4. 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.

1. 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 |

1. 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.
3. 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

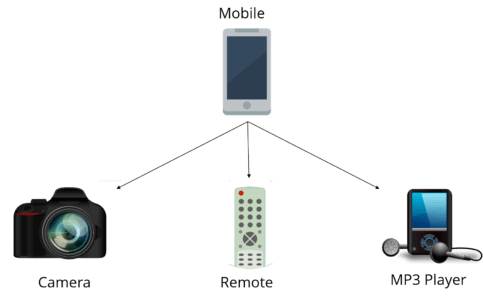
The below image shows the complete hierarchy of the Java Collection.



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

1. 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.

1. 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.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | 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();  }  } |

1. 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)
3. 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();

}

1. 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 |

1. 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.

1. 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.
5. 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.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | 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.

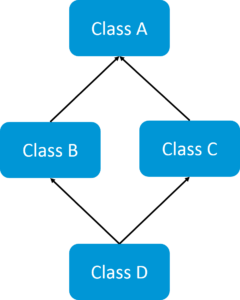
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | class Car {  void run(){  System.out.println(“car is running”);  }  Class Audi extends Car{  void run()  {  System.out.prinltn("Audi is running safely with 100km");  }  public static void main( String args[])  {  Car b=new Audi();  b.run();  }  } |

1. 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.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | 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();  }  } |

1. 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.

1. 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.

1. 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.

1. 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.

1. 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.

1. What is object cloning in Java?

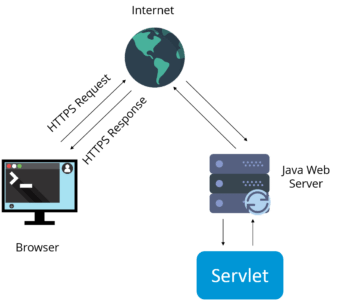
Object cloning in Java is the process of creating an exact copy of an object. It basically means the ability to create an object with a similar state as the original object. To achieve this, Java provides a method **clone()** to make use of this functionality. This method creates a new instance of the 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.

1. 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.

1. 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.



1. 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 | |

1. 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

1. 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. |

1. 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
6. 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.

1. What are the differences between ServletContext vs ServletConfig?

|  |  |
| --- | --- |
| ServletConfig | ServletContext |
| Servlet config object represent single servlet | It represent whole web application running on particular JVM and common for all the servlet |
| 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. |

1. 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

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)

### 

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

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

1. 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.

1. 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.

1. 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

1. 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.

1. What is JDBC ResultSetMetaData interface?

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

1. 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.

1. 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.

1. 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.

1. 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.
4. What is Spring?

Spring is essentially a lightweight, integrated framework that can be used for developing enterprise applications in java.

1. 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

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

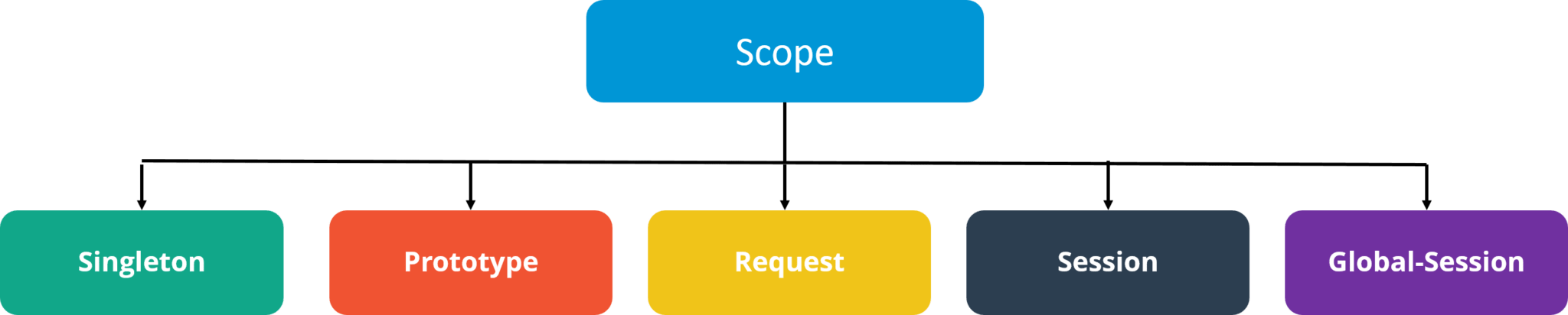
The important annotations are:

* @Required
* @Autowired
* @Qualifier
* @Resource
* @PostConstruct
* @PreDestroy

1. Explain Bean in Spring and List the different Scopes of Spring bean.

Beans are objects that form the backbone of a Spring application. They are managed by the Spring IoC container. In other words, a bean is an object that is instantiated, assembled, and managed by a Spring IoC container.

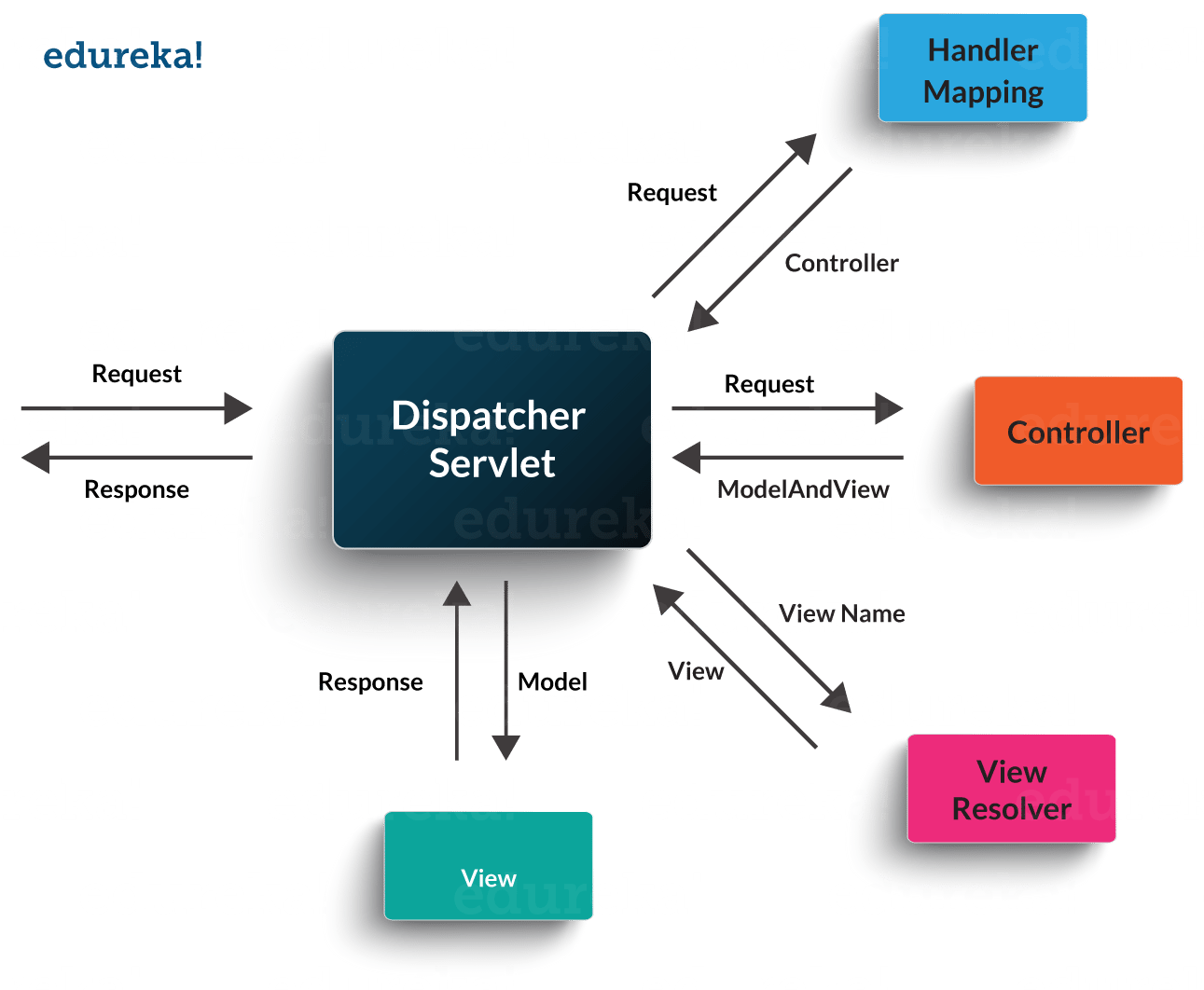
There are five Scopes defined in Spring beans.



* **Singleton**: Only one instance of the bean will be created for each container. This is the default scope for the spring beans. While using this scope, make sure spring bean doesn’t have shared instance variables otherwise it might lead to data inconsistency issues because it’s not thread-safe.
* **Prototype**: A new instance will be created every time the bean is requested.
* **Request**: This is same as prototype scope, however it’s meant to be used for web applications. A new instance of the bean will be created for each HTTP request.
* **Session**: A new bean will be created for each HTTP session by the container.
* **Global-session**: This is used to create global session beans for Portlet applications.

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

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

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

### 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.

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 structured training from edureka!

## Hibernate 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.

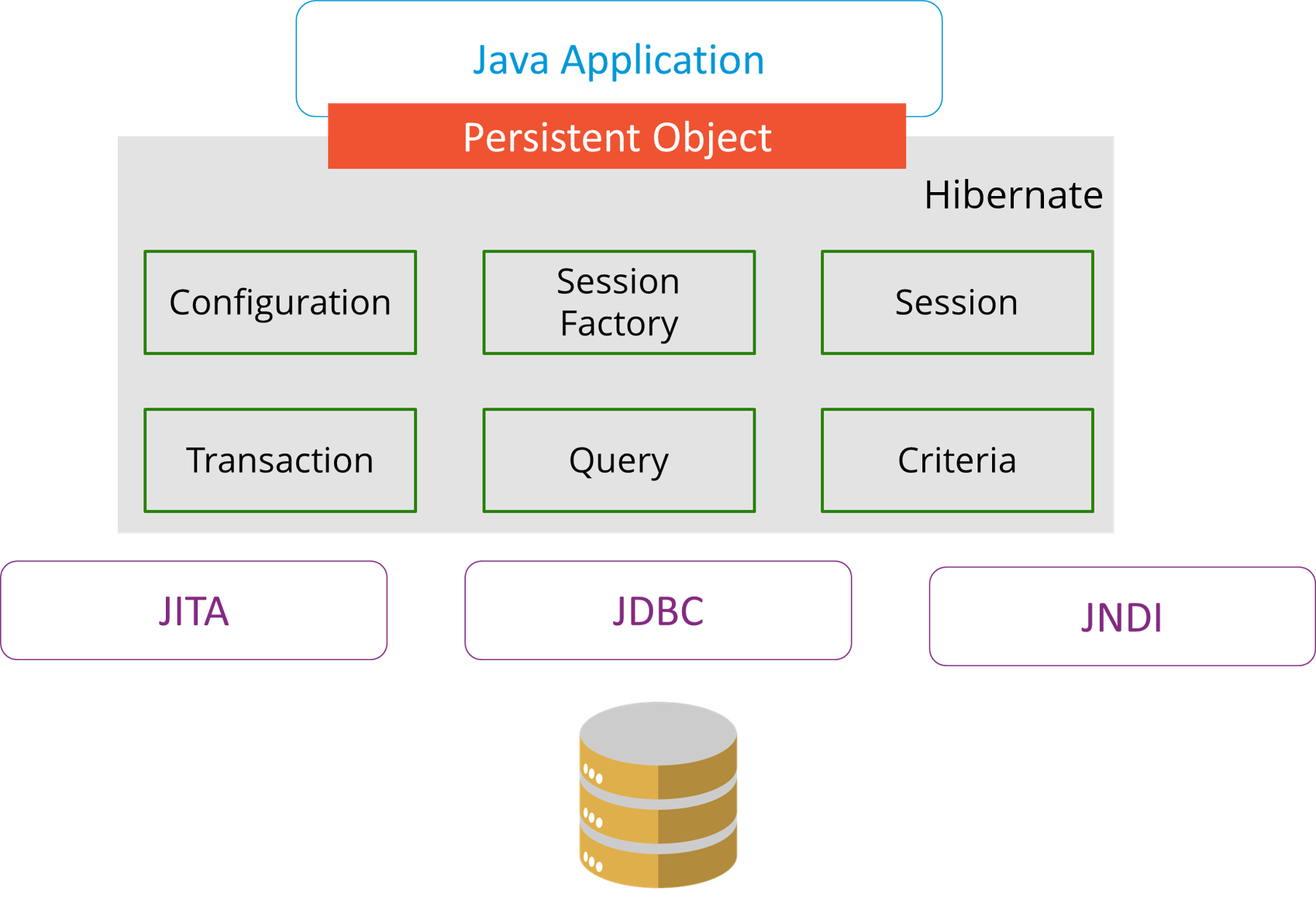
### 2. What are the important benefits of using Hibernate Framework?

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.



### 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!

## Java Interview Questions: JSP

### 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 :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | 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}.

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!

## 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.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | 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.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | 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.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | 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?

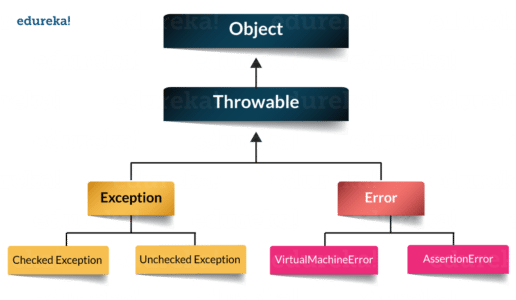
The hierarchy is as follows:

[[](https://www.edureka.co/java-j2ee-training-course)](https://www.edureka.co/java-j2ee-training-course" \t "_blank)

### [Java, J2EE & SOA Certification Training](https://www.edureka.co/java-j2ee-training-course" \t "_blank)

[Weekday / Weekend BatchesSee Batch Details](https://www.edureka.co/java-j2ee-training-course" \t "_blank)

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?

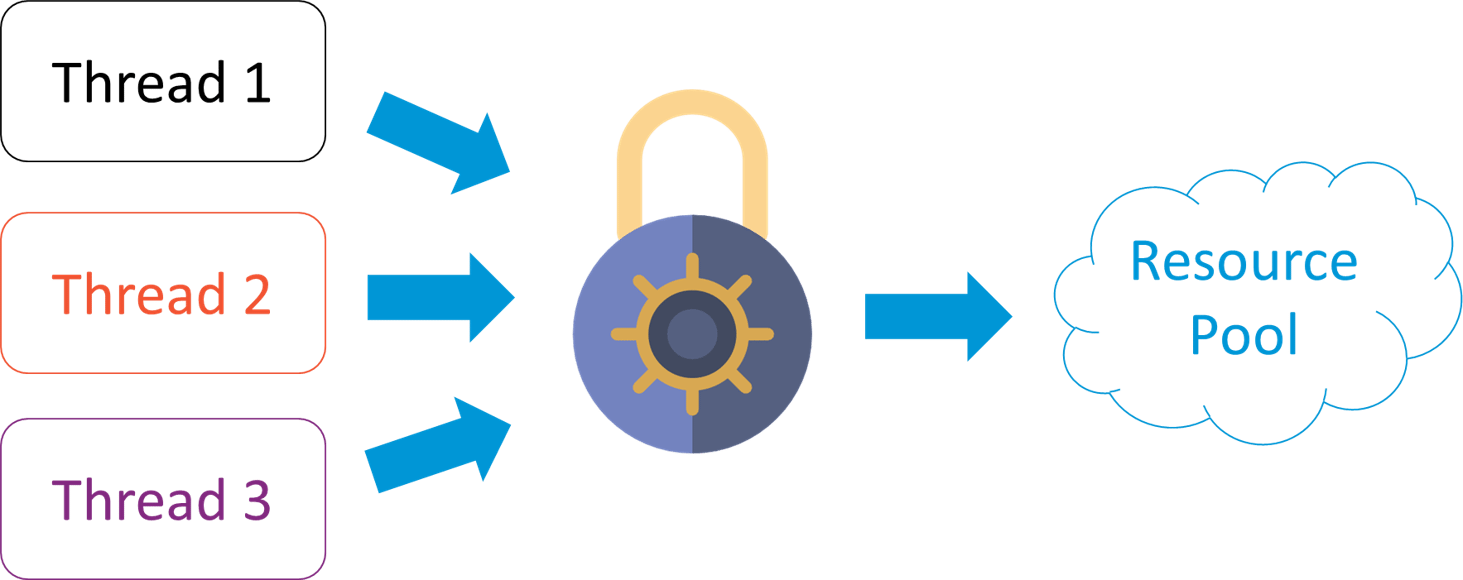
|  |  |  |
| --- | --- | --- |
|  | Process | Thread |
| **Definition** | An executing instance of a program is called a process. | A thread is a subset of the process. |
| **Communication** | Processes must use inter-process communication to communicate with sibling processes. | Threads can directly 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.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | 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

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 structured training from edureka!

So this brings us to the end of the Java interview questions blog. The topics that you learned in this Java Interview Questions blog are the most sought-after skill sets that recruiters look for in a Java Professional. These set of Java Interview Questions will definitely help you ace your job interview.

1. Question 1. What Are The New Features Introduced In Java 8?

Answer :

There are dozens of features added to Java 8, the most significant ones are mentioned below −

* + Lambda expression − Adds functional processing capability to Java.
  + Method references − Referencing functions by their names instead of invoking them directly. Using functions as parameter.
  + Default method − Interface to have default method implementation.
  + New tools − New compiler tools and utilities are added like 'jdeps' to figure out dependencies.
  + Stream API − New stream API to facilitate pipeline processing.
  + Date Time API − Improved date time API.
  + Optional − Emphasis on best practices to handle null values properly.
  + Nashorn, JavaScript Engine − A Java-based engine to execute JavaScript code.

Along with these new featuers, lots of feature enhancements are done under-the-hood, at both compiler and JVM level.

1. Question 2. How Will You Sort A List Of String Using Java 8 Lambda Expression?

Answer :

Following code sorts a list of string using Java 8 lambda expression:

//sort using java 8  
private void sortUsingJava8(List<String> names){  
  Collections.sort(names, (s1, s2) -> s1.compareTo(s2));  
}

1. Question 3. What Are The Characteristics Of A Java 8 Lambda Expression?

Answer :

A lambda expression is characterized by the following syntax -  parameter −> expression body

Following are the important characteristics of a lambda expression −

* + Optional type declaration − No need to declare the type of a parameter. The compiler can inference the same from the value of the parameter.
  + Optional parenthesis around parameter − No need to declare a single parameter in parenthesis. For multiple parameters, parentheses are required.
  + Optional curly braces − No need to use curly braces in expression body if the body contains a single statement.
  + Optional return keyword − The compiler automatically returns the value if the body has a single expression to return the value. Curly braces are required to indicate that expression returns a value.

1. Question 4. Why Lambda Expression Is To Be Used?

Answer :

Lambda expressions are used primarily to define inline implementation of a functional interface, i.e., an interface with a single method only. In the above example, we've used various types of lambda expressions to define the operation method of MathOperation interface. Then we have defined the implementation of sayMessage of GreetingService.

Lambda expression eliminates the need of anonymous class and gives a very simple yet powerful functional programming capability to Java.

1. Question 5. What Kind Of Variable You Can Access In An Lambda Expression?

Answer :

Using lambda expression, you can refer to final variable or effectively final variable (which is assigned only once). Lambda expression throws a compilation error, if a variable is assigned a value the second time.

1. Question 6. What Are Method References?

Answer :

Method references help to point to methods by their names. A method reference is described using :: (double colon) symbol. A method reference can be used to point the following types of methods −

* + Static methods
  + Instance methods
  + Constructors using new operator (TreeSet::new)

1. Question 7. Explain The System.out::println Expression?

Answer :

System.out::println method is a static method reference to println method of out object of System class.

1. Question 8. What Are Functional Interfaces?

Answer :

Functional interfaces have a single functionality to exhibit. For example, a Comparable interface with a single method 'compareTo' is used for comparison purpose. Java 8 has defined a lot of functional interfaces to be used extensively in lambda expressions.

1. Question 9. What Is The Purpose Of Biconsumer<t,u> Functional Interface?

Answer :

It represents an operation that accepts two input arguments, and returns no result.

1. Question 10. What Is The Purpose Of Bifunction<t,u,r> Functional Interface?

Answer :

It represents a function that accepts two arguments and produces a result.

1. Question 11. What Is The Purpose Of Binaryoperator<t> Functional Interface?

Answer :

It represents an operation upon two operands of the same type, producing a result of the same type as the operands.

1. Question 12. What Is The Purpose Of Bipredicate<t,u> Functional Interface?

Answer :

It represents a predicate (Boolean-valued function) of two arguments.

1. Question 13. What Is The Purpose Of Booleansupplier Functional Interface?

Answer :

It represents a supplier of Boolean-valued results.

1. Question 14. What Is The Purpose Of Consumer<t> Functional Interface?

Answer :

It represents an operation that accepts a single input argument and returns no result.

1. Question 15. What Is The Purpose Of Doublebinaryoperator Functional Interface?

Answer :

It represents an operation upon two double-valued operands and producing a double-valued result.

1. Question 16. What Is The Purpose Of Doubleconsumer Functional Interface?

Answer :

It represents an operation that accepts a single double-valued argument and returns no result.

1. Question 17. What Is The Purpose Of Doublefunction<r> Functional Interface?

Answer :

It represents a function that accepts a double-valued argument and produces a result.

1. Question 18. What Is The Purpose Of Doublepredicate Functional Interface?

Answer :

It represents a predicate (Boolean-valued function) of one double-valued argument.

1. Question 19. What Is The Purpose Of Doublesupplier Functional Interface?

Answer :

It represents a supplier of double-valued results.

1. Question 20. What Is The Purpose Of Doubletointfunction Functional Interface?

Answer :

It represents a function that accepts a double-valued argument and produces an int-valued result.

1. Question 21. What Is The Purpose Of Doubletolongfunction Functional Interface?

Answer :

It represents a function that accepts a double-valued argument and produces a long-valued result.

1. Question 22. What Is The Purpose Of Doubleunaryoperator Functional Interface?

Answer :

It represents an operation on a single double-valued operand that produces a double-valued result.

1. Question 23. What Is The Purpose Of Function<t,r> Functional Interface?

Answer :

It represents a function that accepts one argument and produces a result.

1. Question 24. What Is The Purpose Of Intbinaryoperator Functional Interface?

Answer :

It represents an operation upon two int-valued operands and produces an int-valued result.

1. Question 25. What Is The Purpose Of Intconsumer Functional Interface?

Answer :

It represents an operation that accepts a single int-valued argument and returns no result.

1. Question 26. What Is The Purpose Of Intfunction<r> Functional Interface?

Answer :

It represents a function that accepts an int-valued argument and produces a result.

1. Question 27. What Is The Purpose Of Intpredicate Functional Interface?

Answer :

It represents a predicate (Boolean-valued function) of one int-valued argument.

1. Question 28. What Is The Purpose Of Intsupplier Functional Interface?

Answer :

It represents a supplier of int-valued results.

1. Question 29. What Is The Purpose Of Inttodoublefunction Functional Interface?

Answer :

It represents a function that accepts an int-valued argument and produces a double-valued result.

1. Question 30. What Is The Purpose Of Inttolongfunction Functional Interface?

Answer :

It represents a function that accepts an int-valued argument and produces a long-valued result.

1. Question 31. What Is The Purpose Of Intunaryoperator Functional Interface?

Answer :

It represents an operation on a single int-valued operand that produces an int-valued result.

1. Question 32. What Is The Purpose Of Longbinaryoperator Functional Interface?

Answer :

It represents an operation upon two long-valued operands and produces a long-valued result.

1. Question 33. What Is The Purpose Of Longconsumer Functional Interface?

Answer :

It represents an operation that accepts a single long-valued argument and returns no result.

1. Question 34. What Is The Purpose Of Longfunction<r> Functional Interface?

Answer :

It represents a function that accepts a long-valued argument and produces a result.

1. Question 35. What Is The Purpose Of Longpredicate Functional Interface?

Answer :

It represents a predicate (Boolean-valued function) of one long-valued argument.

1. Question 36. What Is The Purpose Of Longsupplier Functional Interface?

Answer :

It represents a supplier of long-valued results.

1. Question 37. What Is The Purpose Of Longtodoublefunction Functional Interface?

Answer :

It represents a function that accepts a long-valued argument and produces a double-valued result.

1. Question 38. What Is The Purpose Of Longtointfunction Functional Interface?

Answer :

It represents a function that accepts a long-valued argument and produces an int-valued result.

1. Question 39. What Is The Purpose Of Longunaryoperator Functional Interface?

Answer :

It represents an operation on a single long-valued operand that produces a long-valued result.

1. Question 40. What Is The Purpose Of Objdoubleconsumer<t> Functional Interface?

Answer :

It represents an operation that accepts an object-valued and a double-valued argument, and returns no result.

1. Question 41. What Is The Purpose Of Objintconsumer<t> Functional Interface?

Answer :

It represents an operation that accepts an object-valued and an int-valued argument, and returns no result.

1. Question 42. What Is The Purpose Of Objlongconsumer<t> Functional Interface?

Answer :

It represents an operation that accepts an object-valued and a long-valued argument, and returns no result.

1. Question 43. What Is The Purpose Of Predicate<t> Functional Interface?

Answer :

It represents a predicate (Boolean-valued function) of one argument.

1. Question 44. What Is The Purpose Of Supplier<t> Functional Interface?

Answer :

It represents a supplier of results.

1. Question 45. What Is The Purpose Of Todoublebifunction<t,u> Functional Interface?

Answer :

It represents a function that accepts two arguments and produces a double-valued result.

1. Question 46. What Is The Purpose Of Todoublefunction<t> Functional Interface?

Answer :

It represents a function that produces a double-valued result.

1. Question 47. What Is The Purpose Of Tointbifunction<t,u> Functional Interface?

Answer :

It represents a function that accepts two arguments and produces an int-valued result.

1. Question 48. What Is The Purpose Of Tointfunction<t> Functional Interface?

Answer :

It represents a function that produces an int-valued result.

1. Question 49. What Is The Purpose Of Tolongbifunction<t,u> Functional Interface?

Answer :

It represents a function that accepts two arguments and produces a long-valued result.

1. Question 50. What Is The Purpose Of Tolongfunction<t> Functional Interface?

Answer :

It represents a function that produces a long-valued result.

1. Question 51. What Is The Purpose Of Unaryoperator<t> Functional Interface?

Answer :

It represents an operation on a single operand that produces a result of the same type as its operand.

1. Question 52. What Are Default Methods?

Answer :

With java 8, an interface can have default implementation of a function in interfaces.

1. Question 53. What Are Static Default Methods?

Answer :

An interface can also have static helper methods from Java 8 onwards.

public interface vehicle {  
   default void print(){  
      System.out.println("I am a vehicle!");  
   }  
   static void blowHorn(){  
      System.out.println("Blowing horn!!!");  
   }  
}

1. Question 54. How Will You Call A Default Method Of An Interface In A Class?

Answer :

Using super keyword along with interface name.

interface Vehicle {  
   default void print(){  
      System.out.println("I am a vehicle!");  
   }  
}  
class Car implements Vehicle {  
   public void print(){  
      Vehicle.super.print();                    
   }  
}

1. Question 55. How Will You Call A Static Method Of An Interface In A Class?

Answer :

Using name of the interface.

interface Vehicle {  
   static void blowHorn(){  
      System.out.println("Blowing horn!!!");  
   }  
}  
class Car implements Vehicle {  
   public void print(){  
      Vehicle.blowHorn();                    
   }  
}

1. Question 56. What Is Streams In Java 8?

Answer :

Stream represents a sequence of objects from a source, which supports aggregate operations.

1. Question 57. What Is Stream Pipelining In Java 8?

Answer :

Most of the stream operations return stream itself so that their result can be pipelined. These operations are called intermediate operations and their function is to take input, process them, and return output to the target. collect() method is a terminal operation which is normally present at the end of the pipelining operation to mark the end of the stream.

1. Question 58. What Is The Difference Between Collections And Stream In Java8 ?

Answer :

Stream operations do the iterations internally over the source elements provided, in contrast to Collections where explicit iteration is required.

1. Question 59. What Is The Purpose Of Foreach Method Of Stream In Java 8?

Answer :

Stream has provided a new method 'forEach' to iterate each element of the stream.

1. Question 60. How Will You Print 10 Random Numbers Using Foreach Of Java 8?

Answer :

The following code segment shows how to print 10 random numbers using forEach.

Random random = new Random();  
random.ints().limit(10).forEach(System.out::println);

1. Question 61. What Is The Purpose Of Map Method Of Stream In Java 8?

Answer :

The 'map' method is used to map each element to its corresponding result.

1. Question 62. How Will You Print Unique Squares Of Numbers In Java 8?

Answer :

The following code segment prints unique squares of numbers using map.

List<Integer> numbers = Arrays.asList(3, 2, 2, 3, 7, 3, 5);

//get list of unique squares

List<Integer> squaresList = numbers.stream().map( i -> i\*i).distinct().collect(Collectors.toList());

1. Question 63. What Is The Purpose Of Filter Method Of Stream In Java 8?

Answer :

The 'filter' method is used to eliminate elements based on a criteria.

1. Question 64. How Will You Print Count Of Empty Strings In Java 8?

Answer :

The following code segment prints a count of empty strings using filter.

List<String>strings = Arrays.asList("abc", "", "bc", "efg", "abcd","", "jkl");  
//get count of empty string  
int count = strings.stream().filter(string −> string.isEmpty()).count();

1. Question 65. What Is The Purpose Of Limit Method Of Stream In Java 8?

Answer :

The 'limit' method is used to reduce the size of the stream.

1. Question 66. How Will You Print 10 Random Numbers In Java 8?

Answer :

The following code segment shows how to print 10 random numbers.

Random random = new Random();  
random.ints().limit(10).forEach(System.out::println);

1. Question 67. What Is The Purpose Of Sorted Method Of Stream In Java 8?

Answer :

The 'sorted' method is used to sort the stream.

1. Question 68. How Will You Print 10 Random Numbers In A Sorted Order In Java 8?

Answer :

The following code segment shows how to print 10 random numbers in a sorted order.

Random random = new Random();  
random.ints().limit(10).sorted().forEach(System.out::println);

1. Question 69. What Is Parallel Processing In Java 8?

Answer :

parallelStream is the alternative of stream for parallel processing. Take a look at the following code segment that prints a count of empty strings using parallelStream.

List<String> strings = Arrays.asList("abc", "", "bc", "efg", "abcd","", "jkl");  
//get count of empty string  
int count = strings.parallelStream().filter(string −> string.isEmpty()).count();  
//It is very easy to switch between sequential and parallel streams.

1. Question 70. What Are Collectors In Java 8?

Answer :

Collectors are used to combine the result of processing on the elements of a stream. Collectors can be used to return a list or a string.

List<String>strings = Arrays.asList("abc", "", "bc", "efg", "abcd","", "jkl");  
List<String> filtered = strings.stream().filter(string -> !string.isEmpty()).collect(Collectors.toList());  
System.out.println("Filtered List: " + filtered);  
String mergedString = strings.stream().filter(string -> !string.isEmpty()).collect(Collectors.joining(", "));  
System.out.println("Merged String: " + mergedString);

1. Question 71. What Are Statistics Collectors In Java 8?

Answer :

With Java 8, statistics collectors are introduced to calculate all statistics when stream processing is being done.

1. Question 72. How Will You Get The Highest Number Present In A List Using Java 8?

Answer :

Following code will print the highest number present in a list.

List numbers = Arrays.asList(3, 2, 2, 3, 7, 3, 5);  
IntSummaryStatistics stats = integers.stream().mapToInt((x) −> x).summaryStatistics();  
System.out.println("Highest number in List : " + stats.getMax());

1. Question 73. How Will You Get The Lowest Number Present In A List Using Java 8?

Answer :

Following code will print the highest number present in a list.

List numbers = Arrays.asList(3, 2, 2, 3, 7, 3, 5);  
IntSummaryStatistics stats = integers.stream().mapToInt((x) −> x).summaryStatistics();  
System.out.println("Lowest number in List : " + stats.getMin());

1. Question 74. How Will You Get The Sum Of All Numbers Present In A List Using Java 8?

Answer :

Following code will print the sum of all numbers present in a list.

List numbers = Arrays.asList(3, 2, 2, 3, 7, 3, 5);  
IntSummaryStatistics stats = integers.stream().mapToInt((x) −> x).summaryStatistics();  
System.out.println("Sum of all numbers : " + stats.getSum());

1. Question 75. How Will You Get The Average Of All Numbers Present In A List Using Java 8?

Answer :

Following code will print the average of all numbers present in a list.

List numbers = Arrays.asList(3, 2, 2, 3, 7, 3, 5);  
IntSummaryStatistics stats = integers.stream().mapToInt((x) −> x).summaryStatistics();  
System.out.println("Average of all numbers : " + stats.getAverage());

1. Question 76. What Is Optional In Java8?

Answer :

Optional is a container object which is used to contain not-null objects. Optional object is used to represent null with absent value. This class has various utility methods to facilitate code to handle values as 'available' or 'not available' instead of checking null values. It is introduced in Java 8 and is similar to what Optional is in Guava.

1. Question 77. What Is Nashorn In Java8?

Answer :

With Java 8, Nashorn, a much improved javascript engine is introduced, to replace the existing Rhino. Nashorn provides 2 to 10 times better performance, as it directly compiles the code in memory and passes the bytecode to JVM. Nashorn uses invokedynamics feature, introduced in Java 7 to improve performance.

1. Question 78. What Is Jjs In Java8?

Answer :

For Nashorn engine, JAVA 8 introduces a new command line tool, jjs, to execute javascript codes at console.

1. Question 79. Can You Execute Javascript Code From Java 8 Code Base?

Answer :

Yes! Using ScriptEngineManager, JavaScript code can be called and interpreted in Java.

1. Question 80. What Is Local Datetime Api In Java8?

Answer :

Local − Simplified date-time API with no complexity of timezone handling.

1. Question 81. What Is Zoned Datetime Api In Java8?

Answer :

Zoned − Specialized date-time API to deal with various timezones.

1. Question 82. What Is Chromounits In Java8?

Answer :

java.time.temporal.ChronoUnit enum is added in Java 8 to replace the integer values used in old API to represent day, month, etc.

1. Question 83. How Will You Get The Current Date Using Local Datetime Api Of Java8?

Answer :

Following code gets the current date using local datetime api −

//Get the current date  
LocalDate today = LocalDate.now();  
System.out.println("Current date: " + today);

1. Question 84. How Will You Add 1 Week To Current Date Using Local Datetime Api Of Java8?

Answer :

Following code adds 1 week to current date using local datetime api −

//add 1 week to the current date  
LocalDate today = LocalDate.now();  
LocalDate nextWeek = today.plus(1, ChronoUnit.WEEKS);  
System.out.println("Next week: " + nextWeek);

1. Question 85. How Will You Add 1 Month To Current Date Using Local Datetime Api Of Java8?

Answer :

Following code adds 1 month to current date using local datetime api:

//add 1 month to the current date  
LocalDate today = LocalDate.now();  
LocalDate nextMonth = today.plus(1, ChronoUnit.MONTHS);  
System.out.println("Next month: " + nextMonth);

1. Question 86. How Will You Add 1 Year To Current Date Using Local Datetime Api Of Java8?

Answer :

Following code adds 1 year to current date using local datetime api −

//add 1 year to the current date  
LocalDate today = LocalDate.now();  
LocalDate nextYear = today.plus(1, ChronoUnit.YEARS);  
System.out.println("Next year: " + nextYear);

1. Question 87. How Will You Add 10 Years To Current Date Using Local Datetime Api Of Java8?

Answer :

Following code adds 10 years to current date using local datetime api −

//add 10 years to the current date  
LocalDate today = LocalDate.now();  
LocalDate nextDecade = today.plus(1, ChronoUnit.DECADES);  
System.out.println("Date after ten year: " + nextDecade);

1. Question 88. How Will You Get Next Tuesday Using Java8?

Answer :

Following code gets next tuesday using java8 −

//get the next tuesday  
LocalDate today = LocalDate.now();  
LocalDate nextTuesday = today.with(TemporalAdjusters.next(DayOfWeek.TUESDAY));  
System.out.println("Next Tuesday on : " + nextTuesday);

1. Question 89. How Will You Get Second Saturday Of Next Month Using Java8?

Answer :

Following code gets second saturday of next month using java8 −

//get the second saturday of next month

LocalDate firstInYear = LocalDate.of(date1.getYear(),date1.getMonth(), 1);

LocalDate secondSaturday = firstInYear.with(TemporalAdjusters.nextOrSame(DayOfWeek.SATURDAY)).with(TemporalAdjusters.next(DayOfWeek.SATURDAY));

System.out.println("Second Saturday on : " + secondSaturday);

1. Question 90. How Will You Get The Instant Of Current Date In Terms Of Milliseconds Using Java8?

Answer :

Following code gets the instant of current date in terms of milliseconds −

//Get the instant of current date in terms of milliseconds

Instant now = currentDate.toInstant();

1. Question 91. How Will You Get The Instant Of Local Date Time Using Time In Of Milliseconds Using Java8?

Answer :

Following code gets the instant of local date time using time in of milliseconds −

Instant now = currentDate.toInstant();  
ZoneId currentZone = ZoneId.systemDefault();  
LocalDateTime localDateTime = LocalDateTime.ofInstant(now, currentZone);  
System.out.println("Local date: " + localDateTime);

1. Question 92. How Will You Get The Instant Of Zoned Date Time Using Time In Of Milliseconds Using Java8?

Answer :

Following code gets the instant of zoned date time using time in of milliseconds −

Instant now = currentDate.toInstant();  
ZoneId currentZone = ZoneId.systemDefault();  
ZonedDateTime zonedDateTime = ZonedDateTime.ofInstant(now, currentZone);  
System.out.println("Zoned date: " + zonedDateTime);

1. Question 93. Which Class Implements A Decoder For Decoding Byte Data Using The Base64 Encoding Scheme In Java8?

Answer :

static class Base64.Decoder − This class implements a decoder for decoding byte data using the Base64 encoding scheme as specified in RFC 4648 and RFC 2045.

1. Question 94. Which Class Implements An Encoder For Encoding Byte Data Using The Base64 Encoding Scheme In Java8?

Answer :

static class Base64.Encoder − This class implements an encoder for encoding byte data using the Base64 encoding scheme as specified in RFC 4648 and RFC 2045.

1. Question 95. How Will You Create A Base64 Decoder?

Answer :

getDecoder() method of Base64 class returns a Base64.Decoder that decodes using the Basic type base64 encoding scheme.

1. Question 96. How Will You Create A Base64 Encoder?

Answer :

getEncoder() method of Base64 class returns a Base64.Encoder that encodes using the Basic type base64 encoding scheme.

1. Question 97. How Will You Create A Base64 Decoder That Decodes Using The Mime Type Base64 Encoding Scheme?

Answer :

getMimeDecoder() method of Base64 class returns a Base64.Decoder that decodes using the MIME type base64 decoding scheme.

1. Question 98. How Will You Create A Base64 Encoder That Encodes Using The Mime Type Base64 Encoding Scheme?

Answer :

getMimeEncoder() method of Base64 class returns a Base64.Encoder that encodes using the MIME type base64 encoding scheme.

1. Question 99. How Will You Create A Base64 Decoder That Decodes Using The Url And Filename Safe Type Base64 Encoding Scheme?

Answer :

getUrlDecoder() method of Base64 class returns a Base64.Decoder that decodes using the URL and Filename safe type base64 encoding scheme.

1. Question 100. How Will You Create A Base64 Encoder That Encodes Using The Url And Filename Safe Type Base64 Encoding Scheme?

Answer :

getUrlEncoder() method of Base64 class returns a Base64.Encoder that encodes using the URL and Filename safe type base64 encoding scheme.

## 1. What new features were added in Java 8?

Java 8 ships with several new features and enhancements but the most significant are the following:

* [Lambda Expressions](http://www.javaguides.net/2018/07/java-8-lambda-expressions.html) − a new language feature allowing treating actions as objects
* [Method References](http://www.javaguides.net/2018/07/java-8-method-references.html) − enable defining Lambda Expressions by referring to methods directly using their names
* [Optional](http://www.javaguides.net/2018/07/java-8-optional-class.html) − This class is to provide a type-level solution for representing optional values instead of using null references.
* [Functional Interface](http://www.javaguides.net/2018/07/java-8-functional-interfaces.html) – an interface with a maximum one abstract method, implementation can be provided using a Lambda Expression
* [Default methods](http://www.javaguides.net/2018/07/java-8-static-and-default-methods-in-interface.html) − give us the ability to add full implementations in interfaces besides abstract methods
* Nashorn, JavaScript Engine − Java-based engine for executing and evaluating JavaScript code
* [Stream API](http://www.javaguides.net/2018/07/java-8-stream-api.html) − a special iterator class that allows processing collections of objects in a functional manner
* [Date and Time API](http://www.javaguides.net/p/java-8-date-time-api-tutorial.html) − an improved, immutable JodaTime-inspired Date API

Along with these new features, lots of feature enhancements are done under-the-hood, at both compiler and JVM level.

The below diagram shows all the Java 8 features and enhancements.

[](https://2.bp.blogspot.com/-sKDGd0GE1Bg/W-kqukrGoBI/AAAAAAAAEoU/bdcYGfAWJbYIVihusyAL2hpCxweuh7ANgCLcBGAs/s1600/java-8-features-vertical.png)

## 2. What is a Lambda Expression?

In very simple terms, a lambda expression is a function that can be referenced and passed around as an object.

Lambda expressions introduce functional style processing in Java and facilitate the writing of compact and easy-to-read code.

Because of this, lambda expressions are a natural replacement for anonymous classes as method arguments. One of their main uses is to define inline implementations of functional interfaces.

Read more in-detail about lambda expressions at [Java 8 Lambda Expressions](http://www.javaguides.net/2018/07/java-8-lambda-expressions.html)

## 3. Why use Lambda Expression?

1. To provide the implementation of the [Java 8 Functional Interface](http://www.javaguides.net/2018/07/java-8-functional-interfaces.html).
2. Less coding - lambda expressions are a natural replacement for anonymous classes as a method argument
3. Lambda Expressions enable you to encapsulate a single unit of behavior and pass it to other code. You can use lambda expressions if you want a certain action performed on each element of a collection, when a process is completed, or when a process encounters an error.

Read more in-detail about lambda expressions at [Java 8 Lambda Expressions](http://www.javaguides.net/2018/07/java-8-lambda-expressions.html).

## 4. Explain Lambda Expression Syntax

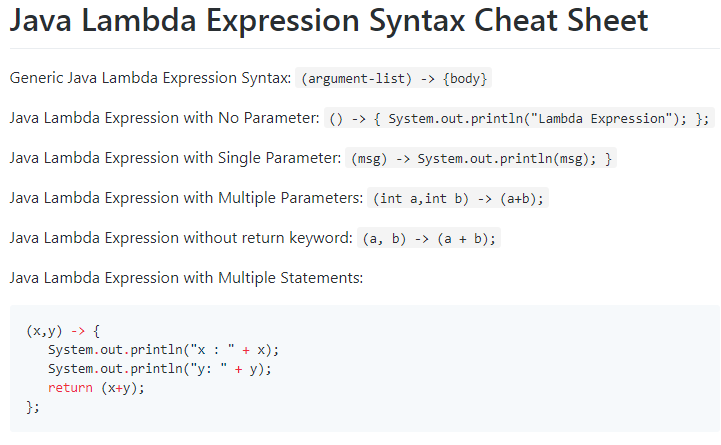
Java Lambda Expression Syntax

(argument-list) -> {body}

Java lambda expression consists of three components.

* Argument-list: It can be empty or non-empty as well.
* Arrow-token: It is used to link arguments-list and body of expression.
* Body: It contains expressions and statements for the lambda expression.

The below diagram shows the important characteristics of a lambda expression.

[](https://1.bp.blogspot.com/-VzrfkEg1s88/W-krSa__kbI/AAAAAAAAEoc/tpbwGMuxJ5AMLoP6mP3F9TEr8ff5z8lKwCLcBGAs/s1600/lambda-expressions.png)

Read more in-detail about lambda expressions at [Java 8 Lambda Expressions](http://www.javaguides.net/2018/07/java-8-lambda-expressions.html).

## 5. What is a functional interface?

An Interface that contains exactly one abstract method is known as a functional interface. It can have any number of default, static methods but can contain only one abstract method. It can also declare methods of the object class.

Functional Interface is also known as Single Abstract Method Interfaces or SAM Interfaces. A functional interface can extend another interface only when it does not have any abstract method.

Java 8 provides predefined functional interfaces to deal with functional programming by using lambda and method references. For example, below Java program to illustrate Predicate functional interface usage.

// Java program to illustrate Simple Predicate

import java.util.function.Predicate;

public class PredicateInterfaceExample {

public static void main(String[] args)

{

// Creating predicate

Predicate<Integer> lesserthan = i -> (i < 18);

// Calling Predicate method

System.out.println(lesserthan.test(10));

}

}

Output

true

Read more at [Java 8 Functional Interfaces with Examples](http://www.javaguides.net/2018/07/java-8-functional-interfaces.html).

## 6. Is it possible to define our own Functional Interface? What is @FunctionalInterface? What are the rules to define a Functional Interface?

Yes, it is possible to define our own Functional Interfaces. We use Java SE 8’s @FunctionalInterface annotation to mark an interface as Functional Interface. We need to follow these rules to define a Functional Interface:

* Define an interface with one and only one abstract method.
* We cannot define more than one abstract method.
* Use @FunctionalInterface annotation in the interface definition.
* We can define any number of other methods like Default methods, Static methods.
* If we override java.lang.Object class’s method as an abstract method, which does not count as an abstract method.

Below example illustrate the defining our own Functional Interface:

Let's create Sayable interface annotated with @FunctionalInterface annotation.

@FunctionalInterface

interface Sayable{

void say(String msg); // abstract method

}

Let's demonstrate a custom functional interface via the main() method.

public class FunctionalInterfacesExample {

public static void main(String[] args) {

Sayable sayable = (msg) -> {

System.out.println(msg);

};

sayable.say("Say something ..");

}

}

Read more at [Java 8 Functional Interfaces with Examples](http://www.javaguides.net/2018/07/java-8-functional-interfaces.html).

## 7. Describe some of the functional interfaces in the standard library.

There are a lot of functional interfaces in the java.util.function package, the more common ones include but not limited to:

* Function – it takes one argument and returns a result
* Consumer – it takes one argument and returns no result (represents a side effect)
* Supplier – it takes no argument and returns a result
* Predicate – it takes one argument and returns a boolean
* BiFunction – it takes two arguments and returns a result
* BinaryOperator – it is similar to a BiFunction, taking two arguments and returning a result. The two arguments and the result are all of the same types
* UnaryOperator – it is similar to a Function, taking a single argument and returning a result of the same type

For more on functional interfaces, see the article at [Java 8 Functional Interfaces with Examples](http://www.javaguides.net/2018/07/java-8-functional-interfaces.html).

## 8. What is a functional interface? What are the rules for defining a functional interface?

An Interface that contains exactly one abstract method is known as a functional interface.

We need to follow these rules to define a Functional Interface:

* Define an interface with one and only one abstract method.
* We cannot define more than one abstract method.
* Use @FunctionalInterface annotation in the interface definition.
* We can define any number of other methods like Default methods, Static methods.
* If we override java.lang.Object class’s method as an abstract method, which does not count as an abstract method.

## 9. What is a method reference?

Method reference is used to refer method of the [functional interface](http://www.javaguides.net/2018/07/java-8-functional-interfaces.html). It is a compact and easy form of a lambda expression. Each time when you are using a lambda expression to just referring a method, you can replace your lambda expression with method reference.

Below are a few examples of method reference:

(o) -> o.toString();

can become:

Object::toString();

A method reference can be identified by a double colon separating a class or object name and the name of the method. It has different variations such as constructor reference:

String::new;

Static method reference:

String::valueOf;

Bound instance method reference:

str::toString;

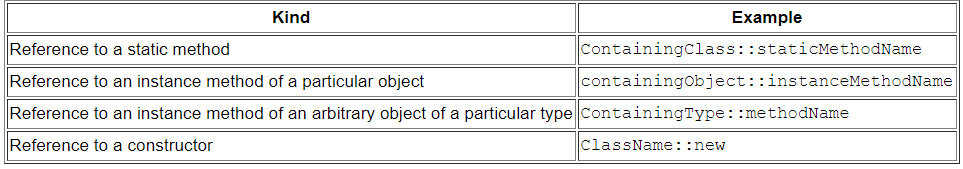
Unbound instance method reference:

String::toString;

You can read a detailed description of method references with full examples at [Java 8 Method References](http://www.javaguides.net/2018/07/java-8-method-references.html).

## 10. What are different kinds of Method References?

There are four kinds of method references:

[](https://3.bp.blogspot.com/-KH4sS24R_JY/W-ksfabgaeI/AAAAAAAAEoo/KLGF1UB90Ek6Euu2VCZl8ct9piYcHS9yACLcBGAs/s1600/method-references.PNG)

1. Reference to a static method. For example:

ContainingClass::staticMethodName

1. Reference to an instance method of a particular object. For example:

containingObject::instanceMethodName

1. Reference to an instance method of an arbitrary object of a particular type. For example:

ContainingType::methodName

1. Reference to a constructor. for example:

ClassName::new

You can read a detailed description of method references with full examples at [Java 8 Method References](http://www.javaguides.net/2018/07/java-8-method-references.html).

## 11. What is a streams in Java 8? How does it differ from a collection?

In simple terms, a stream is an iterator whose role is to accept a set of actions to apply to each of the elements it contains.

The stream represents a sequence of objects from a source such as a collection, which supports aggregate operations. They were designed to make collection processing simple and concise. Contrary to the collections, the logic of iteration is implemented inside the stream, so we can use methods like map and flatMap for performing a declarative processing.

Another difference is that the Stream API is fluent and allows pipelining:

int sum = Arrays.stream(new int[]{1, 2, 3})

.filter(i -> i >= 2)

.map(i -> i \* 3)

.sum();

And yet another important distinction from collections is that streams are inherently lazily loaded and processed.

Read more about streams at [Java 8 Stream APIs with Examples](http://www.javaguides.net/2018/07/java-8-stream-api.html).

## 12. What is stream pipelining in Java 8?

Stream pipelining is the concept of chaining operations together. This is done by splitting the operations that can happen on a stream into two categories: intermediate operations and terminal operations.

Each intermediate operation returns an instance of Stream itself when it runs, an arbitrary number of intermediate operations can, therefore, be set up to process data forming a processing pipeline.

There must then be a terminal operation that returns a final value and terminates the pipeline.

The following example prints the male members contained in the collection roster with a pipeline that consists of the aggregate operations filter and forEach:

roster

.stream()

.filter(e -> e.getGender() == Person.Sex.MALE)

.forEach(e -> System.out.println(e.getName()));

Read more at [Collection Aggregate Operations using Streams](http://www.javaguides.net/2018/07/collections-aggregate-operations.html)

## 13. Explain Differences between Collection API and Stream API?

[](https://4.bp.blogspot.com/-hLGWEA3pa20/W-kvTEwHRQI/AAAAAAAAEpA/qqkuB4DXRPgMRXEpEZdjLn4clR5TP8MeQCLcBGAs/s1600/difference-between-collection-stream.PNG)

## 14. What is Optional in Java 8?

Optional is a container object which is used to contain not-null objects. An optional object is used to represent null with absent value. This class has various utility methods to facilitate code to handle values as 'available' or 'not available' instead of checking null values. It is introduced in **Java 8** and is similar to what Optional is in Guava.

The purpose of the Optional class is to provide a type-level solution for representing optional values instead of using null references.

Read more about Optional class with examples at [**Java 8 Optional Class with Examples**](http://www.javaguides.net/2018/07/java-8-optional-class.html).

## 15. What are the Advantages of Java 8 Optional?

* Null checks are not required.
* No more **NullPointerException** at run-time.
* We can develop a clean and neat APIs.
* No more Boilerplate code

Read more about Optional class with examples at [**Java 8 Optional Class with Examples**](http://www.javaguides.net/2018/07/java-8-optional-class.html).

## 16. What are Collectors Class in Java 8?

*Collectors* is a final class that extends *Object* class. It provides reduction operations, such as accumulating elements into collections, summarizing elements according to various criteria, etc.

The following are examples of using the predefined collectors to perform common mutable reduction tasks:

// Accumulate names into a List

List<String> list = people.stream().map(Person::getName).collect(Collectors.toList());

// Accumulate names into a TreeSet

Set<String> set = people.stream().map(Person::getName).collect(Collectors.toCollection(TreeSet::new));

// Convert elements to strings and concatenate them, separated by commas

String joined = things.stream()

.map(Object::toString)

.collect(Collectors.joining(", "));

// Compute sum of salaries of employee

int total = employees.stream()

.collect(Collectors.summingInt(Employee::getSalary)));

// Group employees by department

Map<Department, List<Employee>> byDept

= employees.stream()

.collect(Collectors.groupingBy(Employee::getDepartment));

// Compute sum of salaries by department

Map<Department, Integer> totalByDept

= employees.stream()

.collect(Collectors.groupingBy(Employee::getDepartment,

Collectors.summingInt(Employee::getSalary)));

// Partition students into passing and failing

Map<Boolean, List<Student>> passingFailing =

students.stream()

.collect(Collectors.partitioningBy(s -> s.getGrade() >= PASS\_THRESHOLD));

Read more about Collectors class at [**Java 8 Collectors Class with Examples**](http://www.javaguides.net/2018/07/java-8-collectors-class.html).

## 17. What is the use of Java 8 StringJoiner Class?

Java added a new final class StringJoiner in java.util package. It is used to construct a sequence of characters separated by a delimiter. Now, we can create a string by passing delimiters like a comma(,), hyphen(-) etc.

Example: Simple Delimiters Example

private static void delimiterDemonstration() {

StringJoiner joinNames = new StringJoiner(","); // passing comma(,) as delimiter

// Adding values to StringJoiner

joinNames.add("Rahul");

joinNames.add("Raju");

joinNames.add("Peter");

joinNames.add("Raheem");

System.out.println(joinNames);

joinNames = new StringJoiner("|"); // passing comma(,) as delimiter

// Adding values to StringJoiner

joinNames.add("Rahul");

joinNames.add("Raju");

joinNames.add("Peter");

joinNames.add("Raheem");

System.out.println(joinNames);

}

Read more about *StringJoiner* class at [**Java 8 StringJoiner Class with Examples**](http://www.javaguides.net/2018/07/java-8-stringjoiner-class.html).

## 18. What is a default method and when do we use it?

A default method is a method with an implementation – which can be found in an interface.

We can use a default method to add a new functionality to an interface while maintaining backward compatibility with classes that are already implementing the interface:

public interface Vehicle {

String getBrand();

String speedUp();

String slowDown();

default String turnAlarmOn() {

return "Turning the vehice alarm on.";

}

default String turnAlarmOff() {

return "Turning the vehicle alarm off.";

}

}

Usually, when a new abstract method is added to an interface, all implementing classes will break until they implement the new abstract method. In Java 8, this problem has been solved by the use of the default method.

For example, the Collection interface does not have a **[forEach](http://www.javaguides.net/2018/06/guide-to-java-8-foreach-method.html" \t "_blank)** method declaration. Thus, adding such a method would simply break the whole collections API.

Java 8 introduces the default method so that the Collection interface can have a default implementation of the **[forEach method](http://www.javaguides.net/2018/06/guide-to-java-8-foreach-method.html" \t "_blank)** without requiring the classes implementing this interface to implement the same.

Read more about Default Methods with examples at [**Java 8 Static and Default Methods in Interface**](http://www.javaguides.net/2018/07/java-8-static-and-default-methods-in-interface.html).

## 19. Why Default Methods in Interfaces Are Needed?

* Default methods enable you to add new functionality to the interfaces of your libraries and ensure binary compatibility with code written for older versions of those interfaces.
* In a typical design based on abstractions, where an interface has one or multiple implementations, if one or more methods are added to the interface, all the implementations will be forced to implement them too. Otherwise, the design will just break down so default interface methods are an efficient way to deal with this issue. They allow us to add new methods to an interface that are automatically available in the implementations. Thus, there’s no need to modify the implementing classes. In this way, backward compatibility is neatly preserved without having to refactor the implementers.
* The default method is used to define a method with a default implementation. You can override the default method also to provide the more specific implementation for the method.

Read more about Default Methods with examples at [**Java 8 Static and Default Methods in Interface**](http://www.javaguides.net/2018/07/java-8-static-and-default-methods-in-interface.html).

## 20. What is the use of static methods in Interface?

The idea behind static interface methods is to provide a simple mechanism that allows us to increase the degree of cohesion of a design by putting together related methods in one single place without having to create an object.

Furthermore, static methods in interfaces make it possible to group related utility methods, without having to create artificial utility classes that are simply placeholders for static methods.

For example:

public interface Vehicle {

String getBrand();

String speedUp();

String slowDown();

default String turnAlarmOn() {

return "Turning the vehice alarm on.";

}

default String turnAlarmOff() {

return "Turning the vehicle alarm off.";

}

static int getHorsePower(int rpm, int torque) {

return (rpm \* torque) / 5252;

}

}

## 21. How will you call a default method of an interface in a class?

Using the *super* keyword along with the interface name.

interface Vehicle {

default void print() {

System.out.println("I am a vehicle!");

}

}

class Car implements Vehicle {

public void print() {

Vehicle.super.print();

}

}

## 22. How will you call a static method of an interface in a class?

Using the name of the interface.

interface Vehicle {

static void blowHorn() {

System.out.println("Blowing horn!!!");

}

}

class Car implements Vehicle {

public void print() {

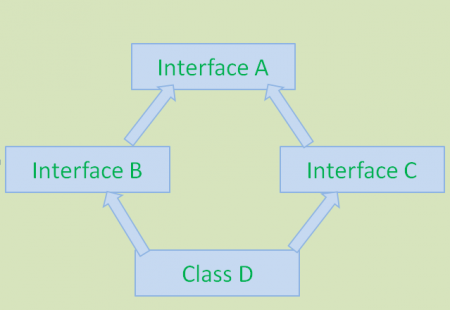
Vehicle.blowHorn();

}

}

## 23. What is Diamond Problem in Inheritance? How Java 8 Solves this problem?

A Diamond Problem is a Multiple Inheritance problem. In Java, It occurs when a Class extends more than one Interface which have the same method implementation (Default method).

**[](https://3.bp.blogspot.com/-LoSnCb_yK3g/W-kub5E-VkI/AAAAAAAAEo0/7S9Hr8PAsu0xiR7o3t6fyeDLgJppeC7ugCLcBGAs/s1600/diamond_problem-450x310.png)**

This above diagram shows Diamond Problem. To avoid this problem, **Java 7** and earlier versions do not support methods implementation in interface and also doesn’t support Multiple Inheritance. Java 8 has introduced a new feature: Default methods to support Multiple Inheritance with some limitations.

Sample Java SE 8 Code to show this Diamond Problem:

public interface A{

default void display() { //code goes here }

}

public interface B extends A{ }

public interface C extends A{ }

public class D implements B,C{ }

In the above code snippet, class D gives compile-time errors because Java Compiler will get a bit confused about which *display(*) has to provide in-class D. Class D inherits the *display()* method from both interfaces B and C. To solve this problem, Java SE 8 has given the following remedy:

public interface A{

default void display() { //code goes here }

}

public interface B extends A{ }

public interface C extends A{ }

public class D implements B,C{

void display() {

B.super.display();

}

}

This *B.super.display();* will solve this Diamond Problem.

## 24. What is Nashorn in Java8?

**Nashorn** is the new Javascript processing engine for the Java platform that shipped with Java 8. Until JDK 7, the Java platform used Mozilla Rhino for the same purpose. as a Javascript processing engine.

**Nashorn** provides better compliance with the ECMA normalized JavaScript specification and better runtime performance than its predecessor.

## 25. What are Java 8 Date-Time API Benefits over Old Date and Calendar Classes?

Here are some of the challenges faced by developers with earlier Java versions:

1. Poor API design (for example, months start with 1 and days start with 0)
2. Not thread-safe
3. No consistency within the java.util and java.sql classes
4. No support for internationalization and time zones

The new Date and Time API was introduced in Java SE 8 with the following solutions:

1. The improved API design clearly separates the human-readable date time and machine time.
2. The new Date and Time API makes all the classes immutable, which is suitable for the multithreaded environment.
3. There is more clarity in the API design level. The methods are clearly defined and perform the same action in all classes.
4. The API defines separate classes for Date, Time, DateTime, Timestamp, TimeZone, and so on. The new Date and Time API works with the ISO-8601 calendar. However, you can use the API with non-ISO calendars as well.
   1. **What are the new features that are added in** [**Java**](https://careerkaizen.com/core-java-interview-questions/) **8?**  
      **Answer:** Java 8 ships with several new features and enhancements but the most significant are the following:

* Lambda Expressions – It is termed as the new language feature and it allows the treating actions as the objects.
* Method References – It helps in enabling to define the Lambda Expressions by referring to the various methods directly by using their names.
* Optional – This is a class which is to provide a type-level solution for the representation of the optional values instead of using the null references.
* Functional Interface – This is an interface which has maximum one abstract method and the implementation can be provided using a Lambda Expression for this interface.
* Default methods – These methods are the methods which gives one’s the ability to add the full implementations in interfaces.
* Nashorn, JavaScript Engine – It is known as the Java-based engine for the execution and evaluation of the JavaScript code.
* Stream API – It is termed as the special iterator class which allows in processing collections of objects in a functional manner.
* Date and Time API – It is an improved, immutable JodaTime-inspired Date API. Besides all these new features all others feature are in the path of compiler and the JVM also.

**2. What makes Java SE 8 superior over others?**  
**Answer:** The features are as follows which makes it superior over others in Java SE 8.

* It writes the parallel code.
* It offers even more useable codes.
* It has improved performance applications.
* It has more readable and concise codes.
* It supports writing database including promotions.

**3. Do we need changes and reforms in Java?**  
**Answer:** Yes, we do need changes and reforms in Java language because of the following reasons.

1. To make the best and efficient use of multi-million CPUs that are deployed at work throughout the world.
2. To meet with the changes and up-gradation made in the technology hardware and software.
3. To support the running of various kinds of applications.
4. To create highly concurrent and scalable applications.
5. To make use of all the functional programming features which are an integral part of the latest Java SE 8 computer language.

**4. What is a Lambda Expression?**

**Answer:** We can say that, a lambda expression is a function which can be referenced and passed around as an object. Lambda expressions are a function that introduces functional style processing in Java and helps in facilitating the writing of compact and easy-to-read code.

Due to this the lambda expressions are simply and natural replacement for same or anonymous classes as method arguments and one of their main uses is defining the inline implementations of the functional interfaces.

**5. Define the main parts of a Lambda expression in Java Programming Language?**  
**Answer:** Three main parts of a Lambda expression are:

* Parameter list: As the Lambda expression can have zero or more parameters. Parameter list is optional in case of Lambda.
* Lambda arrow operator: It is termed as the Lambda arrow operator. It helps in separating the list of parameters and the body of Lambda.
* Lambda expression body: The piece of code which the user wants to execute is written in the Lambda expression body.

**6. Why we use Lambda Expression?**  
**Answer:**

* It helps in providing the implementation of the Java 8 Functional Interface.
* As we can see that the lambda expressions are a natural replacement for same or anonymous classes as a method argument.
* Lambda Expressions helps in encapsulating by enabling user to encapsulate a single unit of behavior and then pass it to other code. We can use lambda expressions if we need a certain action performed on each element of a collection, when a process is completed, or when a process encounters an error.

**7. What is the main theme behind the release of a new version of Java like Java 8?**  
**Answer:** The main theme of Java 8 is its support for the functional programming. With increase in Database size and growth of multi-code CPU servers, there is need for Java to support such large-scale systems. With new features of Java 8, it is possible to create functional programs to interact efficiently with Big Data systems. Support for Streams is very helpful in this regard.

Lambda expressions are very useful for cloud computing where we can pass code as data and run the same code on multiple servers. Optional is a best practice that is borrowed from Google Guava library for handling the exceptional cases. This has made programs more robust with support for edge cases.

**8. What is a Functional Interface?**  
**Answer:** An Interface is termed as the functional interface if it contains exactly one abstract method. It can have any number of default, static methods but it contains only one abstract method. It also helps in declaring methods of the object class.

Functional Interface is also known as Single Abstract Method Interfaces or SAM Interfaces. A functional interface has the ability of extending another interface only when it does not have any abstract method. Java 8 helps in providing the predefined functional interfaces to deal with functional programming by making use of lambda and method references.

**9. Can we define our own Functional Interface in Java? What is a Functional Interface? What** **are the rules needed to define a Functional Interface?**  
**Answer:** Yes, we can define our own Functional Interfaces in Java. We make use of the Java SE 8’s @ Functional Interface annotation for marking an interface as the Functional Interface. We need to follow some rules for defining a Functional Interface:

* We are able to define an interface with one and only one abstract method not more than this.
* We are also not able to define more than one abstract method.
* Use @FunctionalInterface annotation in the interface definition.
* In this we have the ability to define any number of other methods like Default methods, Static methods.
* If we override java.lang.Object class’s method as an abstract method then it does not count as an abstract method.

**10. Describe some of the Functional interfaces in the standard library.**  
**Answer:** The functional interfaces in the java.util.function package are as follows:

* Function – It is the functional interface which takes one argument and then returns a result.
* Consumer – It is the functional interface that takes one argument and returns no result.
* Supplier – It is the functional interface that takes no argument and returns a result.
* Predicate – It is the functional interface which takes one argument and returns a Boolean.
* BiFunction – It is the functional interface which takes two arguments and returns a result.
* BinaryOperator – It is the functional interface which is same as a BiFunction. It takes two arguments and returns a result. The two arguments and the result are all of the same types.
* UnaryOperator – It is the functional interface which is same as a Function. It takes a single argument and returns a result of the same type.

**11. Differentiate between the Functional interface and SAM interface?**  
**Answer:** There is no difference between Functional interface and SAM interface. The SAM interface or Single Abstract Method interface is a kind of Functional interface defined in Java SE 8 API.

**12. Explain the differences and similarities between Function and Predicate in java 8** **programming** **language?**  
**Answer:**

**Differences:**  
Return Type: Function returns an Object and it is a single argument function. Predicate return type is boolean (i.e true or false) and it is also a single argument function.  
**Similarities:** Both are functional interfaces i.e both contain single abstract method.

**13. What are the guidelines that are needed to be followed in Functional Interface?**  
**Answer:** There are several guidelines stated below which are needed to be followed in Functional Interface.

1. The interface should be defined with only one abstract method.
2. Not more than one abstract can be defined.
3. Making use of @Functionalinterface annotation in the interface definition.
4. The override of the Java.lang.object class’s method will not be considered as an abstract method.
5. Any methods can be used for defining a number.

**14. What is a Method Reference?**  
**Answer:** Method reference is used for referring to a method of the functional interface. Method reference is defined as the compact and easy form of a lambda expression. Each time when you are using a lambda expression to just referring a method, you can replace your lambda expression with method reference.

**15. What are the different kinds of Method References?**  
**Answer:** There are four kinds of method references:

|  |  |
| --- | --- |
| **KIND** | **EXAMPLE** |
| Reference to static method | ContainingClass::staticMethodName |
| Reference to an instance method of a particular object | ContainingObject::instanceMethodName |
| Reference to an instance method of an arbitrary object of a particular type | ContainingType::methodName |
| Reference to a constructor | ClassName::new |

**16. Is it possible for us to provide implementation of a method in a Java Interface?**  
**Answer:** Yes, it is possible to provide implementation of a method in a Java Interface as it was not possible before Java8. It has introduced the flexibility of providing implementation of a method in an interface.  
There are two options for that:  
1.     Default Method: We can give default implementation of a method.  
2.     Static Method: We can create a static method in an interface and provide implementation

**17. What is a default method in Java 8 and when do we make use of it?**  
**Answer:** A default method is a method with an implementation that can be found in an interface. We make use of a default method for the addition of a new functionality to an interface while also maintaining backward compatibility with classes that are already implementing the interface.

Generally, when we add a new abstract method to an interface then all the implementing classes will break until they implement the new abstract method. In Java 8, this problem has found its solution by making the use of default method. For example, Collection interface does not have  forEach  method declaration. Thus, due to the addition of such method will simply break the whole collections API.

Java 8 introduces the default method so that the Collection interface can have a default implementation of  forEach method  without the requirement of the classes implementing this interface to implement the same.

**18. Why default methods in Interfaces are needed?**  
**Answer:** Default methods helps the user in enabling to add new functionality to the interfaces of the libraries and also ensures that the binary compatibility with code written for older versions of those interfaces. In a typical design based on the abstractions where an interface has one or more implementations and if one or more methods are being added to the interface then all the implementations will be forced to implement them too.

Otherwise, the design will just get break down so that the default interface methods are an efficient way to deal with this issue. This allow us to add new methods to an interface which are automatically available in the implementations. Therefore, there is no need in modifying the implementing classes. In this way the backward compatibility gets neatly preserved without having to refactor the implementers.

The default method is used for defining a method with a default implementation. We can override the default method and also to provide the more specific implementation for the method.

**19. Why static methods in interface are needed?**  
**Answer:**The static interface methods helps in providing a simple mechanism which allows us to increase the degree of cohesion of a design by putting together all related methods in one single place without having the need to create an object.

Static methods in interfaces help in making possible to group related utility methods without having the need for creating artificial utility classes which are simply placeholders for static methods.

**20. Differentiate between Iterator and Spliterator?**  
**Answer:** Difference between Iterator and Spliterator  are as follow:

* Introduction: Iterator is first introduced in jdk 1.2 while Spliterator is introduced in jdk 1.8.
* Use in API: Iterator is used for Collection API while Spliterator is used for Stream API.
* Parallel programming: Iterator can be used for iterating the elements in Collection in sequential order while Spliterator can be used for iterating the Stream elements in parallel or sequential order.
* Universal Iterator: Iteartor is universal iterator while Spliterator is not a universal iterator.

**21. How does Internal Iteration work in Java 8?**  
**Answer:** In an Iterator, the fundamental question that comes in our mind is which party controls the iteration work in Java 8. When a collection is able to control the iterator, then it is termed as External Iteration. When the Iterator is able to control the iteration then it is called as Internal Iteration.

In case of the Internal Iteration, the Iterator applies the operation to all the elements in aggregate and the client hands over an operation to Iterator. Internal Iteration is easier for the implementations as the Iterator doesn’t need to store the state of the collection.

**22. Differentiate between Internal and External Iterator?**  
**Answer:** Some of the differences between Internal and External Iterator are as follows:

1. An Internal Iterator controls the iteration itself. An External Iterator collection controls the iteration.
2. Internal Iterator can iterate elements in individually as well as in Bulk (like forEach). External iterator iterates element one by one.
3. Internal Iterator does not have to iterate elements only sequentially. External Iterator always iterates sequentially.
4. Internal Iterator supports declarative programming style that goes well with functional programming. External Iterator follows imperative style OOPS programming.
5. Some people consider Internal Iterator code more readable than that of External Iterator.

**23. Is Internal Iterator is advantageous over External Iterator in Java 8?**  
**Answer:**Yes, it is more advantageous then the external iterator. Some of the advantages of Internal Iterator are listed below:

1. Internal Iterator is based on Functional programming; therefore it can work on declarative style code.
2. There is no need to sequentially iterate elements in Internal Iterator.
3. Code is more readable and concise in Internal Iterator.
4. Internal Iterator supports concurrency and parallel processing.

**24. What are the applications of Internal Iteration?**  
**Answer:** We need Internal Iterator in applications that require high performance, parallel processing, fast iteration and bulk operations support. In the case of Internal Iteration applications, we are not having much control over the iteration. The other features like parallel processing etc. become more important.

**25. Explain the disadvantage of Internal Iteration over External Iteration?**  
**Answer:** Internal Iteration has many advantages over External Iteration. But it has one big disadvantage. Since Java API is responsible for iterating in Internal iterator, developer does not get any control over iteration.

**26. What is Nashorn in java 8?**  
**Answer:** Nashorn is defined as the latest javascript engine that is released with java 8. Before jdk 8, the javascript engine was based on Mozilla Rhino. It provides better compliance with ECMA normalized javascript specification and better runtime performance.

**27. What is a Stream in Java? How it is different from collection?**  
**Answer:** A stream is simply defined as an iterator whose role is to accept a set of actions to apply on each of the elements that it contains. The stream in Java represents a sequence of the objects from a source such as a collection which helps in supporting aggregate operations.

They were designed so that they can make collection processing simpler, easier and concise. Contrary to the collections, the logic of iteration is implemented inside the stream, so that we can make use of methods like map and flatMap for performing a declarative processing.

As the Stream API is fluent and also allows pipelining. Streams are distinct from the collections as the streams are inherently lazily loaded and processed.

**28. What is a Stream pipelining in Java 8?**  
**Answer:** Stream pipelining is the concept of chaining operations together. This can be done by splitting the operations that can happen on a stream into two categories: intermediate operations and terminal operations.

Each intermediate operation in the stream pipelining is the operation which returns an instance of Stream itself when it runs an arbitrary number of intermediate operations. Therefore, set up for processing of data which forms a processing pipeline. There is a need of the terminal operation which returns a final value and terminates the pipeline.

**29. Differentiate between Collection API and Stream API?**

|  |  |
| --- | --- |
| **Collection API** | **Stream API** |
| It is available since Java 1.2 | It is introduced in Java SE8. |
| It is used to store data that contains a set of objects. | It is used for computation of data that involves computation on a set of objects. |
| In this API we are able to use both spliterator and Iterator to iterate the elements. We can use forEach to perform an action for each element. | In this API we are not able to use spliterator and Iterator to iterate the elements. |
| It is used for storing limited number of elements. | It is used for storing either limited number or infinite number of elements. |
| Collection API uses External iteration concept to iterate elements such as the iterator. | Stream API uses external iteration concept for the iteration of the elements using forEach methods. |
| Collection Objects are constructed Eagerly. | Stream Objects are constructed lazily. |
| We add elements to Collection objects only after it is computed completely. | We add elements to Stream objects without any prior computation. That means stream objects are computed on- demand. |
| In this API we are having the ability to iterate and consume elements from a collection object at any number of times. | In this API we are having ability to iterate and consume elements from a stream object only once. |

**30. What is optional in Java?**  
**Answer:** Optional is a container object in Java that is used to contain not-null objects. Optional object is used for representing null with the absent value. This optional class has various utility methods for facilitating the code to handle values as ‘available’ or ‘not available’ instead of checking the null values. It is introduced in Java 8 and also similar to what Optional is in Guava.

The purpose of the Optional class is to provide a type-level solution for representation of optional values instead of using the null references.

**31. Describe the advantages of Java 8 optional?**  
**Answer:**

* In this the null checks are not required.
* No more NullPointerException at run-time.
* We can develop a clean and neat APIs.
* No more Boilerplate code is needed.

**32. What is Collectors class in Java 8?**  
**Answer:** Collectors are known as the final class that extends the Objectclass. It helps in providing the reduction of the operations like accumulating elements into the collections, summarizing elements according to various criteria, etc. Collectors perform the reduction operations. It helps in combining the results of the processing ongoing on the elements of the stream like collecting the elements into collections, summarizing the elements according to the different criteria. Collectors return the list or a string. As a result, the readability is increased. Static import used here is static import java.util.stream.Collectors.\*;

**33. What is StringJoiner?**  
**Answer:** StringJoiner belongs to the java.util package and it is used for combining the different strings into a single string with delimiters, and also with the prefixes and suffixes. It uses the two different constructors for this purpose. The constructors are called as the Public StringJoiner(CharSequence delimiter) and Public StringJoiner(CharSequence delimiter, CharSequence prefix, CharSequence suffix).

**34. What is the use of Java 8 Stringjoiner class?**  
**Answer:** Java programming language had added a new final class StringJoiner in the java.util package. It is used for the construction of a sequence of characters that are separated by a delimiter. Now, we can create a string by passing delimiters like a comma(,), hyphen(-) etc.

**35. How Java 8 supports Multiple Inheritance?**  
**Answer:** In Multiple Inheritance a class can inherit behavior from more than one parent classes. Prior to Java 8, a class can implement multiple interfaces but extend only one class. In Java 8, we are having the method implementation within an interface. So an interface behaves like an Abstract class. Now if we implement more than one interface with method implementation in a class, it means we are inheriting behavior from multiple abstract classes. That is how we get Multiple Inheritance in Java 8.

**36. What is Diamond Problem in inheritance? How Java 8 solves this problem?**  
**Answer:** A Diamond Problem is a Multiple Inheritance problem. In Java 8, it takes place when a Class extends more than one Interface which has same method implementation (Default method). To counter with this problem, Java 7 and the earlier versions does not support methods implementation in interface and also they don’t support Multiple Inheritance. Java 8 has introduced some new features like it has introduced the default methods to support the Multiple Inheritance with some limitations.

**37. What is jjs?**  
**Answer:** In Java 8 programming language, jjs is defined as the new executable or command line tool which is used for executing Javascript code at the console.

**38. What is the difference between Terminal and Intermediate options?**

|  |  |
| --- | --- |
| **Intermediate** | **Terminal** |
| The intermediate operation help in producing the stream pipelining. | Terminal operation terminates the pipeline. |
| The intermediate operations are the operations that can be chained multiple times on a stream. | Terminal operations cannot be chained various times. |
| Intermediate operations are the operations which cannot be evaluated independently and they need a terminal operation for their evaluation. | Terminal Operations can be evaluated independently. |

**39. What is spliterator in Java 8?**  
**Answer:** A Spliterator is one of the types of Java8 Iterator in java util package which traverses and partitions the elements of a source in Java into another Spliterator. A source of the spliterator may be a collection or an IO channel or a generator function.

**40. Differentiate between map and flatmap stream operation in Java 8 Programming Language?**

|  |  |
| --- | --- |
| **Map** | **Flat Map** |
| In the case of Map, for one input value we get one output value. | In case of flat Map, for one input value we gets the arbitrary number. |

**41. Explain the sequential and parallel stream in Java 8?**  
**Answer:** Parallel streams are the streams which help in dividing the provided venture into many and run them in unique threads and make use of a couple of cores of the computer. On the other hand, sequential streams work just like for-loop.

The duties which are supplied to the streams are usually the iterative operations performed on the factors of a series or array or from different dynamic sources. In case of Parallel execution of streams they run iterations which are more than one, concurrently in the extraordinary accessible cores.

**42. Explain local date-time API in Java 8?**  
**Answer:** In the case of new data-time API, one of the classes is the Local Date-Time API where there is no problem like the handling of the time-zones. Programmers or users make use of this API where there is no need of time zones. Local Date-Time API is also defined in the package java.time.

**43. What are features of Java 8 Date-Time API over the old date and Calender classes?**  
**Answer:** Here are some of the challenges faced by developers with earlier Java versions:

* It was having poor API design (for example, months start with 1 and days start with 0).
* It was not considered as thread safe.
* There is no consistency within the java.util and java.sql classes.
* There is no support for internationalization and time zone.

The new Date and Time API has been introduced in Java SE 8 with the following features:

* The improved API design has the ability of clearly separating the human- readable date time and machine time.
* The new Date and Time API helps in making all the classes immutable that is safe and best suited for the multithreaded environment.
* The new Date and Time API has more clarity in the API design level. The methods in this are clearly defined and perform the same action in all classes.
* The API has the ability to define a separate class for Date, Time, DateTime, Timestamp, TimeZone, and so on but the new Date and Time API works with the ISO-8601 calendar. However, we can also use the API with non-ISO calendars as well.

**44. What are the basic ideas behind the Date/Time API of Java 8 Programming** **Language?**  
**Answer:** There are some basic ideas behind the Date/Time API of Java 8:

1. Immutable-value classes: The new API avoids thread safety and concurrency issues by ensuring that all the core classes are immutable   and represent well-defined values.
2. Domain-driven design: The new API is modeled on precise domain with classes that represent different use cases for Date and Time.
3. The emphasis on domain-driven design offers benefits like clarity and understandability.
4. Separation of chronologies: The new API allows people to work with different calendar systems. It supports the needs of users in different areas of the world likes Japan or Thailand that don’t follow ISO-8601.

**45. Describe the advantages of new Date and Time API in Java 8 over old Date API?**  
**Answer:** The advantages of Java 8 Date Time API over old Date API are as follows:  
1. Concurrency: Existing Date Time classes (such as java.util.Date and SimpleDateFormatter) are not thread-safe. This does not work well in concurrent applications. In new Date Time API, developer does not have to deal with concurrency issues while writing datehandling code.

2. Better Design: Date/Time classes prior to Java 8 have poor API design. For example, years in java.util.Date start at 2000, months start at 1, and days start at 0. It is not an innovative idea. Java 8 Date Time API handles this problem very well and efficiently.

3. No need for 3rd Party Libraries: With the popularity of third-party Date/Time libraries like Joda Time, Java has to make its native Date/Time API comparable. Now we can use the Java API instead of using 3rd party libraries.

**46. Is it possible to get current time by using the Date/Time API of Java 8 Programming** **Language?**  
**Answer:** In Java 8 we have the option of using Clock class to get the current time. Instead of using old method System.currentTimeMillis(), we can create a Clock object and call millis() method to get the current time in milliseconds. We can also call instant() method on Clock object to get the current time in a readable format.

**47. What do you understand by the term Chromounits in Java 8?**  
**Answer:** The Chromounit was added in Java 8 to replace the value of an integer which was used in old API for representing the month, day, year, etc. unit is defined in the java.time.temporal.ChronoUnit.

**48. What is Type interference in Java 8?**  
Type interference is defined as the feature of Java which gives the capability to the compiler at each method invocation and that is corresponding announcement for determining the type of arguments. Java provides multiplied model of type inference in Java 8.

**49. What are the Parameters for Evaluating Java programmers?**  
**Answer:** Some of the parameters for evaluating Java programmers are as follows:

* Analyse the application of functional paradigm or OOPs.
* View the number of lines in code and determine if it could have been shorter.
* Examine the type of data structure used for problem-solving.
* We have to check that if the codes are easy to read, understand, and maintain.
* We should evaluate if the codes are having the quality of extensibility.
* Assess the quality of the Build Scripts.
* We must also check for the quality, readability and coverage of unit tests.

**50. What are various ways of creating Optional?**  
**Answer:**

* Optional.empty() – This method is used for creating optional which will return an empty Optional object.
* Optional.of() – This method will return an Optional of object passed as an argument to of method. It also returns an Optional with the specified present non-null value.
* Optional.ofNullable() – This method returns an Optional describing the specified value, if non-null, otherwise returns an empty Optional.

**51. What are the methods that are available in Optional?**  
1. **isPresent()** – This method will return true if the object is present into an Optional object.  
if(graphicsCard.isPresent()){  
System.out.println(graphicsCard.get());  
}  
**2. get()**– This method works if a value is present in this Optional, returns the value, otherwise throws NoSuchElementException. GraphicsCardnewCard=graphicsCard.get();  
**3. ifPresent** – This method works if a value is present, invoke the specified consumer with the value, otherwise do nothing.  
graphicsCard.ifPresent(System.out::println);  
**4. orElse(T other)** – This method return the value if present, otherwise returns the object passed as a parameter. Internals of orElse method in Java 8 Optional class.  
publicTorElse(Tother){  
returnvalue!=null?value:other;  
}

GraphicsCardnewCard=graphicsCard.orElse(newGraphicsCard());  
**5. orElseThrow** – This method returns the contained value, if present, otherwise throws an exception to be created by the provided supplier.  
GraphicsCardnewCard=graphicsCard.orElseThrow(IllegalArgumentException::new);

**52. Differentiate between Optional.of() and Optional.ofNullable() ?**   
Optional.of() versus Optional.ofNullable()

Optional.of() will return the object of Optional without null check.  
We should look at the interns of Optional.of() method:  
publicstatic&lt;T&gt;Optional&lt;T&gt;of(Tvalue){  
returnnewOptional&lt;&gt;(value);  
}

Optional.ofNullable() will return the object of Optional with a null check. If this method gets the null as an input then this method will return the empty Optional otherwise it returns an Optional with the specified present non-null value. We should look at the interns of Optional.ofNullable() method:  
publicstatic&lt;T&gt;Optional&lt;T&gt;ofNullable(Tvalue){  
returnvalue==null?empty():of(value);  
}  
Optional.of() should be used when we are sure that Optional will never have a null object and it will contain the object value or it will be empty but it will not be null. Optional.of() can also throw a NullPointerEception if the Optional is created with a null value.  
Optional.ofNullable()- is used when there are chances that the object might be null.

**53. What are repeating annotations?**  
**Answer:** Java 8 Programming Language has introduced a new Java language feature regarding the annotations. We are now able to repeat an annotation multiple times on a type declaration.

**54. What are Type Annotations? Name some common Type Annotations?**  
**Answer:** In addition to use the annotations on type declarations, we can now apply the annotations whenever we use types.  
Following are some examples of Type annotations  
@NonNull  
@ReadOnly  
@Regex,  
@Tainted  
@Untainted

**55. What is a Consumer?**  
**Answer:** Consumer is a functional interface added in Java 8 that has a single abstract method accept(). A consumer is used when you want to perform an operation that takes a parameter but doesn’t return anything.

**56. What is a Supplier?**  
**Answer:** Supplier is a functional interface with a single abstract method get(). A Supplier is used when we need to generate the instances without taking any input.

**57. Differentiate between Predicate, Supplier and Consumer in Java 8?**  
**Answer:** The differences between Predicate, Supplier and Consumer in Java 8 Programming Language are as follows:

* Predicate is an anonymous function that accepts one argument and returns a result.
* Supplier is an anonymous function that accepts no argument and returns a result.
* Consumer is an anonymous function that accepts one argument and returns no result.

**58. Differentiate between an interface with default method and an abstract class in Java** **8?**  
**Answer:** An interface having a default method seems to be same as an Abstract class in Java. But there are subtle differences between these two that are as follows:

1. Instance variable: An interface cannot have instance variables. An abstract class can have instance variables.
2. Constructor: An interface cannot have a constructor. An abstract class can have constructor.
3. Concrete Method: An interface cannot have concrete methods other than default method. An abstract class is having the ability for defining concrete methods with implementation.
4. Lambda: An interface with exactly one default method can be used for lambda expression. We cannot use abstract class for lambda expression.

**59. Does Java 7 support Type Inference?**  
**Answer:** Yes, Java 7 supports Type Inference. In Java 8, Oracle has enhanced the Type Inference concept. Now it can be used to define Lambda expressions, functions and Method references.

**60. Which method in Optional helps in providing with the fallback mechanism in case** **of null value?**  
**Answer:** In case, an Optional has null value, we can use orElseGet() method as fallback mechanism. If we implement orElseGet() method, it will be invoked when the value of Optional is null.

**61. How can we analyze the dependencies in Java classes and packages?**  
**Answer:** Java 8 usually comes with a new command line tool jdeps which helps in analyzing the package-level and class-level dependencies to the programmer. We can pass a jar file name or a class name as an argument to this tool. It will list all the dependencies of that jar or class.

**62. What are the JVM arguments that are introduced by Java 8?**  
**Answer:** In Java 8, PermGen space of ClassLoader has been removed. PermGen has been replaced with MetaSpace. Now we have the ability to set the initial and maximum size of the MetaSpace. The JVM options -XX:PermSize and –XX:MaxPermSize are replaced by –  
XX:MetaSpaceSize and -XX:MaxMetaspaceSize respectively in Java 8.

**63. Define the characteristics of a Lambda Expression?**  
**Answer:** Lambda expression has below mentioned characteristics:  
**1) Optional type declaration:**  
Optional type declaration states that when we are declaring the Lambda argument on the left side of the lambda then the type (of argument) is not required to declare. Compiler itself infer them from their values. For example:

1. int param -&gt; { body.. }
2. param -&gt; { body.. }

Both are valid.

**2) Optional parentheses :**  
Optional parentheses states that parentheses are not required in case only one single parameter is declared in Lambda. For Example :

1. param -&gt; { body.. }
2. (param) -&gt; { body.. }

Both are valid statements.  
**3) Optional curly braces :**  
It states that curly braces around lambda body is not required if the expressions part only has a single statement. For example :

* param – &gt; statement
* param – &gt; {statement;}

Both are valid expressions.  
**4) Optional return statement :**  
It states that if a lambda expression returns a value and the value is wrapped inside curly braces then the return statement is not required.

1. (x, y) – &gt; {return x+y;}
2. (x, y) – &gt; {x+y;}

Both are valid and equal.

**64. What is a SAM Interface?**  
**Answer:** SAM interface stands for Single Abstract Method interface. It is the other name for Functional interface. SAM has only one abstract method. Java 8 had released many functional interfaces.

**65. Are the abstract classes still useful?**  
**Answer:** Abstract classes can still do more in comparison to Java 8 interfaces:

1. Abstract class can have a constructor. The interface has no constructors to be invoked by the descendants.
2. Abstract classes are more structured and can hold a state. In comparison, Interface methods are all public, field members are all constant. You may want to restrict access privileges of methods and/or make them operate on non-constant state.
3. A child class can call upon the abstract class method(s) by super, while it can not do so on default interface methods.
4. Type clarity: You can only extend one class. This makes it clearer what your object is and how to use it.

**66. Do we have PermGen in Java 8? Are you aware of MetaSpace?**  
**Answer:** Until Java 7 the JVM makes use of an area termed as PermGen to store the classes. PermGen got removed in Java 8 and it is replaced by the MetaSpace.

Major advantage of MetaSpace over permgen:  
PermGen was fixed in term of maximum size and can not grow dynamically but Metaspace can grow dynamically and do not have any size constraint.

**67. What is the advantage of Metaspace over PermGen?**  
**Answer:** PerGen was fixed in size and could not grow dynamically, while Metaspace can grow dynamically and does have any type of size constraint.

**68. Difference between Stream’s findFirst() and findAny()?**  
**Answer:** findFirst in a stream will always return the first element from the stream whereas findAny is allowed to choose any element from the stream. findFirst has a deterministic behavior whereas findAny has a nondeterministic behavior.

**69. Explain the System.out::println expression.**  
**Answer:** System.out::println method is defined as the static method reference to the println method of object of the System class.

**70. What is the purpose of BiConsumer functional interface?**  
**Answer:** It helps in representing an operation that accepts two input arguments and returns no result.

**71. What is the purpose of BiFunction functional interface?**  
**Answer:** It represents a function that accepts two arguments and produces a result.

**72. What is the purpose of BinaryOperator functional interface?**  
**Answer:** It helps in representing an operation upon two operands of the same type which produces a result of the same type as the operands.

**73. What is the purpose of BiPredicate functional interface?**  
**Answer:** It represents a predicate (Boolean-valued function) of two arguments.

**74. What is the purpose of BooleanSupplier functional interface?**  
**Answer:** It represents a supplier of Boolean-valued results.

**75. What is the purpose of Consumer functional interface?**  
**Answer:** It helps in representing an operation which accepts a single input argument and returns no result.

**76. What is the purpose of DoubleBinaryOperator functional interface?**  
**Answer:** It helps in the representation of an operation upon two double-valued operands and that produces a double-valued result.

**77. What is the purpose of DoubleConsumer functional interface?**  
**Answer:** It helps in representing an operation that accepts a single double-valued argument and then returns no result.

**78. What is the purpose of DoubleFunction functional interface?**  
**Answer:** It represents a function that accepts a double-valued argument and produces a result.

**79. What is the purpose of DoublePredicate functional interface?**  
**Answer:** It represents a predicate (Boolean-valued function) of one double-valued argument.

**80. What is the purpose of DoubleSupplier functional interface?**  
**Answer:** It represents a supplier of double-valued results.

**81. What is the purpose of DoubleToIntFunction functional interface?**  
**Answer:** It represents a function that accepts a double-valued argument and produces an int-valued result.

**82. What is the purpose of DoubleToLongFunction functional interface?**  
**Answer:** It helps in representing a function which accepts a double-valued argument and produces a long-valued result.

**83. What is the purpose of DoubleUnaryOperator functional interface?**  
**Answer:** It represents an operation on a single double-valued operand that produces a double- valued result.

**84. What is the purpose of Function functional interface?**  
**Answer:** It represents a function that accepts one argument and produces a result.

**85. What is the purpose of IntBinaryOperator functional interface?**  
**Answer:** It helps in the representation of an operation upon two int-valued operands and produces an int-valued result.

**86. What is the purpose of IntConsumer functional interface?**  
**Answer:** It helps in the representation of an operation which accepts a single int-valued argument and returns no result.

**87. What is the purpose of IntFunction&lt;R&gt; functional interface?**  
**Answer:** It represents a function that accepts an int-valued argument and produces a result.

**88. What is the purpose of IntPredicate functional interface?**  
**Answer:** It represents a predicate (Boolean-valued function) of one int-valued argument.

**89. What is the purpose of IntSupplier functional interface?**  
**Answer:** It represents a supplier of int-valued results.

**90. What is the purpose of IntToDoubleFunction functional interface?**  
**Answer:** It helps in the representation of a function that accepts an int-valued argument and that produces a double-valued result.

**91. What is the purpose of IntToLongFunction functional interface?**  
It helps in the representation of a function that accepts an int-valued argument and that produces a long-valued result.

**92. What is the purpose of IntUnaryOperator functional interface?**  
**Answer:** It represents an operation on a single int-valued operand that produces an int-valued result.

**93. What is the purpose of LongBinaryOperator functional interface?**  
**Answer:** It represents an operation upon two long-valued operands and produces a long-valued result.

**94. What is the purpose of LongConsumer functional interface?**  
**Answer:** It helps in the representation of an operation that accepts a single long-valued argument and returns no result.

**95. What is the purpose of LongFunction&lt;R&gt; functional interface?**  
**Answer:** It represents a function that accepts a long-valued argument and produces a result.

**96. What is the purpose of LongPredicate functional interface?**  
**Answer:** It helps in the representation of a predicate (Boolean-valued function) of one long-valued argument.

**97. What is the purpose of LongSupplier functional interface?**  
**Answer:** It represents a supplier of long-valued results.

**98. What is the purpose of LongToDoubleFunction functional interface?**  
**Answer:** It helps in the representation of a function which accepts a long-valued argument and that produces a double-valued result.

**99. What is the purpose of LongToIntFunction functional interface?**  
**Answer:** It helps in the representation of a function which accepts a long-valued argument and that produces an int-valued result.

**100. What is the purpose of LongUnaryOperator functional interface?**  
**Answer:** It represents an operation on a single long-valued operand that produces a long-valued result.

**101. What is the purpose of ObjDoubleConsumer functional interface?**  
**Answer:** It represents an operation that accepts an object-valued and a double-valued argument, and returns no result.

**102. What is the purpose of ObjIntConsumer functional interface?**  
**Answer:** It represents an operation that accepts an object-valued and an int-valued argument, and returns no result.

**103. What is the purpose of ObjLongConsumer functional interface?**  
**Answer:** It represents an operation that accepts an object-valued and a long-valued argument, and returns no result.

**104. What is the purpose of Predicate functional interface?**  
**Answer:**It represents a predicate (Boolean-valued function) of one argument.

**105. What is the purpose of Supplier functional interface?**  
**Answer:** It represents a supplier of results.

**106. What is the purpose of ToDoubleBiFunction functional interface?**  
**Answer:** It helps in the representation of a function that accepts two arguments and that produces a double-valued result.

**107. What is the purpose of ToDoubleFunction functional interface?**  
**Answer:** It represents a function that produces a double-valued result.

**108. What is the purpose of ToIntBiFunction functional interface?**  
**Answer:** It represents a function that accepts two arguments and produces an int-valued result.

**109. What is the purpose of ToIntFunction functional interface?**  
**Answer:** It represents a function that produces an int-valued result.

**110. What is the purpose of ToLongBiFunction functional interface?**  
**Answer:** It helps in representation of a function that accepts two arguments and produces a long- valued result.

**111. What is the purpose of ToLongFunction functional interface?**  
**Answer:** It represents a function that produces a long-valued result.

**112. What is the purpose of UnaryOperator functional interface?**  
**Answer:** It represents an operation on a single operand that produces a result of the same type as its operand.

**113. Is it possible to list the numbers and remove the duplicate elements in the list using Java SE 8 features?**  
**Answer:** Yes, we can list the numbers and remove the duplicate elements in the list by applying stream and then collecting it to set using Collections.toSet() method.