**CORE JAVA**

Features Of Java:

---------------------

1. simple

2. object oriented

3. robust

4. platform independent

5. portable

6. architectural neutral

7. secure

8. multi-threaded

9. distributed

10. high performance

11. dynamic

1. simple:

------

- java follows the same syntaxes of c and c++ languages which makes learning java simple.

- pointers, goto statement and multiple inheritance are the concepts which are available

in c & c++ languages which can make programmer confuse easily, so sun micro system has

removed pointers, goto statement and multiple inheritance from java to make java as

simple.

2. object oriented :

---------------

what is object oriented?

- object oriented is a programming paradigm or a programming methodology which has set

of principles.

principles

----------

i. class and object

ii. data hiding

iii. abstraction

iv. encapsulatuion

v. inheritance (is-a relationship)

vi. composition (has-a relationship)

vii. polymorphism

- as java supports all the principles of object oriented programming paradigm

thats why java is called as object oriented.

3. robust :

i. java has better memory management.

ii. java has better exception handling mechanism.

memory management: allocating and deallocating the memory is called as memory management.

there are 2 types of memory management.

1. static memory allocation

- allocating the memory at the time of compilation is called as static memory allocation.

ex:

int a;

int a[5];

etc

2. dynamic memory allocation

- allocating the memory at runtime dynamically is called as dynamic memory allocation.

ex:

malloc(), realloc(), calloc(), free() -> dynamic memory allocation functions

- java has better memory management because in java programmer is responsible to allocate

the memory and programmer is not responsible for deallocating the memory which is taken

care by garbage collector automatically.

- in java, sun micro system has provided a rich set of api classes and interfaces to work

with exception handling which makes exception handling mechanism better.

4. platform independent

--------------------

- once we compile a .java file, java compiler converts into .class file with byte code

and .class file is not specific to any platform it is platform independent we can carry

on any operating system and .class file is specific to jvm means only jvm can understand and execute the byte code so we can carry .class file with byte code on any machine with any operating system and any processor this is called as

platform independent.

5. architectural neutral

---------------------

- architecture means micro processor

- when we compile a java program, java compiler converts source code into byte code which

is not specific to any micro processor so we can execute the byte code on any micro processor

irrespective of their vendor this is called architectural neutral.

6. portable

--------

- java supports developing different types of applications and which can run on different

machines or devices and networks thats why java is called as portable.

ex: standalone applications -> client machine

mobile applications -> mobile devices

enterprise web applications -> networks

7. multi-threaded

--------------

- java is multi-threaded because in java, sun micro system has given a rich set of api

classes and interfaces to develop multi-threaded applications easily.

8. distributed

-----------

- In java, sun micro system has provided several distributed technologies (corba, rmi, ejb etc)

to develop distributed applications thats why java is called as distributed.

9. secure

------

- java is a secure language.

i. java will not allow to create pointers

ii. byte code is executed inside the jvm where byte code verification will be done before executing the

byte code.s

10. high performance

----------------

- java is high performance because in java along with interpreter JIT Compiler is there to execute the byte code.

11. dynamic

-------

- java supports dynamic memory allocation using new operator.

- classes will be loaded dynamically at runtime by the classloaders.

* **Variables:**

## **What is the Variable?**

Variable is a name which is given to a memory cell. Which is used to store data value during program execution in memory.

**Why is specifying datatype before variable mandatory?**

JVM will assign space for the variable in the memory. If we do not specify datatype, JVM cannot decide how much space is required by that variable during program execution.

**What is variable initialization? how is it different than assignment?**

When we assign a value to a variable first time, it is known as variable initialization. Before initialization objects have null value and primitive types have default values such as 0 or false

**can we declare and initialize a variable together?**

Yes, we can declare and initialize a variable together. Here is the syntax for the same:

**int** number=10;

**boolean** flag=**true**;

String name="CodePumpkin";

**how many types of datatypes are there in java?**

Java has two types of datatypes.

1. Primitive Data Type   
           There are 8 primitive data types in java i.e. short, byte, int, long, float, double, boolean and char.
2. Non-primitive Data Type  
         – array, class, interface and enum are non-primitive data types. If you have noticed, String is not primary / primitive data type in java. String is a name of class. Similarly, you can also define your own custom class types like Student, User, etc.

**How can you create constants in java?**

So, what is constant? we can assign values to variable any number of times, but what if we want variable value to be fixed and we don't want developers to change its value, once variable is initialized. We can achieve this by declaring variable as final. We call them such fields as constants.

public class ConstantDemo {

public static void main(String[] args) {

// variable declaration

final int NUMBER=10;

// Compiler will throw error on below line as we can not change value of constants

NUMBER = NUMBER + 20;

}

}

**What are the rules for naming java variables?**

There are some rules related to Java variable names:

1. Java Variable names are case sensitive. Variable declared as codepumpkin is different than CodePumpkin or CODEPUMPKIN
2. Variable name must start with a letter, or $ or \_ character. No other special character or digits are allowed as first character of variable name.
3. After the first character, java variable name can have digits as well along with letter, $ and \_
4. Variable name cannot be equal to reserved keyword in java. For example, you cannot give your variable name as int, if or boolean as they are reserved keyword in java.

Here are some of the valid java variable names:

|  |  |
| --- | --- |
| 1  2  3  4 | Codepumpkin  \_CodePumpkin  $CodePumpkin  Code\_Pumpkin\_1 |

Below are some of the invalid java variable names:

@codepumpkin

1\_CodePumpkin

Code\_Pumpkin\_#

for

if

**what are some common naming conventions for java variables?**

Compiler will not enforce you to follow the naming conventions. But as its common practice by all java developers, when some other developer will read your code, it will be easier for them to understand.

1. variable names are written in lowercase i.e. number, name.
2. If there are multiple words in the variable name, then use camel case i.e. first character of each word should be capitalized except first word i.e. codePumpkin, cityName
3. constant name should be inupper case. If there are multiple words in constant name, use underscore between two words i.e. NUMBER, STATUS\_CODE

**Where does java store variable?**

Variables are stored in RAM memory. RAM stands for Random Access Memory. RAM is a volatile memory, means when the power of system is off all the data is erased into it. User application & application software occupies the space in RAM. Variables take area in memory which is known as reserved area.

**Where does Java store data?**

Whenever an object is created, it's always stored in the Heap space and stack memory contains the reference to it. Stack memory only contains local primitive variables and reference variables to objects in heap space.

**Java Heap Space**

Java Heap space is used by java runtime to allocate memory to Objects and JRE classes. Whenever we create an object, it’s always created in the Heap space. Garbage Collection runs on the heap memory to free the memory used by objects that don’t have any reference. Any object created in the heap space has global access and can be referenced from anywhere of the application.

**Java Stack Memory**

Java Stack memory is used for the execution of a thread. They contain method-specific values that are short-lived and references to other objects in the heap that is getting referred from the method. Stack memory is always referenced in LIFO (Last-In-First-Out) order. Whenever a method is invoked, a new block is created in the stack memory for the method to hold local primitive values and reference to other objects in the method. As soon as the method ends, the block becomes unused and becomes available for the next method. Stack memory size is very less compared to Heap memory.

**Heap and Stack Memory in Java Program**

public class Memory {

public static void main(String[] args) { // Line 1

int i=1; // Line 2

Object obj = new Object(); // Line 3

Memory mem = new Memory(); // Line 4

mem.foo(obj); // Line 5

} // Line 9

private void foo(Object param) { // Line 6

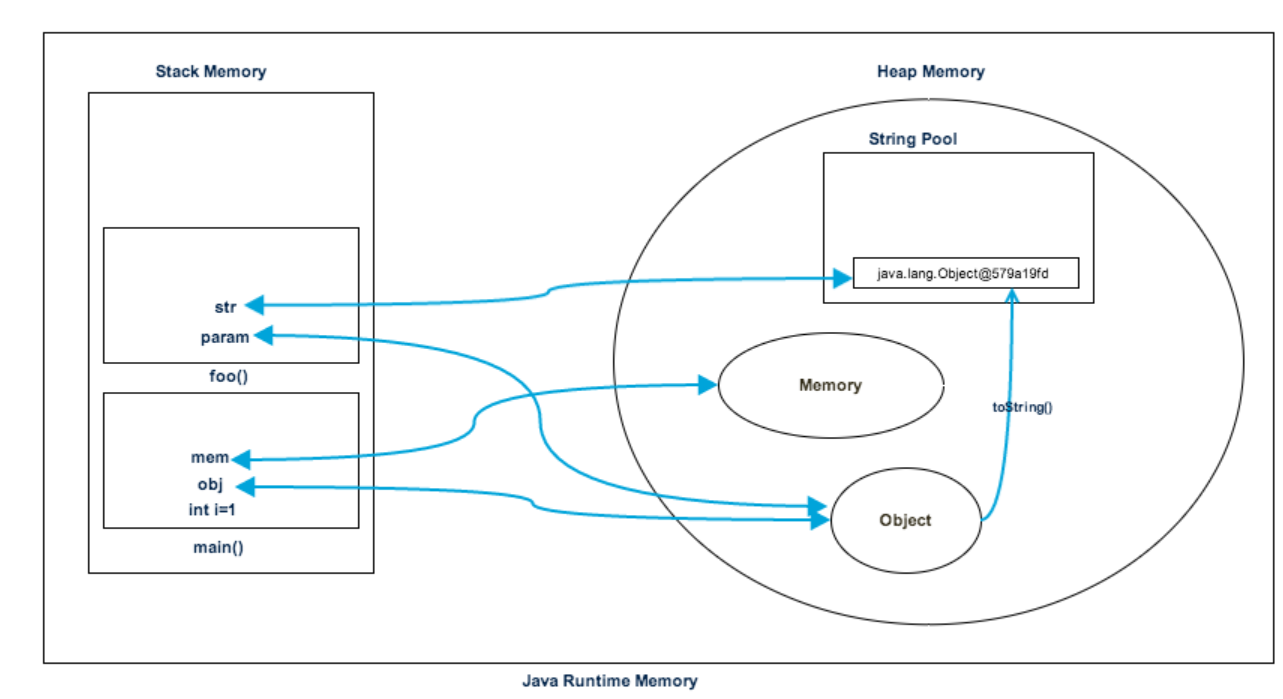
String str = param.toString(); //// Line 7

System.out.println(str);

} // Line 8

}

The below image shows the Stack and Heap memory with reference to the above program and how they are being used to store primitive, Objects, and reference variables.



Let’s go through the steps of the execution of the program.

As soon as we run the program, it loads all the Runtime classes into the Heap space. When the main() method is found at line 1, Java Runtime creates stack memory to be used by main() method thread.

We are creating primitive local variable at line 2, so it’s created and stored in the stack memory of main() method.

Since we are creating an Object in the 3rd line, it’s created in heap memory and stack memory contains the reference for it. A similar process occurs when we create Memory object in the 4th line.

Now when we call the foo() method in the 5th line, a block in the top of the stack is created to be used by the foo() method. Since Java is pass-by-value, a new reference to Object is created in the foo() stack block in the 6th line.

A string is created in the 7th line, it goes in the String Pool in the heap space and a reference is created in the foo() stack space for it.

foo() method is terminated in the 8th line, at this time memory block allocated for foo() in stack becomes free.

In line 9, main() method terminates, and the stack memory created for main() method is destroyed. Also, the program ends at this line, hence Java Runtime frees all the memory and ends the execution of the program.

**Difference between Java Heap Space and Stack Memory**

Heap memory is used by all the parts of the application whereas stack memory is used only by one thread of execution.

Whenever an object is created, it’s always stored in the Heap space and stack memory contains the reference to it. Stack memory only contains local primitive variables and reference variables to objects in heap space.

Objects stored in the heap are globally accessible whereas stack memory can’t be accessed by other threads.

Memory management in stack is done in LIFO manner whereas it’s more complex in Heap memory because it’s used globally. Heap memory is divided into Young-Generation, Old-Generation etc, more details at Java Garbage Collection.

Stack memory is short-lived whereas heap memory lives from the start till the end of application execution.

We can use -Xms and -Xmx JVM option to define the startup size and maximum size of heap memory. We can use -Xss to define the stack memory size.

When stack memory is full, Java runtime throws java.lang.StackOverFlowError whereas if heap memory is full, it throws java.lang.OutOfMemoryError: Java Heap Space error.

Stack memory size is very less when compared to Heap memory. Because of simplicity in memory allocation (LIFO), stack memory is very fast when compared to heap memory.

**🡪 Oops**

**What is Object Oriented Programming (OOPs)?**

object oriented is a programming paradigm or a programming methodology which has set of principles.

### What is a Class?

A **class** is a blueprint or template of objects. It is a user-defined data type that contains the data members and member methods.

**What is an Object?**

An **object** is an instance of a class. Data members and methods of a class cannot be used directly. We need to create an object (or instance) of the class to use them.

**What are the main features of OOPs?**

The main feature of the OOPs, also known as 4 pillars or basic principles of OOPs are as follows:

1. Encapsulation
2. Data Abstraction
3. Polymorphism
4. Inheritance

### ****6. What is Encapsulation?****

Encapsulation is the binding of data and methods that manipulate them into a single unit such that the sensitive data is hidden from the users.

**Advantages of Encapsulation:**

* Data Hiding
* Increase Flexibility
* Reusability

|  |
| --- |
| **public** **class** Student{  **private** String name;  **private** **int** rollNo;  **public** String getName(){  **return** name;      }  **public** **int** getRollNo(){  **return** rollNo;      }  **public** **void** setName(String name){  **this**.name=name      }  **public** **void** setRollNo(**int** rollNo){  **this**.rollNo=rollNo;      }  } |

**What is Abstraction?**

It means showing only the necessary information and hiding the other irrelevant information from the user. Abstraction is implemented using classes and interfaces.

|  |
| --- |
| **abstract** **class** Animal {  **abstract** **void** walk();  **void** eat()      {          System.out.println("The animal is eating.");      }      Animal()      {          System.out.println(              "An Animal is going to be created.");      }  }    **class** Cow **extends** Animal {      Cow() { System.out.println("You have created a Cow"); }  **void** walk() { System.out.println("Cow is walking."); }  }    **class** Goat **extends** Animal {      Goat()      {          System.out.println("You have created a Goat");      }  **void** walk() { System.out.println("Goat is walking."); }  }    **public** **class** OOPS {  **public** **static** **void** main(String args[])      {          Cow cow = **new** Cow();          cow.walk();          cow.eat();          Goat goat = **new** Goat();          goat.walk();          goat.eat();      }  } |

### ****8. What is Polymorphism?****

The word “**Polymorphism**” means having many forms.

Polymorphism can be classified into two types:

A. Compile Time Polymorphism  
B. Runtime Polymorphism

**A) Compile-Time Polymorphism**

Compile-time polymorphism is also known as static polymorphism or early binding.

Compile-time polymorphism is a polymorphism that is resolved during the compilation process. Compile-time polymorphism is achieved by **method overloading**and **operator overloading.**

### ****1. Method overloading****

Method Overloading occurs when a class has many methods with the same name but different parameters.

**B) Runtime Polymorphism**

It is Also known as dynamic polymorphism or late binding, runtime polymorphism is the type of polymorphism where the actual implementation of the function is determined during the runtime or execution.

Method overriding is an example of this method.

|  |
| --- |
| // An example of method overloading  **class** Student {     String name,surname;  **int** rollNo;    **public** **void** showStudentDetails(String name) {         System.out.println("The name of the student is " + name);     }      **public** **void** showStudentDetails(**int** rollNo) {         System.out.println("the roll no of the student is "+ rollNo);     }      **public** **void** showStudentDetails(String name, String surname, **int** rollNo) {         System.out.println(name);         System.out.println(surname);         System.out.println(age);     }  } |
| // an example of method oerriding  **class** Student {  **public** **void** read() {         System.out.println("The student is reading");     }  }  **class** SchoolStudent **extends** Student {  **public** **void** read(String book) {         System.out.println("the student is reding "+ book);     }  }  **class** CollegeStudent **extends** Student {  **public** **void** read(String researchPaper , String labJournal) {         System.out.println("the student is reading "+researchPaper +" and "+ labJournal);     }  } |

### ****9. What is Inheritance? What is its purpose?****

creating or deriving new classes from the existing classes so that new classes acquire all the properties of existing classes is called as inheritance.

The main purpose of Inheritance is to increase code reusability. It is also used to achieve Runtime Polymorphism.

|  |
| --- |
| // an example of inheritance  **class** Student {  **public** **void** read() {         System.out.println("The student is reading");     }  }  **class** SchoolStudent **extends** Student {  **public** **void** read(String book) {         System.out.println("the student is reding "+ book);     }  } |

### ****10. What are access specifiers? What is their significance in OOPs?****

Access specifiers are special types of keywords that are used to specify or control the accessibility of entities like classes, methods, and so on. **Private**, **Public**, and **Protected** are examples of access specifiers or access modifiers.

### 11. What are the advantages and disadvantages of OOPs?

| **Advantages of OOPs** | **Disadvantages of OOPs** |
| --- | --- |
| OOPs provides enhanced code reusability. | The programmer should be well-skilled and should have excellent thinking in terms of objects as everything is treated as an object in OOPs. |
| The code is easier to maintain and update. | Proper planning is required because OOPs is a little bit tricky. |
| It provides better data security by restricting data access and avoiding unnecessary exposure. | OOPs concept is not suitable for all kinds of problems. |
| Fast to implement and easy to redesign resulting in minimizing the complexity of an overall program. | The length of the programs is much larger in comparison to the procedural approach. |

**13. What is the difference between Structured Programming and Object-Oriented Programming?**

| **Object-Oriented Programming** | **Structural Programming** |
| --- | --- |
| Programming that is object-oriented is built on objects having a state and behavior. | A program’s logical structure is provided by structural programming, which divides programs into their corresponding functions. |
| It follows a bottom-to-top approach. | It follows a Top-to-Down approach. |
| Restricts the open flow of data to authorized parts only providing better data security. | No restriction to the flow of data. Anyone can access the data. |
| Enhanced code reusability due to the concepts of polymorphism and inheritance. | Code reusability is achieved by using functions and loops. |
| In this, methods are written globally, and code lines are processed one by one i.e., Run sequentially. | In this, the method works dynamically, making calls as per the need of code for a certain time. |
| Modifying and updating the code is easier. | Modifying the code is difficult as compared to OOPs. |
| Data is given more importance in OOPs. | Code is given more importance. |

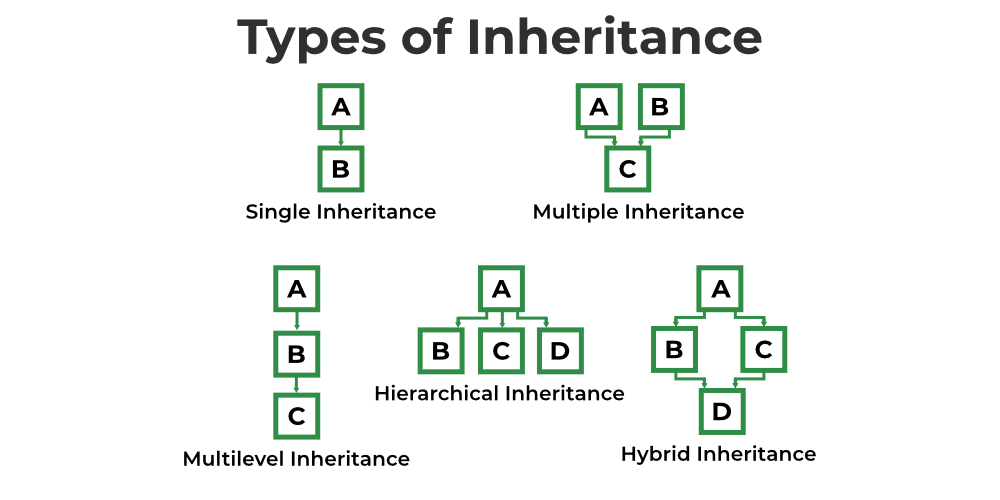
### ****Are there any limitations on Inheritance?****

**Yes,** there are more challenges when you have more authority. Although inheritance is a very strong OOPs feature, it also has significant drawbacks.

* As it must pass through several classes to be implemented, inheritance takes longer to process.
* The base class and the child class, which are both engaged in inheritance, are also closely related to one another (called tightly coupled). Therefore, if changes need to be made, they may need to be made in both classes at the same time.
* Implementing inheritance might be difficult as well. Therefore, if not implemented correctly, this could result in unforeseen mistakes or inaccurate outputs.

### ****18. What different types of inheritance are there?****

Inheritance can be classified into 5 types which are as follows:



1. **Single Inheritance:** Child class derived directly from the base class
2. **Multiple Inheritance:** Child class derived from multiple base classes.
3. **Multilevel Inheritance:**Child class derived from the class which is also derived from another base class.
4. **Hierarchical Inheritance:** Multiple child classes derived from a single base class.
5. **Hybrid Inheritance:** Inheritance consisting of multiple inheritance types of the above specified.

|  |
| --- |
| // an example of single inheritance  **class** Father {  }  **class** Son **extends** Father { } |
| // an example of hierarchial inheritance  **class** Father {  }  **class** Son **extends** Father {  }  **class** Daughter **extends** Father { } |
| // an example of multilevel inheritance  **class** Father {  }  **class** Son **extends** Father {  }  **class** GrandChild **extends** Son {    } |

### ****What is an interface?****

The interface is a mechanism to achieve abstraction. There can be only abstract methods in the interface, not the method body. It is used to achieve abstraction and multiple inheritance.

**How is an abstract class different from an interface?**

Both abstract classes and interfaces are special types of classes that just include the declaration of the methods, not their implementation. An abstract class is completely distinct from an interface, though. Following are some major differences between an abstract class and an interface.

| **Abstract Class** | **Interface** |
| --- | --- |
| A class that is abstract can have both abstract and non-abstract methods. | An interface can only have abstract methods. |
| An abstract class can have final, non-final, static, and non-static variables. | The interface has only static and final variables. |
| Abstract class doesn’t support multiple inheritance. | An interface supports multiple inheritance. |

### ****21. How much memory does a class occupy?****

Classes do not use memory. They serve as a template from which items are made. Now, objects initialize the class members and methods when they are created, using memory in the process.

### ****Is it always necessary to create objects from class?****

**No.**If the base class includes non-static methods, an object must be constructed. But no objects need to be generated if the class includes static methods. In this instance, you can use the class name to directly call those static methods.

### ****what is a constructor?****

### **- a constructor is like a special method which is called and executed automatically by the jvm at**

### **the time of creating an object.**

### **- constructor is used to initialize the instance variables only.**

### ****rules for writing a constructor:****

### **1. a constructor name must be same as class name and ending with parenthesis.**

### **2. a constructor cannot have a return type not even void.**

### **3. a constructor cannot have return statement also.**

### **4. a constructor can have access modifier like public, private, protected and default.**

### **5. a constructor must have a body.**

### **6. a constructor may or may not have parameters or arguments.**

### Can we overload the constructor in a class?

We can overload the constructor in a class. In fact, the default constructor, parameterized constructor, and copy constructor are the overloaded forms of the constructor.

### Can we overload the destructor in a class?

No. A destructor cannot be overloaded in a class. They can only be one destructor present in a class.

* **Exceptional Handling**

**what is an exception?**

--> an exception is an event (action) which is raised during the execution of a

program and which disrupts the normal flow of execution.

**what is exception handling?**

--> exception handling is a mechanism of providing alternate way or alternate path

for the exception so that the normal flow of execution is not interrupted.

**what is the difference between exception and error?**

--> exceptions are caused due to java programs and exceptions are recoverable.

--> errors are caused due to the lack of system resources and errors are not

recoverable.

Note:

--> All exceptions are runtime errors only.

--> All exceptions and errors are classes in java.

**Types of exceptions**

--------------------------

--> There are 2 types of exceptions

1. checked exceptions

2. unchecked exceptions

--> the exceptions which are checked or reported by the compiler at compilation time for smooth execution of a program at runtime and which are recoverable are called as checked exceptions.

--> the exceptions which are not checked or reported by the compiler at compilation time, which are raised directly at runtime, and which are not recoverable are called as unchecked exceptions.

--> Checked exceptions are 2 types

1. partially checked

2. fully checked

--> if parent is checked, some children are checked and some children are

unchecked such exceptions are called as partially checked.

ex: Exception and Throwable

--> if parent is checked and all children are also checked such exceptions

are called fully checked.

ex: IOException

--> default exception handler

--> There are 2 ways to handling checked exceptions

1. using try and catch

2. using throws keyword

--> If we don’t want to handle an exception because we don’t have any handling code or alternate path in that case simply, we can delegate (pass) the responsibility of handling a check exception to the caller of a method where checked exceptions are reporting by declaring exceptions to throws to that method.

--> There is only one way to handle unchecked exception

1. using try and catch

**methods to display exception information.**

---------------------------------------------------

1. void printStackTrace() - Throwable

-> it displays 3 informations

i. name of the exception

ii. description of the exception

iii. stack trace

2. String getMessage() - Throwable

--> it displays 1 information

i. description of the exception

3. String toString() - Throwable

--> it displays 2 informations

i. name of the exception

ii. description of the exception

**throws:**

-----------

--> it is a keyword, if we don’t want to handle a checked exception simply we want

to delegate (pass or give) the responsibility of handling a checked exception

to the caller of a method then use throws keyword to declare that exception to

that method.

--> throws keyword is used to declare only checked exceptions.

**throw:**

---------

--> it is a keyword if we want to create and throw an exception object explicitly

we use throw keyword.

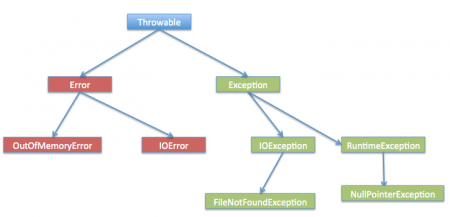
syntax:

throw exceptionObj;

### [What are the Exception Handling Keywords in Java?](https://www.digitalocean.com/community/tutorials/java-exception-interview-questions-and-answers#2-what-are-the-exception-handling-keywords-in-java)

1. **throw**: Sometimes we explicitly want to create an exception object and then throw it to halt the normal processing of the program. The **throw** keyword is used to throw exceptions to the runtime to handle it.
2. **throws**: When we are throwing any checked exception in a method and not handling it, then we need to use the throws keyword in the method signature to let the caller program know the exceptions that might be thrown by the method. The caller method might handle these exceptions or propagate them to its caller method using the throws keyword. We can provide multiple exceptions in the throws clause and it can be used with the **main()** method also.
3. **try-catch**: We use try-catch block for exception handling in our code. try is the start of the block and catch is at the end of the try block to handle the exceptions. We can have multiple catch blocks with a try and try-catch blocks can be nested also. catch block requires a parameter that should be of type Exception.
4. **finally**: The finally block is optional and can be used only with a try-catch block. Since exception halts the process of execution, we might have some resources open that will not get closed, so we can use the finally block. The finally block gets executed always, whether an exception occurs or not.

### [Explain Java Exception Hierarchy?](https://www.digitalocean.com/community/tutorials/java-exception-interview-questions-and-answers#3-explain-java-exception-hierarchy)

* Throwable is the parent class of Java Exceptions Hierarchy, and it has two child objects – Error and Exception.
* Exceptions are further divided into checked exceptions and runtime exceptions. **Errors** are exceptional scenarios that are out of the scope of application and it’s not possible to anticipate and recover from them, for example, hardware failure, JVM crash, or out-of-memory error.
* **Checked Exceptions** are exceptional scenarios that we can anticipate in a program and try to recover from it, for example, FileNotFoundException. We should catch this exception and provide a useful message to the user and log it properly for debugging purposes.
* Exception is the parent class of all Checked Exceptions. **Runtime Exceptions** are caused by bad programming, for example, trying to retrieve an element from the Array. We should check the length of the array first before trying to retrieve the element otherwise it might throw ArrayIndexOutOfBoundException at runtime. RuntimeException is the parent class of all runtime exceptions.[](https://journaldev.nyc3.cdn.digitaloceanspaces.com/2013/07/exception-hierarchy.png)

### [What are the important methods of Java Exception Class?](https://www.digitalocean.com/community/tutorials/java-exception-interview-questions-and-answers#4-what-are-the-important-methods-of-java-exception-class)

Exception and all its subclasses don’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 its constructor.
2. **String getLocalizedMessage()** - This method is provided so that subclasses can override it to provide the locale-specific messages 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 if 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.

### [Explain Java 7 ARM Feature and multi-catch block?](https://www.digitalocean.com/community/tutorials/java-exception-interview-questions-and-answers#5-explain-java-7-arm-feature-and-multi-catch-block)

If you are catching a lot of exceptions in a single try block, you will notice that catch block code looks very ugly and mostly consists of redundant code to log the error, keeping this in mind Java 7 one of the features was the multi-catch block where we can catch multiple exceptions in a single catch block. The catch block with this feature looks like below:

catch(IOException | SQLException | Exception ex){

logger.error(ex);

throw new MyException(ex.getMessage());

}

### [What is the difference between Checked and Unchecked Exceptions in Java?](https://www.digitalocean.com/community/tutorials/java-exception-interview-questions-and-answers#6-what-is-the-difference-between-checked-and-unchecked-exceptions-in-java)

1. Checked Exceptions should be handled in the code using try-catch block or else the method should use the throws keyword to let the caller know about the checked exceptions that might be thrown from the method. Unchecked Exceptions are not required to be handled in the program or to mention them in the throws clause of the method.
2. Exception is the superclass of all checked exceptions whereas RuntimeException is the superclass of all unchecked exceptions. Note that RuntimeException is the child class of Exception.
3. Checked exceptions are error scenarios that require to be handled in the code, or else you will get compile time error. For example, if you use FileReader to read a file, it throws FileNotFoundException and we must catch it in the try-catch block or throw it again to the caller method. Unchecked exceptions are mostly caused by poor programming, for example, NullPointerException when invoking a method on an object reference without making sure that it’s not null. For example, I can write a method to remove all the vowels from the string. It’s the caller’s responsibility to make sure not to pass a null string. I might change the method to handle these scenarios but ideally, the caller should take care of this.

### [What is the difference between the throw and throws keyword in Java?](https://www.digitalocean.com/community/tutorials/java-exception-interview-questions-and-answers#7-what-is-the-difference-between-the-throw-and-throws-keyword-in-java)

throws keyword is used with method signature to declare the exceptions that the method might throw whereas throw keyword is used to disrupt the flow of the program and handing over the exception object to runtime to handle it.

### [How to write custom exceptions in Java?](https://www.digitalocean.com/community/tutorials/java-exception-interview-questions-and-answers#8-how-to-write-custom-exceptions-in-java)

We can extend Exception class or any of its subclasses to create our custom exception class. The custom exception class can have its own variables and methods that we can use to pass error codes or other exception-related information to the exception handler. A simple example of a custom exception is shown below.

package com.journaldev.exceptions;

import java.io.IOException;

public class MyException extends IOException {

private static final long serialVersionUID = 4664456874499611218L;

private String errorCode="Unknown\_Exception";

public MyException(String message, String errorCode){

super(message);

this.errorCode=errorCode;

}

public String getErrorCode(){

return this.errorCode;

}

}

### What is OutOfMemoryError in Java?

OutOfMemoryError in Java is a subclass of java.lang.VirtualMachineError and it’s thrown by JVM when it ran out of heap memory. We can fix this error by providing more memory to run the java application through java options. $>java MyProgram -Xms1024m -Xmx1024m -XX:PermSize=64M -XX:MaxPermSize=256m

### [What are different scenarios causing “Exception in thread main”?](https://www.digitalocean.com/community/tutorials/java-exception-interview-questions-and-answers#10-what-are-different-scenarios-causing-exception-in-thread-main)

Some of the common main thread exception scenarios are:

* **Exception in thread main java.lang.UnsupportedClassVersionError**: This exception comes when your java class is compiled from another JDK version and you are trying to run it from another java version.
* **Exception in thread main java.lang.NoClassDefFoundError**: There are two variants of this exception. The first one is where you provide the class full name with .class extension. The second scenario is when Class is not found.
* **Exception in thread main java.lang.NoSuchMethodError: main**: This exception comes when you are trying to run a class that doesn’t have the main method.
* **Exception in thread “main” java.lang.ArithmeticException**: Whenever an exception is thrown from the main method, it prints the exception in the console. The first part explains that an exception is thrown from the main method, the second part prints the exception class name and then after a colon, it prints the exception message.

### [What is the difference between final, finally, and finalize in Java?](https://www.digitalocean.com/community/tutorials/java-exception-interview-questions-and-answers#11-what-is-the-difference-between-final-finally-and-finalize-in-java)

final and finally are keywords in java whereas finalize is a method. final keyword can be used with class variables so that they can’t be reassigned, with the class to avoid extending by classes and with methods to avoid overriding by subclasses, finally keyword is used with try-catch block to provide statements that will always get executed even if some exception arises, usually finally is used to close resources. finalize() method is executed by Garbage Collector before the object is destroyed, it’s a great way to make sure all the global resources are closed. Out of the three, only finally is related to java exception handling.

### [What happens when an exception is thrown by the main method?](https://www.digitalocean.com/community/tutorials/java-exception-interview-questions-and-answers#12-what-happens-when-an-exception-is-thrown-by-the-main-method)

When an exception is thrown by a main() method, Java Runtime terminates the program and prints the exception message and stack trace in the system console.

### [Can we have an empty catch block?](https://www.digitalocean.com/community/tutorials/java-exception-interview-questions-and-answers#13-can-we-have-an-empty-catch-block)

We can have an empty catch block but it’s an example of bad programming. We should never have an empty catch block because if the exception is caught by that block, we will have no information about the exception and it wil be a nightmare to debug it. There should be at least a logging statement to log the exception details in console or log files.

### [What is the problem with the below programs and how do we fix it?](https://www.digitalocean.com/community/tutorials/java-exception-interview-questions-and-answers#15-what-is-the-problem-with-the-below-programs-and-how-do-we-fix-it)

In this section, we will investigate some programming questions related to java exceptions.

1. **What is the problem with the below program?**
2. package com.journaldev.exceptions;
3. import java.io.FileNotFoundException;
4. import java.io.IOException;
5. public class TestException {
6. public static void main(String[] args) {
7. try {
8. testExceptions();
9. } catch (FileNotFoundException | IOException e) {
10. e.printStackTrace();
11. }
12. }


16. public static void testExceptions() throws IOException, FileNotFoundException{
18. }
19. }

The above program won’t compile and you will get an error message as “The exception FileNotFoundException is already caught by the alternative IOException”. This is because FileNotFoundException is a subclass of IOException, there are two ways to solve this problem. The first way is to use a single catch block for both the exceptions.

try {

testExceptions();

}catch(FileNotFoundException e){

e.printStackTrace();

}catch (IOException e) {

e.printStackTrace();

}

Another way is to remove the FileNotFoundException from the multi-catch block.

try {

testExceptions();

}catch (IOException e) {

e.printStackTrace();

}

You can chose any of these approach based on your catch block code.

1. **What is the problem with the below program?**
2. package com.journaldev.exceptions;
3. import java.io.FileNotFoundException;
4. import java.io.IOException;
5. import javax.xml.bind.JAXBException;
6. public class TestException1 {
7. public static void main(String[] args) {
8. try {
9. go();
10. } catch (IOException e) {
11. e.printStackTrace();
12. } catch (FileNotFoundException e) {
13. e.printStackTrace();
14. } catch (JAXBException e) {
15. e.printStackTrace();
16. }
17. }
18. public static void go() throws IOException, JAXBException, FileNotFoundException{
20. }
21. }

The program won’t compile because FileNotFoundException is a subclass of IOException, so the catch block of FileNotFoundException is unreachable and you will get an error message as “Unreachable catch block for FileNotFoundException. It is already handled by the catch block for IOException”. You need to fix the catch block order to solve this issue.

try {

go();

} catch (FileNotFoundException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

} catch (JAXBException e) {

e.printStackTrace();

}

Notice that JAXBException is not related to IOException or FileNotFoundException and can be put anywhere in the above catch block hierarchy.

1. **What is the problem with the below program?**
2. package com.journaldev.exceptions;
3. import java.io.IOException;
4. import javax.xml.bind.JAXBException;
5. public class TestException2 {
6. public static void main(String[] args) {
7. try {
8. foo();
9. } catch (IOException e) {
10. e.printStackTrace();
11. }catch(JAXBException e){
12. e.printStackTrace();
13. }catch(NullPointerException e){
14. e.printStackTrace();
15. }catch(Exception e){
16. e.printStackTrace();
17. }
18. }
19. public static void foo() throws IOException{
21. }
22. }

The program won’t compile because JAXBException is a checked exception and foo() method should throw this exception to catch in the calling method. You will get an error message as “Unreachable catch block for JAXBException. This exception is never thrown from the try statement body”. To solve this issue, you will have to remove the catch block of JAXBException. Notice that catching NullPointerException is valid because it’s an unchecked exception.

1. **What is the problem with the below program?**
2. package com.journaldev.exceptions;
3. public class TestException3 {
4. public static void main(String[] args) {
5. try{
6. bar();
7. }catch(NullPointerException e){
8. e.printStackTrace();
9. }catch(Exception e){
10. e.printStackTrace();
11. }
13. foo();
14. }
15. public static void bar(){
17. }
19. public static void foo() throws NullPointerException{
21. }
22. }

This is a trick question, there is no problem with the code and it will compile successfully. We can always catch an Exception or any unchecked exception even if it’s not in the throws clause of the method. Similarly, if a method (foo) declares an unchecked exception in the throws clause, it is not mandatory to handle that in the program.

1. **What is the problem with the below program?**
2. package com.journaldev.exceptions;
3. import java.io.IOException;
4. public class TestException4 {
5. public void start() throws IOException{
6. }
8. public void foo() throws NullPointerException{
10. }
11. }
12. class TestException5 extends TestException4{
14. public void start() throws Exception{
15. }
17. public void foo() throws RuntimeException{
19. }
20. }

The above program won’t compile because the start() method signature is not the same in the subclass. To fix this issue, we can either change the method singnature in the subclass to be exactly the same as the superclass or we can remove the throws clause from the subclass method as shown below.

@Override

public void start(){

}

1. **What is the problem with the below program?**
2. package com.journaldev.exceptions;
3. import java.io.IOException;
4. import javax.xml.bind.JAXBException;
5. public class TestException6 {
6. public static void main(String[] args) {
7. try {
8. foo();
9. } catch (IOException | JAXBException e) {
10. e = new Exception("");
11. e.printStackTrace();
12. }catch(Exception e){
13. e = new Exception("");
14. e.printStackTrace();
15. }
16. }
17. public static void foo() throws IOException, JAXBException{
19. }
20. }

The above program won’t compile because the exception object in the multi-catch block is final and we can’t change its value. You will get compile time error as “The parameter e of a multi-catch block cannot be assigned”. We have to remove the assignment of “e” to a new exception object to solve this error.

* **Packages**

## **1. What is package?**

Package is a collection of group of similar types of classes, interfaces and sub packages.

## **2. State the use of package?**

1. Package use to search .class file very faster.
2. It is use to categorize classes & interfaces so that they can be easily maintained.
3. Package use to remove naming collision.

## **3. What are the diff. types of packages with example& define them?**

Package categorized into 2 types:

1. Built-in package
2. User-defined package.

Built-in package-If the package already design & stored in the predefined class then it is known as Built –in package User-defined package-User-defined classes are the collection of user defined classes.

## **4. How to create a user defined package?**

Package is a keyword in java which is used to designed user defined package.

## **5. What is the name of default package & its benefits?**

Lang is the default package in java.

## **6. How to set the class path & its advantage?**

Class path able to set by 2 different ways:

1. Permanently within environment variables
2. Temporarily by set class path command.

## **7. How to compile a package program?**

Javac -d . filename.java

## **8. How to create a .jar file?**

jar -cvf Filename.jar Filename.class

## **9. How to create a jar executable file?**

jar -cmf manifest\_fileFilename.jar Filename.class

## **10. How to use the class file, i.e. ways to access the class file. And diff b/w import and fully qualified name?**

Two different ways user can use the class file.

1. by import keyword
2. by fully qualified name.

In case of package importing no need to specify the full path of package. But in case of fully qualified name user has to specify the package name each time to access the class and it is not flexible.

## **11. Specify a situation where fully qualified is necessary?**

Fully qualified name is requiredwhen same class present in more than one package.

## **12. Is there any need to import java.lang package?**

No not required

## **13. How to access package from another package?**

There are 3 ways to access the package from another package

1. import package.\*;
2. import package . class name.
3. fully qualified name.

## **14. What is sub package?**

Package inside the package is called the sub package.It is use to categorise the package further.

## **15. Which package is always imported by default?**

Lang package.

## **16. Does Importing a package imports its sub-packages as well in Java?**

No importing package means only current package not its sub-package.

## **17. What are the advantages of package?**

1. package provides access protection.
2. package removes naming collision.

## **18. State the use of import keyword?**

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

## **19. How much pre-defined package java supports?**

80,000 classes present in 202 no of packages.

## **20. Who is the default package in java?**

Lang is the default package in java.

## **21. State the use of package keywords**

The package keyword is used to create a package.

## **22. When to use fully qualified name?**

We need to use fully qualified name every time when accessing the classes and interfaces.

## **23. State the operation & package of Socket& Server Socket classes?**

Socket & Server socket class present in net package and its operation mainly for networking purpose.

## **24. Input/Output classes present in which package?**

Input/Output classes present in io package.

## **25. What are the ways to load class file?**

There are 2 ways to load class file

**Temporary**:

1. by setting the class path in the command prompt.
2. by class path switch.

**Permanent**:

1. by setting the class path in the environment variable.
2. by creating the jar file.

## **26. How to create document API in java?**

Document API can create in java by the help of javadoc tool.

## **27. Which package is used for achieving the language Functionalities?**

java.lang.\* is used for language functionalities.

## **28. Which package is used for developing file handling application?**

java.io.\* is used for file handling application.

## **29. Which package is used for developing GUI components?**

java.awt.\* is used for GUI components.

## **30. Which package is used for developing browser oriented application?**

java.applet.\* is used for browser oriented application.

## **31. Which package is used for developing client server application?**

java.net.\* is used for client server application.

## **32. Which package is used for developing quality or reliable application in java?**

java.util.\* is used for quality or reliable application in java.

## **33. Which package is used for formatting data and time on day to day business operation?**

java.text.\* is used for formatting data and time on day to day business operation.

## **34. State the use of Event package?**

This package is used for providing the functionality of GUI components.

## **35. Which package is also known as collection framework in java?**

java.util.\* is also known as collection framework in java.

## **36. Which package is used for runtime information about the class and interface?**

java.lang.reflect.\* is used for study run time information about the class and interface.

## **37. Which package is used for retrieving data from database & performing various operation on database?**

java.sql.\* is used for retrieving data from db& performing various operation on db.

## **38. Which is known as universal access specifier?**

Public access specifier.

## **39. Which is known as package access specifier?**

Default access specifier.

## **40. Which is known as inherited access specifier?**

Protected access specifier.

## **41. Which is known as native access specifier?**

Private access specifier.

## **42. What access specifier represent?**

Access specifier represent the visibility of data or accessibility of data.

* **Multi-Threading**

### 1) What is multithreading?

Multithreading is a process of executing multiple threads simultaneously. Multithreading is used to obtain the multitasking. It consumes less memory and gives the fast and efficient performance. Its main advantages are:

* Threads share the same address space.
* The thread is lightweight.
* The cost of communication between the processes is low.

### 2) What is the thread?

A thread is a lightweight subprocess. It is a separate path of execution because each thread runs in a different stack frame. A process may contain multiple threads. Threads share the process resources, but still, they execute independently.

[More details.](https://www.javatpoint.com/multithreading)

### 3) Differentiate between process and thread?

There are the following differences between the process and thread.

* A Program in the execution is called the process whereas; A thread is a subset of the process
* Processes are independent whereas threads are the subset of process.
* Process have different address space in memory, while threads contain a shared address space.
* Context switching is faster between the threads as compared to processes.
* Inter-process communication is slower and expensive than inter-thread communication.
* Any change in Parent process doesn't affect the child process whereas changes in parent thread can affect the child thread.



### 4) What do you understand by inter-thread communication?

* The process of communication between synchronized threads is termed as inter-thread communication.
* Inter-thread communication is used to avoid thread polling in Java.
* The thread is paused running in its critical section, and another thread is allowed to enter (or lock) in the same critical section to be executed.
* It can be obtained by wait(), notify(), and notifyAll() methods.

### 5) What is the purpose of wait() method in Java?

The wait() method is provided by the Object class in Java. This method is used for inter-thread communication in Java. The java.lang.Object.wait() is used to pause the current thread, and wait until another thread does not call the notify() or notifyAll() method. Its syntax is given below.

public final void wait()

### 6) Why must wait() method be called from the synchronized block?

We must call the wait method otherwise it will throw **java.lang.IllegalMonitorStateException** exception. Moreover, we need wait() method for inter-thread communication with notify() and notifyAll(). Therefore It must be present in the synchronized block for the proper and correct communication.

### 7) What are the advantages of multithreading?

Multithreading programming has the following advantages:

* Multithreading allows an application/program to be always reactive for input, even already running with some background tasks
* Multithreading allows the faster execution of tasks, as threads execute independently.
* Multithreading provides better utilization of cache memory as threads share the common memory resources.
* Multithreading reduces the number of the required server as one server can execute multiple threads at a time.

### 8) What are the states in the lifecycle of a Thread?

A thread can have one of the following states during its lifetime:

1. **New:** In this state, a Thread class object is created using a new operator, but the thread is not alive. Thread doesn't start until we call the start() method.
2. **Runnable:** In this state, the thread is ready to run after calling the start() method. However, the thread is not yet selected by the thread scheduler.
3. **Running:** In this state, the thread scheduler picks the thread from the ready state, and the thread is running.
4. **Waiting/Blocked:** In this state, a thread is not running but still alive, or it is waiting for the other thread to finish.
5. **Dead/Terminated:** A thread is in terminated or dead state when the run() method exits.



### 9) What is the difference between preemptive scheduling and time slicing?

Under preemptive scheduling, the highest priority task executes until it enters the waiting or dead states or a higher priority task comes into existence. Under time slicing, a task executes for a predefined slice of time and then reenters the pool of ready tasks. The scheduler then determines which task should execute next, based on priority and other factors.

### 10) What is context switching?

In Context switching the state of the process (or thread) is stored so that it can be restored and execution can be resumed from the same point later. Context switching enables the multiple processes to share the same CPU.

### 11) Differentiate between the Thread class and Runnable interface for creating a Thread?

The Thread can be created by using two ways.

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

However, the primary differences between both the ways are given below:

* By extending the Thread class, we cannot extend any other class, as Java does not allow multiple inheritances while implementing the Runnable interface; we can also extend other base class(if required).
* By extending the Thread class, each of thread creates the unique object and associates with it while implementing the Runnable interface; multiple threads share the same object
* Thread class provides various inbuilt methods such as getPriority(), isAlive and many more while the Runnable interface provides a single method, i.e., run().

### 12) What does join() method?

The join() method waits for a thread to die. In other words, it causes the currently running threads to stop executing until the thread it joins with completes its task. Join method is overloaded in Thread class in the following ways.

* public void join()throws InterruptedException
* public void join(long milliseconds)throws InterruptedException

[More details.](https://www.javatpoint.com/join()-method)

### 13) Describe the purpose and working of sleep() method.

The sleep() method in java is used to block a thread for a particular time, which means it pause the execution of a thread for a specific time. There are two methods of doing so.

**Syntax:**

* public static void sleep(long milliseconds)throws InterruptedException
* public static void sleep(long milliseconds, int nanos)throws InterruptedException

**Working of sleep() method**

When we call the sleep() method, it pauses the execution of the current thread for the given time and gives priority to another thread(if available). Moreover, when the waiting time completed then again previous thread changes its state from waiting to runnable and comes in running state, and the whole process works so on till the execution doesn't complete.

### 14) What is the difference between wait() and sleep() method?

|  |  |
| --- | --- |
| **wait()** | **sleep()** |
| 1) The wait() method is defined in Object class. | The sleep() method is defined in Thread class. |
| 2) The wait() method releases the lock. | The sleep() method doesn't release the lock. |

### 15) Is it possible to start a thread twice?

No, we cannot restart the thread, as once a thread started and executed, it goes to the Dead state. Therefore, if we try to start a thread twice, it will give a runtimeException "java.lang.IllegalThreadStateException". Consider the following example.

1. **public** **class** Multithread1 **extends** Thread
2. {
3. **public** **void** run()
4. {
5. **try** {
6. System.out.println("thread is executing now........");
7. } **catch**(Exception e) {
8. }
9. }
10. **public** **static** **void** main (String[] args) {
11. Multithread1 m1= **new** Multithread1();
12. m1.start();
13. m1.start();
14. }
15. }

**Output**

thread is executing now........

Exception in thread "main" java.lang.IllegalThreadStateException

at java.lang.Thread.start(Thread.java:708)

at Multithread1.main(Multithread1.java:13)

[More details.](https://www.javatpoint.com/can-we-start-a-thread-twice)

### 16) Can we call the run() method instead of start()?

Yes, calling run() method directly is valid, but it will not work as a thread instead it will work as a normal object. There will not be context-switching between the threads. When we call the start() method, it internally calls the run() method, which creates a new stack for a thread while directly calling the run() will not create a new stack.

[More details.](https://www.javatpoint.com/what-if-we-call-run()-method-directly)

### 17) What about the daemon threads?

The daemon threads are the low priority threads that provide the background support and services to the user threads. Daemon thread gets automatically terminated by the JVM if the program remains with the daemon thread only, and all other user threads are ended/died. There are two methods for daemon thread available in the Thread class:

* **public void setDaemon(boolean status):** It used to mark the thread daemon thread or a user thread.
* **public boolean isDaemon():** It checks the thread is daemon or not.

[More details.](https://www.javatpoint.com/daemon-thread)

### 18)Can we make the user thread as daemon thread if the thread is started?

ADVERTISEMENT

ADVERTISEMENT

No, if you do so, it will throw IllegalThreadStateException. Therefore, we can only create a daemon thread before starting the thread.

1. **class** Testdaemon1 **extends** Thread{
2. **public** **void** run(){
3. System.out.println("Running thread is daemon...");
4. }
5. **public** **static** **void** main (String[] args) {
6. Testdaemon1 td= **new** Testdaemon1();
7. td.start();
8. setDaemon(**true**);// It will throw the exception: td.
9. }
10. }

**Output**

Running thread is daemon...

Exception in thread "main" java.lang.IllegalThreadStateException

at java.lang.Thread.setDaemon(Thread.java:1359)

at Testdaemon1.main(Testdaemon1.java:8)

[More details.](https://www.javatpoint.com/daemon-thread)

### 19)What is shutdown hook?

The shutdown hook is a thread that is invoked implicitly before JVM shuts down. So we can use it to perform clean up the resource or save the state when JVM shuts down normally or abruptly. We can add shutdown hook by using the following method:

1. **public** **void** addShutdownHook(Thread hook){}
2. Runtime r=Runtime.getRuntime();
3. r.addShutdownHook(**new** MyThread());

Some important points about shutdown hooks are :

* Shutdown hooks initialized but can only be started when JVM shutdown occurred.
* Shutdown hooks are more reliable than the finalizer() because there are very fewer chances that shutdown hooks not run.
* The shutdown hook can be stopped by calling the halt(int) method of Runtime class.

[More details.](https://www.javatpoint.com/ShutdownHook-thread)

### 20)When should we interrupt a thread?

We should interrupt a thread when we want to break out the sleep or wait state of a thread. We can interrupt a thread by calling the interrupt() throwing the InterruptedException.

[More details.](https://www.javatpoint.com/interrupting-a-thread)

### 21) What is the synchronization?

Synchronization is the capability to control the access of multiple threads to any shared resource. It is used:

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

When the multiple threads try to do the same task, there is a possibility of an erroneous result, hence to remove this issue, Java uses the process of synchronization which allows only one thread to be executed at a time. Synchronization can be achieved in three ways:

* by the synchronized method
* by synchronized block
* by static synchronization

Syntax for synchronized block

1. **synchronized**(object reference expression)
2. {
3. //code block
4. }

[More details.](https://www.javatpoint.com/synchronization)

### 22) What is the purpose of the Synchronized block?

The Synchronized block can be used to perform synchronization on any specific resource of the method. Only one thread at a time can execute on a particular resource, and all other threads which attempt to enter the synchronized block are blocked.

* Synchronized block is used to lock an object for any shared resource.
* The scope of the synchronized block is limited to the block on which, it is applied. Its scope is smaller than a method.

[More details.](https://www.javatpoint.com/synchronized-block-example)

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

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

### 24) What is static synchronization?

If you make any static method as synchronized, the lock will be on the class not on the object. If we use the synchronized keyword before a method so it will lock the object (one thread can access an object at a time) but if we use static synchronized so it will lock a class (one thread can access a class at a time). [More details.](https://www.javatpoint.com/static-synchronization-example)

### 25)What is the difference between notify() and notifyAll()?

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

### 26)What is the deadlock?

Deadlock is a situation in which every thread is waiting for a resource which is held by some other waiting thread. In this situation, Neither of the thread executes nor it gets the chance to be executed. Instead, there exists a universal waiting state among all the threads. Deadlock is a very complicated situation which can break our code at runtime.

[More details.](https://www.javatpoint.com/deadlock-in-java)

### 27) How to detect a deadlock condition? How can it be avoided?

We can detect the deadlock condition by running the code on cmd and collecting the Thread Dump, and if any deadlock is present in the code, then a message will appear on cmd.

**Ways to avoid the deadlock condition in Java:**

* **Avoid Nested lock:** Nested lock is the common reason for deadlock as deadlock occurs when we provide locks to various threads so we should give one lock to only one thread at some particular time.
* **Avoid unnecessary locks:** we must avoid the locks which are not required.
* **Using thread join:** Thread join helps to wait for a thread until another thread doesn't finish its execution so we can avoid deadlock by maximum use of join method.

### 28) What is Thread Scheduler in java?

In Java, when we create the threads, they are supervised with the help of a Thread Scheduler, which is the part of JVM. Thread scheduler is only responsible for deciding which thread should be executed. Thread scheduler uses two mechanisms for scheduling the threads: Preemptive and Time Slicing.

Java thread scheduler also works for deciding the following for a thread:

* It selects the priority of the thread.
* It determines the waiting time for a thread
* It checks the Nature of thread

### 29) Does each thread have its stack in multithreaded programming?

Yes, in multithreaded programming every thread maintains its own or separate stack area in memory due to which every thread is independent of each other.

### 30) How is the safety of a thread achieved?

If a method or class object can be used by multiple threads at a time without any race condition, then the class is thread-safe. Thread safety is used to make a program safe to use in multithreaded programming. It can be achieved by the following ways:

* Synchronization
* Using Volatile keyword
* Using a lock based mechanism
* Use of atomic wrapper classes

### 31) What is race-condition?

A Race condition is a problem which occurs in the multithreaded programming when various threads execute simultaneously accessing a shared resource at the same time. The proper use of synchronization can avoid the Race condition.

### 32) What is the volatile keyword in java?

Volatile keyword is used in multithreaded programming to achieve the thread safety, as a change in one volatile variable is visible to all other threads so one variable can be used by one thread at a time.

### 33) What do you understand by thread pool?

* Java Thread pool represents a group of worker threads, which are waiting for the task to be allocated.
* Threads in the thread pool are supervised by the service provider which pulls one thread from the pool and assign a job to it.
* After completion of the given task, thread again came to the thread pool.
* The size of the thread pool depends on the total number of threads kept at reserve for execution.

The advantages of the thread pool are :

* Using a thread pool, performance can be enhanced.
* Using a thread pool, better system stability can occur.

## **Concurrency Interview Questions**

### 34) What are the main components of concurrency API?

Concurrency API can be developed using the class and interfaces of java.util.Concurrent package. There are the following classes and interfaces in java.util.Concurrent package.

* Executor
* FarkJoinPool
* ExecutorService
* ScheduledExecutorService
* Future
* TimeUnit(Enum)
* CountDownLatch
* CyclicBarrier
* Semaphore
* ThreadFactory
* BlockingQueue
* DelayQueue
* Locks
* Phaser

### 35) What is the Executor interface in Concurrency API in Java?

The Executor Interface provided by the package java.util.concurrent is the simple interface used to execute the new task. The execute() method of Executor interface is used to execute some given command. The syntax of the execute() method is given below.

**void execute(Runnable command)**

Consider the following example:

1. **import** java.util.concurrent.Executor;
2. **import** java.util.concurrent.Executors;
3. **import** java.util.concurrent.ThreadPoolExecutor;
4. **import** java.util.concurrent.TimeUnit;
6. **public** **class** TestThread {
7. **public** **static** **void** main(**final** String[] arguments) **throws** InterruptedException {
8. Executor e = Executors.newCachedThreadPool();
9. e.execute(**new** Thread());
10. ThreadPoolExecutor pool = (ThreadPoolExecutor)e;
11. pool.shutdown();
12. }
14. **static** **class** Thread **implements** Runnable {
15. **public** **void** run() {
16. **try** {
17. Long duration = (**long**) (Math.random() \* 5);
18. System.out.println("Running Thread!");
19. TimeUnit.SECONDS.sleep(duration);
20. System.out.println("Thread Completed");
21. } **catch** (InterruptedException ex) {
22. ex.printStackTrace();
23. }
24. }
25. }
26. }

**Output**

Running Thread!

Thread Completed

### 36) What is BlockingQueue?

The java.util.concurrent.BlockingQueue is the subinterface of Queue that supports the operations such as waiting for the space availability before inserting a new value or waiting for the queue to become non-empty before retrieving an element from it. Consider the following example.

2. **import** java.util.Random;
3. **import** java.util.concurrent.ArrayBlockingQueue;
4. **import** java.util.concurrent.BlockingQueue;
6. **public** **class** TestThread {
8. **public** **static** **void** main(**final** String[] arguments) **throws** InterruptedException {
9. BlockingQueue<Integer> queue = **new** ArrayBlockingQueue<Integer>(10);
11. Insert i = **new** Insert(queue);
12. Retrieve r = **new** Retrieve(queue);
14. **new** Thread(i).start();
15. **new** Thread(r).start();
17. Thread.sleep(2000);
18. }

21. **static** **class** Insert **implements** Runnable {
22. **private** BlockingQueue<Integer> queue;
24. **public** Insert(BlockingQueue queue) {
25. **this**.queue = queue;
26. }
28. @Override
29. **public** **void** run() {
30. Random random = **new** Random();
32. **try** {
33. **int** result = random.nextInt(200);
34. Thread.sleep(1000);
35. queue.put(result);
36. System.out.println("Added: " + result);
38. result = random.nextInt(10);
39. Thread.sleep(1000);
40. queue.put(result);
41. System.out.println("Added: " + result);
43. result = random.nextInt(50);
44. Thread.sleep(1000);
45. queue.put(result);
46. System.out.println("Added: " + result);
47. } **catch** (InterruptedException e) {
48. e.printStackTrace();
49. }
50. }
51. }
53. **static** **class** Retrieve **implements** Runnable {
54. **private** BlockingQueue<Integer> queue;
56. **public** Retrieve(BlockingQueue queue) {
57. **this**.queue = queue;
58. }
60. @Override
61. **public** **void** run() {
63. **try** {
64. System.out.println("Removed: " + queue.take());
65. System.out.println("Removed: " + queue.take());
66. System.out.println("Removed: " + queue.take());
67. } **catch** (InterruptedException e) {
68. e.printStackTrace();
69. }
70. }
71. }
72. }

**Output**

Added: 96

Removed: 96

Added: 8

Removed: 8

Added: 5

Removed: 5

### 37) How to implement producer-consumer problem by using BlockingQueue?

The producer-consumer problem can be solved by using BlockingQueue in the following way.

2. **import** java.util.concurrent.BlockingQueue;
3. **import** java.util.concurrent.LinkedBlockingQueue;
4. **import** java.util.logging.Level;
5. **import** java.util.logging.Logger;
6. **public** **class** ProducerConsumerProblem {
7. **public** **static** **void** main(String args[]){
8. //Creating shared object
9. BlockingQueue sharedQueue = **new** LinkedBlockingQueue();
11. //Creating Producer and Consumer Thread
12. Thread prod = **new** Thread(**new** Producer(sharedQueue));
13. Thread cons = **new** Thread(**new** Consumer(sharedQueue));
15. //Starting producer and Consumer thread
16. prod.start();
17. cons.start();
18. }
20. }
22. //Producer Class in java
23. **class** Producer **implements** Runnable {
25. **private** **final** BlockingQueue sharedQueue;
27. **public** Producer(BlockingQueue sharedQueue) {
28. **this**.sharedQueue = sharedQueue;
29. }
31. @Override
32. **public** **void** run() {
33. **for**(**int** i=0; i<10; i++){
34. **try** {
35. System.out.println("Produced: " + i);
36. sharedQueue.put(i);
37. } **catch** (InterruptedException ex) {
38. Logger.getLogger(Producer.**class**.getName()).log(Level.SEVERE, **null**, ex);
39. }
40. }
41. }
43. }
45. //Consumer Class in Java
46. **class** Consumer **implements** Runnable{
48. **private** **final** BlockingQueue sharedQueue;
50. **public** Consumer (BlockingQueue sharedQueue) {
51. **this**.sharedQueue = sharedQueue;
52. }
54. @Override
55. **public** **void** run() {
56. **while**(**true**){
57. **try** {
58. System.out.println("Consumed: "+ sharedQueue.take());
59. } **catch** (InterruptedException ex) {
60. Logger.getLogger(Consumer.**class**.getName()).log(Level.SEVERE, **null**, ex);
61. }
62. }
63. }
64. }

**Output**

Produced: 0

Produced: 1

Produced: 2

Produced: 3

Produced: 4

Produced: 5

Produced: 6

Produced: 7

Produced: 8

Produced: 9

Consumed: 0

Consumed: 1

Consumed: 2

Consumed: 3

Consumed: 4

Consumed: 5

Consumed: 6

Consumed: 7

Consumed: 8

Consumed: 9

### 38) What is the difference between Java Callable interface and Runnable interface?

The Callable interface and Runnable interface both are used by the classes which wanted to execute with multiple threads. However, there are two main differences between the both :

* A Callable <V> interface can return a result, whereas the Runnable interface cannot return any result.
* A Callable <V> interface can throw a checked exception, whereas the Runnable interface cannot throw checked exception.
* A Callable <V> interface cannot be used before the Java 5 whereas the Runnable interface can be used.

### 39) What is the Atomic action in Concurrency in Java?

* The Atomic action is the operation which can be performed in a single unit of a task without any interference of the other operations.
* The Atomic action cannot be stopped in between the task. Once started it fill stop after the completion of the task only.
* An increment operation such as a++ does not allow an atomic action.
* All reads and writes operation for the primitive variable (except long and double) are the atomic operation.
* All reads and writes operation for the volatile variable (including long and double) are the atomic operation.
* The Atomic methods are available in java.util.Concurrent package.

### 40) What is lock interface in Concurrency API in Java?

The java.util.concurrent.locks.Lock interface is used as the synchronization mechanism. It works similar to the synchronized block. There are a few differences between the lock and synchronized block that are given below.

* Lock interface provides the guarantee of sequence in which the waiting thread will be given the access, whereas the synchronized block doesn't guarantee it.
* Lock interface provides the option of timeout if the lock is not granted whereas the synchronized block doesn't provide that.
* The methods of Lock interface, i.e., Lock() and Unlock() can be called in different methods whereas single synchronized block must be fully contained in a single method.

### 41) Explain the ExecutorService Interface.

The ExecutorService Interface is the subinterface of Executor interface and adds the features to manage the lifecycle. Consider the following example.

2. **import** java.util.concurrent.ExecutorService;
3. **import** java.util.concurrent.Executors;
4. **import** java.util.concurrent.TimeUnit;
6. **public** **class** TestThread {
7. **public** **static** **void** main(**final** String[] arguments) **throws** InterruptedException {
8. ExecutorService e = Executors.newSingleThreadExecutor();
10. **try** {
11. e.submit(**new** Thread());
12. System.out.println("Shutdown executor");
13. e.shutdown();
14. e.awaitTermination(5, TimeUnit.SECONDS);
15. } **catch** (InterruptedException ex) {
16. System.err.println("tasks interrupted");
17. } **finally** {
19. **if** (!e.isTerminated()) {
20. System.err.println("cancel non-finished tasks");
21. }
22. e.shutdownNow();
23. System.out.println("shutdown finished");
24. }
25. }
27. **static** **class** Task **implements** Runnable {
29. **public** **void** run() {
31. **try** {
32. Long duration = (**long**) (Math.random() \* 20);
33. System.out.println("Running Task!");
34. TimeUnit.SECONDS.sleep(duration);
35. } **catch** (InterruptedException ex) {
36. ex.printStackTrace();
37. }
38. }
39. }
40. }

**Output**

Shutdown executor

shutdown finished

### 42) What is the difference between Synchronous programming and Asynchronous programming regarding a thread?

**Synchronous programming:**In Synchronous programming model, a thread is assigned to complete a task and hence thread started working on it, and it is only available for other tasks once it will end the assigned task.

**Asynchronous Programming:**In Asynchronous programming, one job can be completed by multiple threads and hence it provides maximum usability of the various threads.

### 43) What do you understand by Callable and Future in Java?

**Java Callable interface:**In Java5 callable interface was provided by the package java.util.concurrent. It is similar to the Runnable interface but it can return a result, and it can throw an Exception. It also provides a run() method for execution of a thread. Java Callable can return any object as it uses Generic.

**Syntax:**

public interface Callable<V>

**Java Future interface:** Java Future interface gives the result of a concurrent process. The Callable interface returns the object of java.util.concurrent.Future.

Java Future provides following methods for implementation.

* **cancel(boolean mayInterruptIfRunning):** It is used to cancel the execution of the assigned task.
* **get():** It waits for the time if execution not completed and then retrieved the result.
* **isCancelled():** It returns the Boolean value as it returns true if the task was canceled before the completion.
* **isDone():** It returns true if the job is completed successfully else returns false.

### 44. What is the difference between ScheduledExecutorService and ExecutorService interface?

ExecutorServcie and ScheduledExecutorService both are the interfaces of java.util.Concurrent package but scheduledExecutorService provides some additional methods to execute the Runnable and Callable tasks with the delay or every fixed time period.

### 45) Define FutureTask class in Java?

Java FutureTask class provides a base implementation of the Future interface. The result can only be obtained if the execution of one task is completed, and if the computation is not achieved then get method will be blocked. If the execution is completed, then it cannot be re-started and can't be canceled.

**Syntax**

public class FutureTask<V> extends Object implements RunnableFuture<V>

* **Collections**

### ****1. What is Collection in Java?****

The term **collection** refers to a group of objects represented as one unit. Classes in the Java collection class hierarchy are divided into two “**root**” interfaces: Collection **(java.util.Collection)** and Map **(java.util.Map)**. Terms that you will encounter while learning about the collection in Java:

* **Collection Framework:**Java’s Collection Framework defines classes and interfaces for representing groups of objects as a single entity. C++ developers can compare the Collection framework with [**STL (Standard Template Library)**](https://www.geeksforgeeks.org/the-c-standard-template-library-stl/) and Container Framework with the Collection Framework if they come from a C++ background.
* **Collection Interface:** A class’s interface specifies what it should do, not how. In other words, it is the blueprint for the class. This interface provides the most common methods for all collection objects that are part of the Collection Framework. Alternatively, it represents the individual object as a whole.
* **Collections Class:**A member of Collection Framework, it is part of java.util package. The collection object is provided with many utility methods in this class.

### ****2. What is a Framework in Java?****

Frameworks are sets of [classes](https://www.geeksforgeeks.org/classes-objects-java/)and [interfaces](https://www.geeksforgeeks.org/interfaces-in-java/) that provide a ready-made architecture. It is not necessary to define a framework in order to implement new features or classes. As a result, an optimal object-oriented design includes a framework containing a collection of classes that all perform similar tasks. The framework can be used in a variety of ways, such as by calling its methods, extending it, and supplying “callbacks”, listeners, and other implementations. Some of the popular frameworks in java are:

* Spring
* Hibernate
* Struts
* Google Web Toolkit (GWT)
* JavaServer Faces (JSF)

### ****3. What is the difference between Array and Collection in Java?****

Arrays are a collection of similar-typed variables with a common name in Java. There are some differences between arrays in Java and C/C++. On the other hand, Collections are groups of individual objects that form a single entity known as the collection of objects.

| **Arrays** | **Collection** |
| --- | --- |
| Arrays are fixed in size that is once we create an array we can not increase or decrease based on our requirements. | The collection is growable in nature and is based on our requirements. We can increase or decrease of size. |
| With respect to memory, Arrays are not recommended for use. | With respect to memory, collections are recommended for use. |
| With respect to performance, Arrays are recommended for use. | With respect to performance, collections are not recommended for use. |
| Arrays can hold only homogeneous data types elements. | Collection can hold both homogeneous and heterogeneous elements. |

*For more information, refer to the article –* [Difference Between Arrays and Collections in Java](https://www.geeksforgeeks.org/difference-between-arrays-and-collection-in-java/)

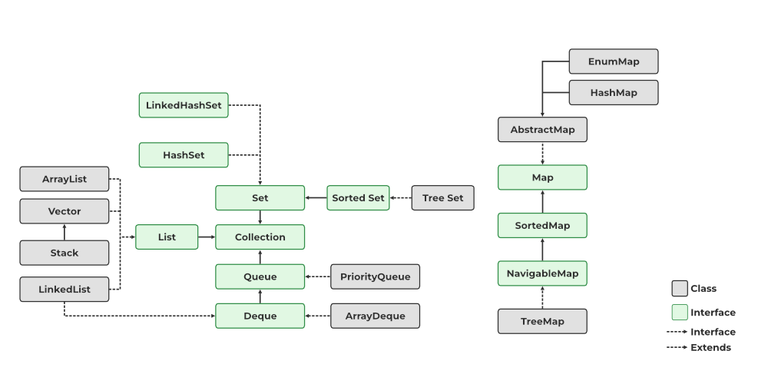
### ****4.  What are the various interfaces used in Java Collections Framework?****

The collection is known as the root of the collection hierarchy. Collections represent groups of objects known as elements. The java platform does not provide any direct implementation of this interface but the Collection interface is being implemented by List and Set classes.

* **Collection interface**
* **List interface**
* **Set interface**
* **Queue interface**
* **Dequeue interface**
* **Map interface**

### ****5. Explain the hierarchy of the Collection framework in Java.****

All classes and interfaces required by the collection framework are contained in the utility package (java. util). Collection frameworks have an interface called an iterable interface, which allows the iterator to iterate over all collections. In addition to this interface, the main collection interface acts as a root for the collection framework. All the collections extend this collectlion interface thereby extending the properties of the iterator and the methods of this interface. The following figure illustrates the hierarchy of the collection framework.

e

*Java Collection Hierarchy*

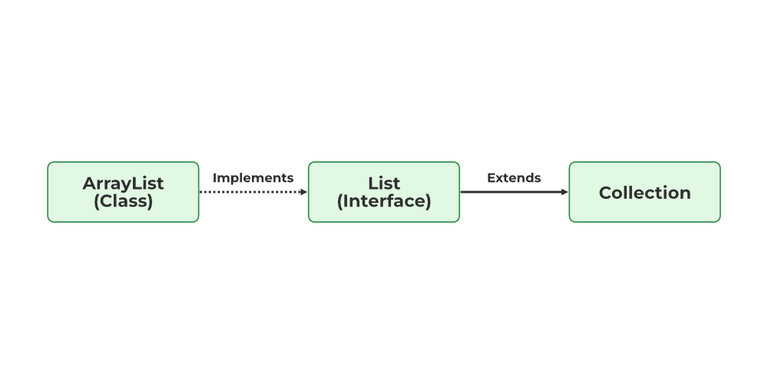
### 6. What are the advantages of the collection Framework?

**Advantages of the Collection Framework:** Since the lack of a collection framework gave rise to the above set of disadvantages, the following are the advantages of the collection framework.

* **Consistent API:** The API has a basic set of interfaces like Collection, Set, List, or Map, all the classes (ArrayList, LinkedList, Vector, etc) that implement these interfaces have some common set of methods.
* **Reduces programming effort:** A programmer doesn’t have to worry about the design of the Collection but rather he can focus on its best use in his program. Therefore, the basic concept of Object-oriented programming (i.e.) abstraction has been successfully implemented.
* **Increases program speed and quality:** Increases performance by providing high-performance implementations of useful data structures and algorithms because in this case, the programmer need not think of the best implementation of a specific data structure. He can simply use the best implementation to drastically boost the performance of his algorithm/program.

### 7. What is ArrayList in Java?

ArrayList is a part of the Java collection framework and it is a class of java.util package. It provides us with dynamic arrays in Java. The main advantages of ArrayList are, if we declare an array then it’s needed to mention the size but in ArrayList, it is not needed to mention the size of ArrayList if you want to mention the size then you can do it.



*Image of Array List*

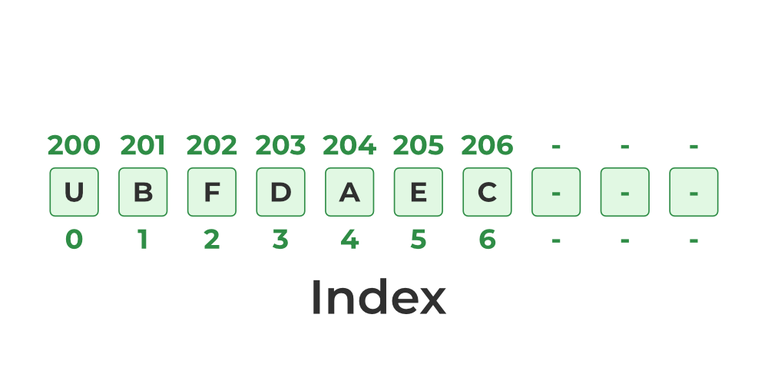
*For more information, refer to the article –*[*ArrayList in Java*](https://www.geeksforgeeks.org/arraylist-in-java/)

### 8. What is the difference between Collection and Collections?

| **Collection** | **Collections** |
| --- | --- |
| It is an interface. | It is a utility class. |
| It is used to represent a group of individual objects as a single unit. | It defines several utility methods that are used to operate on collection. |
| The Collection is an interface that contains a static method since java8. The Interface can also contain abstract and default methods. | It contains only static methods. |

*For more information, refer to the article –*[*Collection vs Collections in Java with Example*](https://www.geeksforgeeks.org/collection-vs-collections-in-java-with-example/)

### 9. Difference between ArrayList and LinkedList in the java collection framework?



*ArrayList and LinkedList*

| **ArrayList** | **LinkedList** |
| --- | --- |
| This class uses a dynamic array to store the elements in it. With the introduction of generics, this class supports the storage of all types of objects. | This class uses a doubly linked list to store the elements in it. Similar to the ArrayList, this class also supports the storage of all types of objects. |
| Manipulating ArrayList takes more time due to the internal implementation. Whenever we remove an element, internally, the array is traversed and the memory bits are shifted. | Manipulating LinkedList takes less time compared to ArrayList because, in a doubly-linked list, there is no concept of shifting the memory bits. The list is traversed and the reference link is changed. |
| This class implements a List interface. Therefore, this acts as a list. | This class implements both the List interface and the Deque interface. Therefore, it can act as a list and a deque. |
| This class works better when the application demands storing the data and accessing it. | This class works better when the application demands manipulation of the stored data. |

*For more information, refer to the article –*[*ArrayList vs LinkedList in Java*](https://www.geeksforgeeks.org/arraylist-vs-linkedlist-java/)

### 10. What is an iterator?

**Java’s Collection framework** uses iterators to retrieve elements one by one. This iterator is universal since it can be used with any type of Collection object. Using Iterator, we can perform both reading and removing operations. This is an improved version of Enumeration with the addition of removing elements.

When enumerating elements in all Collection framework implemented interfaces, such as **Set**, **List**, **Queue**, **Deque**, and all implemented classes of Map, an Iterator must be used. The only cursor available for the entire collection framework is the iterator. Using the iterator() method in the Collection interface, you can create an iterator object.

**Syntax:**

Iterator itr = c.**iterator**();

***Note:*** Here “c” is any Collection object. itr is of type Iterator interface and refers to “c”.

*For more information, refer to the article –*[*Iterators in Java*](https://www.geeksforgeeks.org/iterators-in-java/)

### 11. What is the difference between an Iterator and an Enumeration?

A major difference between iterator and enumeration is that iterators have a remove() method while enumerations do not. Thus, using Iterator we can manipulate objects by adding and removing them from collections. Since enumeration can only traverse objects and fetch them, it behaves like a read-only interface.

*For more information, refer to the article –*[*Difference between Iterator and Enumeration*](https://www.geeksforgeeks.org/difference-between-iterator-and-enumeration-in-java-with-examples/#:~:text=Iterator%20can%20do%20modifications%20(e.g,the%20elements%20of%20the%20Collection.)

### 12. What is the difference between List and Set in Java

A major difference between a List and a Set is that a List can contain duplicate elements while a set contains only unique elements. The list is Ordered and maintains the order of the object to which they are added. The set is unordered.

| **List** | **Set** |
| --- | --- |
| The List is an indexed sequence. | The Set is a non-indexed sequence. |
| The list allows duplicate elements | The set doesn’t allow duplicate elements. |
| Elements by their position can be accessed. | Position access to elements is not allowed. |
| Multiple null elements can be stored. | Null elements can store only once. |
| List implementations are ArrayList, LinkedList, Vector, Stack | Set implementations are HashSet, LinkedHashSet. |

*For more information, refer to the article –*[*Difference Between List and Set in Java*](https://www.geeksforgeeks.org/difference-between-list-and-set-in-java/)

### 13. What are the best practices for Java Collections Framework?

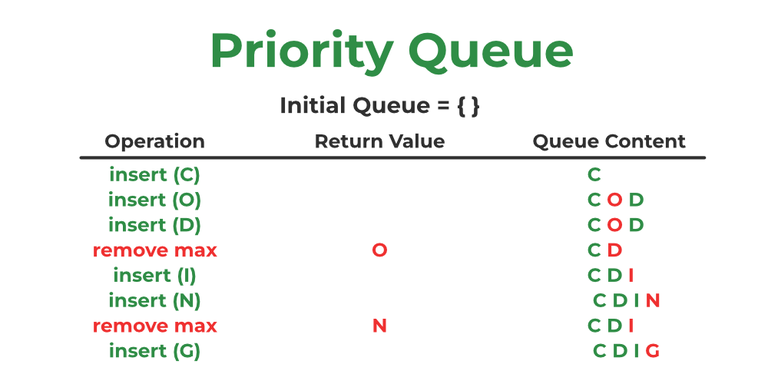
Following are some of the best practices while using Java Collections:

* Programs should be written as interfaces, not implementations, so we can modify the implementation later.
* Whenever possible, use Generics to ensure type safety and avoid ClassCastExceptions.
* Choosing the appropriate type of collection based on the need. For example, if the size is fixed, we might want to use an Array over an ArrayList. When iterating over the Map, we should use LinkedHashMap. Set is the best way to avoid duplicates.
* Use immutable classes provided by JDK as keys in Map to avoid implementation of hashCode() and equals().
* In order to increase the readability of the code, we should use isEmpty() instead of finding the size of the collection and comparing it to zero.
* Rather than writing your own implementation, use the Collections utility class to get read-only, Synchronized, or empty collections instead. It enhances code reuse while resulting in greater stability.

### 14. What is a priority queue in Java?

PriorityQueues are used to process objects according to their priority. Queues follow the First-In-First-Out algorithm, but sometimes the elements of the queue need to be processed according to their priority, which is where PriorityQueue comes into play. Priority queues are based on priority heaps.

The elements of the priority queue are ordered according to the natural ordering, or by a Comparator provided at queue construction time, depending on which constructor is used.



*Priority Queues in Java*

**Declaration:**

public class PriorityQueue<E> extends AbstractQueue<E> implements Serializable

where E is the type of elements held in this queue

The class implements Serializable, Iterable<E>, Collection<E>, and Queue<E> interfaces.

### 15. What is the difference between List, set, and map in java?

| **List** | **Set** | **Map** |
| --- | --- | --- |
| The list interface allows duplicate elements | The set does not allow duplicate elements. | The map does not allow duplicate elements |
| The list maintains insertion order. | The set does not maintain any insertion order. | The map also does not maintain any insertion order. |
| We can add any number of null values. | But in the set almost only one null value. | The map allows a single null key at most and any number of null values. |
| The list implementation classes are Array List and LinkedList. | Set implementation classes are HashSet, LinkedHashSet, and TreeSet. | Map implementation classes are HashMap, HashTable, TreeMap, ConcurrentHashMap, and LinkedHashMap. |

*For more information, refer to the article –*[*Difference between List, Set, and Map in Java*](https://www.geeksforgeeks.org/difference-between-list-set-and-map-in-java/)

### 16. What is the difference between Queue and Stack?

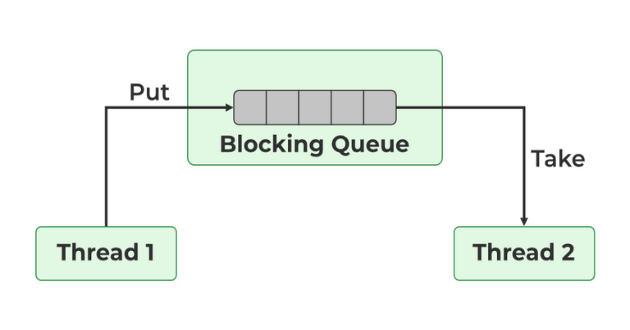
| **Stack** | **Queue** |
| --- | --- |
| Stacks works on the LIFO principle, which means that the element inserted at the last will be the first element that will be taken out. | Queues work on the FIFO principle, which means that the element inserted first will be the first element that will be taken out. |
| In stacks, insertion, and deletions take place only from the top. | In queues, insertion occurs at the rear of the list and deletion takes place from the front of the list. |
| Insert operation is called push operation. | Insert operation is called enqueue operation. |
| Delete operation is called pop operation. | Delete operation is called dequeue operation. |
| The top of a stack always points to the last element in the list, which is the only pointer used to access the list. | Two pointers are maintained for accessing queues. The front pointer points to the first inserted element, and the rear pointer points to the last inserted element. |

### 17. What is BlockingQueue in Java?

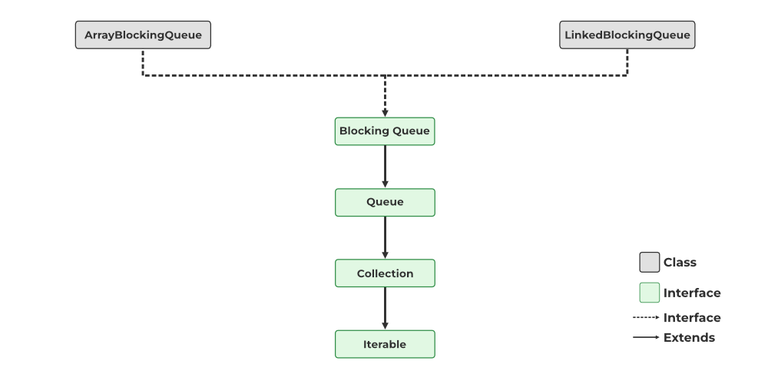
The BlockingQueue interface in Java is added in Java 1.5 along with various other concurrent Utility classes like ConcurrentHashMap, Counting Semaphore, CopyOnWriteArrrayList, etc. The BlockingQueue interface supports flow control (in addition to queue) by introducing blocking if either BlockingQueue is full or empty.

A thread trying to enqueue an element in a full queue is blocked until some other thread makes space in the queue, either by dequeuing one or more elements or clearing the queue completely. Similarly, it blocks a thread trying to delete from an empty queue until some other threads insert an item. BlockingQueue does not accept a null value. If we try to enqueue the null item, then it throws NullPointerException.

**Usage of BlockingQueue**



**The Hierarchy of BlockingQueue**



*Hierarchy of Blocking Queue in Java*

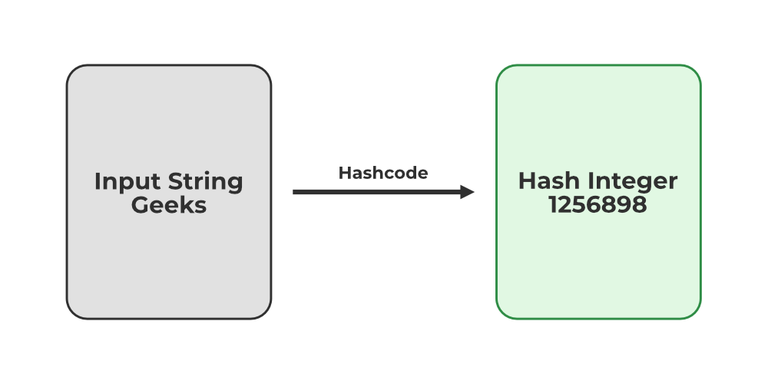
**Declaration:**

public interface BlockingQueue<E> extends Queue<E>

Here,**E**is the type of elements stored in the Collection.

*For more information, refer to the article –*[*BlockingQueue Interface in Java*](https://www.geeksforgeeks.org/blockingqueue-interface-in-java/)

### 18. What is the hashCode()?



*Image to demonstrate Java Hash Code*

hashCode() method returns the hashcode value as an Integer. It is defined in the Java Object class which computes the hash values of given input objects. Hashcode value is mostly used in hashing-based collections like HashMap, HashSet, HashTable….etc. This method must be overridden in every class which overrides the equals() method.

**Syntax :**

public int hashCode()

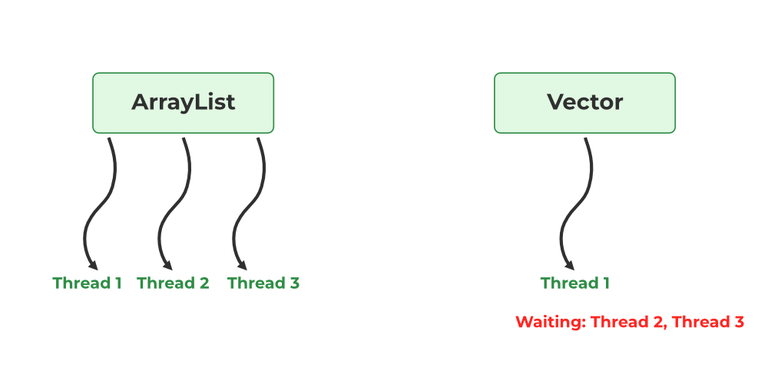
// This method returns the hash code value

// for the object on which this method is invoked.

*For more information, refer to the article –* [equals() and hashCode() methods in Java](https://www.geeksforgeeks.org/equals-hashcode-methods-java/)

### 19. Distinguish between ArrayList and Vector in the Java Collection Framework.

In collection interviews, this question is frequently asked; however, Vector is synchronized whereas ArrayList is not. ArrayList is faster than Vector. ArrayList’s Array size is increased by 50% when needed, while Vector’s capacity is doubled whenever it is needed.



*Array List vs Vector in java*

| **ArrayList** | **Vector** |
| --- | --- |
| ArrayList is not Synchronized | The vector is synchronized. |
| The size of ArrayList is incremented up to 50% of the current array size if the number of elements exceeds its capacity. | The size of ArrayList is incremented up to 100% of the current array size if the number of elements exceeds its capacity. |
| ArrayList is fast because it is non-Synchronized. | Vector is slower because it’s synchronized. |
| The iterator interface is used to traverse the elements | An iterator interface or Enumeration can be used to traverse the vector. |

*For more information, refer to the article –* [*Vector vs ArrayList in Java*](https://www.geeksforgeeks.org/vector-vs-arraylist-java/)

### 20. Differentiate between Iterator and ListIterator.

| **Iterator** | **ListIterator** |
| --- | --- |
| Can traverse elements present in Collection only in the forward direction. | Can traverse elements present in the Collection both in the forward and backward directions. |
| Helps to traverse Map, List, and Set. | Can only traverse List and not the other two. |
| Indexes cannot be obtained by using Iterator. | It has methods like nextIndex() and previousIndex() to obtain indexes of elements at any time while traversing the List. |
| Cannot modify or replace elements present in the Collection | We can modify or replace elements with the help of set(E e) |

*For more information, refer to the article –*[Difference Between an Iterator and ListIterator](https://www.geeksforgeeks.org/difference-between-an-iterator-and-listiterator-in-java/)

### 21. What is the difference between an Iterator and an Enumeration?

**Iterator:**It is a universal iterator as we can apply it to any Collection object. By using an Iterator, we can perform both read and remove operations.

**Syntax:**

// Here "c" is any Collection object. itr is of

// type Iterator interface and refers to "c"

Iterator itr = c.iterator();

**Enumeration:**Enumeration (or enum) is a user-defined data type. It is mainly used to assign names to integral constants, the names make a program easy to read and maintain.

**Syntax:**

// A simple enum example where enum is declared

// outside any class (Note enum keyword instead of

// class keyword)

enum Color

{

RED, GREEN, BLUE;

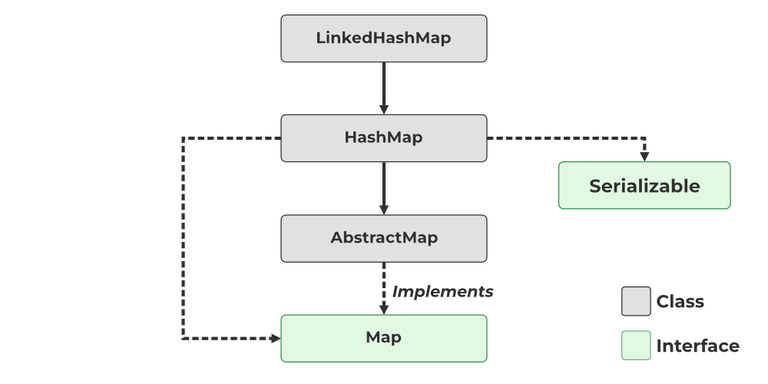
}

| **Iterator** | **Enumeration** |
| --- | --- |
| The iterator is a universal cursor as it is applicable to all the collection classes. | Enumeration is not a universal cursor as it applies only to legacy classes. |
| The iterator has the remove() method. | Enumeration does not have the remove() method. |
| The iterator can do modifications (e.g using the remove() method which removes the element from the Collection during traversal). | The enumeration interface acts as a read-only interface, one can not do any modifications to the Collection while traversing the elements of the Collection. |
| Iterator is not a legacy interface. Iterator can be used for the traversal of HashMap, LinkedList, ArrayList, HashSet, TreeMap, and TreeSet. | Enumeration is a legacy interface that is used for traversing Vector, and Hashtable. |

*For more information, refer to the article –* [Difference between Iterator and Enumeration](https://www.geeksforgeeks.org/difference-between-iterator-and-enumeration-in-java-with-examples/#:~:text=Iterator%20can%20do%20modifications%20(e.g,the%20elements%20of%20the%20Collection.)

### 22. What are the features of Java Hashmap?

HashMap is similar to HashTable, but it is unsynchronized. It allows us to store the null keys as well, but there should be only one null key object and there can be any number of null values.  This class makes no guarantees as to the order of the map. To use this class and its methods, you need to import **java.util. HashMap** package or its superclass.



*HashMap in Java*

**Syntax:**

**public class** HashMap<K,V> **extends** AbstractMap<K,V> **implements** Map<K,V>, Cloneable, Serializable

**Parameters:**It takes two parameters namely as follows:

* The type of keys maintained by this map (K)
* The type of mapped values (V)

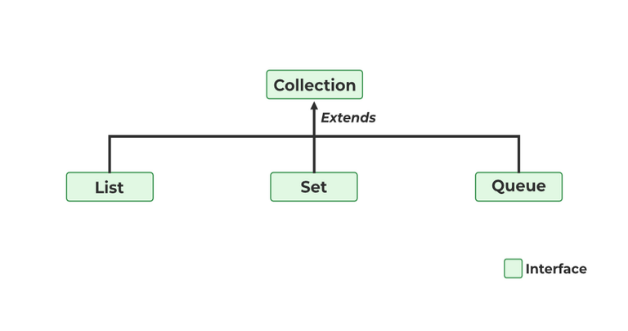
*For more information, refer to the article –*[HashMap in Java with Examples](https://www.geeksforgeeks.org/java-util-hashmap-in-java-with-examples/)

### 23. What are Collection Interfaces?

The **Collection**interface is a member of the Java Collections Framework. It is a part of **java.util** package. It is one of the root interfaces of the Collection Hierarchy. The Collection interface is not directly implemented by any class. However, it is implemented indirectly via its subtypes or subinterfaces like List, Queue, and Set.

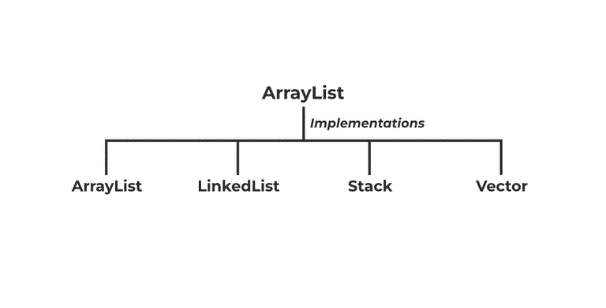
**For Example,** the HashSet class implements the Set interface which is a subinterface of the Collection interface. If a collection implementation doesn’t implement a particular operation, it should define the corresponding method to throw UnsupportedOperationException.

**The Hierarchy of Collection:**



*Collection Interface in Java*

### 24. Explain the list interface.



*Class Interface in Java*

In Java, the List interface allows the user to store an ordered collection of objects. The list is the child interface of Collection. In Collection, a list is an ordered collection of objects which can have duplicate values. Since List preserves the insertion order, it allows positional access and insertion, which also allows duplicate values.

**Syntax:**

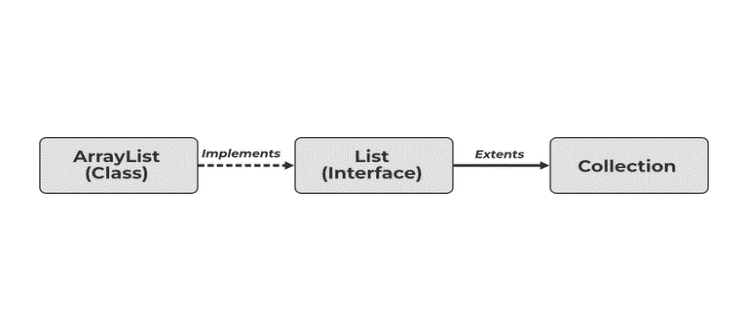
public interface List<E> extends Collection<E> ;

This list interface is implemented by various classes like ArrayList, Vector, Stack, etc. Since all the subclasses implement the list, we can instantiate a list object with any of these classes.

**Example:**

List <T> al = new ArrayList<> (); List <T> ll = new LinkedList<> (); List <T> v = new Vector<> ();

Where T is the type of the object



The classes which implement the List interface are as follows:

* ArrayList
* LinkedList
* Vector
* Stack

### 25. Write a program to convert a given array into a collection with the asList() method.

To convert array-based data into Collection based we can use **java.util.Arrays**class. This class provides a static method asList(T… a) that converts the array into a Collection.

* Java

|  |
| --- |
| // Convert an Array into Collection in Java  // import java util library  **import** java.util.\*;    // class for writing logic of the problem  **public** **class** ArrayToCollection {  **public** **static** **void** main(String args[])      {          // array input          String students[] = { "Kamlesh", "Abhay",                                "Abhishek", "Shivansh" };            // printing input elements for comparison          System.out.println("Array input: "                             + Arrays.toString(students));            // converting array into Collection          // with asList() function          List studentList = Arrays.asList(students);            // print converted elements          System.out.println("Converted elements: "                             + studentList);      }  } |

**Output**

Array input: [Kamlesh, Abhay, Abhishek, Shivansh]

Converted elements: [Kamlesh, Abhay, Abhishek, Shivansh]

### 26. Differentiate between HashSet and HashMap

| **HashSet** | **HashMap** |
| --- | --- |
| HashSet implements the Set interface | HashMap implements the Map interface |
| No Duplicates are allowed | Yes duplicates values are allowed but no duplicate key is allowed |
| Dummy values are allowed in HashSet. | No Dummy values are allowed in HashMap. |
| A single Object is required during an add operation | 2 Objects are required during an add operation |
| Speed is comparatively slower than HashMap | Speed is comparatively faster than HashSet because of hashing technique has been used here. |
| Have a single null value | Single null key and any number of null values |
| Add() method is used for the insertion | The put () method is used for insertion. |

*For more information, refer to the article –*[Difference between HashMap and HashSet](https://www.geeksforgeeks.org/difference-between-hashmap-and-hashset)

### 27. Differentiate between HashSet and HashTable.

| **HashSet** | **HashTable** |
| --- | --- |
| HashSet allows NULL Elements | HashTable does not allow NULL Elements. |
| Objects that you insert in HashSet are not guaranteed to be inserted in the same order. Objects are inserted based on their hash code. LinkedHashSet can be used to maintain order. | HashTable does not maintain insertion order. |
| HashSet is not Synchronized but it can be synchronized externally. | HashTable is Synchronized. |
| add() method is used to insert into HashSet | put() method is used to insert into HashTable |

### 28. What is the default size of the load factor in the hashing-based collection?

As the Load Factor increases, the capacity increases so that the operational complexity of the HashMap remains O(1) if the ratio of the current element to the initial capacity crosses the threshold. The meaning of operational complexity of O(1) means the retrieval and insertion operations take constant time. The default load factor size is **0.75**. The default capacity is calculated by multiplying the initial capacity by the load factor.

*For more information, refer to the article –*[*Load Factor in HashMap in Java with Examples*](https://www.geeksforgeeks.org/load-factor-in-hashmap-in-java-with-examples/)

## Java Collection Interview Questions For Experienced

### 29. What is the difference between Comparable and Comparator in Java?

Java provides two interfaces to sort objects using data members of the class:

* Comparable
* Comparator

| **Comparable** | **Comparator** |
| --- | --- |
| The Comparable interface provides a single sorting sequence. | The Comparator interface provides multiple sorting sequences. |
| The actual class is modified by a comparable interface | The actual class is not modified by the Comparator interface. |
| compareTo() method is used to sort elements. | compare() method is used to sort elements. |
| Comparable is present in the package java.lang | Comparator is present in the package java.util |

*For more information, refer to the article –*[*Comparable vs Comparator in Java*](https://www.geeksforgeeks.org/comparable-vs-comparator-in-java/)

### 30. What is the difference between fail-fast and failsafe?

Iterators in Java are used to iterate over the Collection objects. Fail-Fast iterators immediately throw ConcurrentModificationException if there is a **structural modification** of the collection. Structural modification means adding, or removing any element from a collection while a thread is iterating over that collection. Iterator on ArrayList and HashMap classes are some examples of fail-fast Iterator.

| **Fail-Fast** | **Fail-Safe** |
| --- | --- |
| ConcurrentModificationException is thrown while modifying the object during the iteration process. | No Exception is thrown |
| Fail-Fast needs less memory during the process. | Fail-Safe iterator requires more memory during the process. |
| A clone Object is not created during the iteration process. | A clone Object or a Copy is created during the iteration process. |
| Fail-Fast does not allow modification during the process of iteration. | Fail-Safe allows modification during the process of iteration. |
| Fail-Fast is fast, | Fail-Safe is slightly slower than fail fast. |
| **Examples:**  ArrayList, Vector, HashMap, HashSet, etc. | **Examples:**  ConcurrentHashMap, CopyOnWriteArrayList, etc. |

*For more information, refer to the article –*[*Fail Fast and Fail Safe Iterators in Java*](https://www.geeksforgeeks.org/fail-fast-fail-safe-iterators-java/)

### 31. Write a program to iterate the list using the lambda expression.

Iteration can be done using alambda expression.

**Syntax:**

list\_name.forEach(variable->{//block of code})

* Java

|  |
| --- |
| // Java Program to iterate over a List  // using forEach()    // Importing all classes of  // java.util method  **import** java.util.\*;    // Class  **class** GFG {        // Main driver method  **public** **static** **void** main(String args[])      {          // Creating an ArrayList          List<String> l = **new** ArrayList<String>();            // Adding elements to the List          // Custom inputs          l.add("Geeks");          l.add("for");          l.add("Geeks");            // Lambda expression printing all elements in a List          l.forEach((temp) -> { System.out.println(temp); });      }  } |

**Output**

Geeks

for

Geeks

### 32. What is IdentityHashMap?

The IdentityHashMap implements the Map interface using Hashtable, comparing keys (and values) using reference equality instead of object equality. This class implements the Map interface, but it intentionally breaks Map’s general contract, which demands that objects are compared using the equals() method. This class is used when the user allows objects to be compared using references. It belongs to java.util package.

For more information, refer to the article – [IdentityHashMap class in Java](https://www.geeksforgeeks.org/identityhashmap-class-java/)

### 33. Write a program in Java to display the contents of a HashTable using enumeration.

The hashtable class implements a hash table, which maps keys to values. Any non-null object can be used as a key or as a value. To successfully store and retrieve objects from a hashtable, the objects used as keys must implement the hashCode method and the equals method. Below is the program to display the contents of a HashTable using enumeration:

* Java

|  |
| --- |
| // Java Program to Demonstrate Getting Values  // as an Enumeration of Hashtable class    **import** java.io.\*;  **import** java.util.Enumeration;  **import** java.util.Hashtable;    // Main class  // EnumerationOnKeys  **public** **class** GFG {        // Main driver method  **public** **static** **void** main(String[] args)      {          // Creating an empty hashtable          Hashtable<Integer, String> hash              = **new** Hashtable<Integer, String>();            // Inserting key-value pairs into hash table          // using put() method          hash.put(1, "Geeks");          hash.put(2, "for");          hash.put(3, "Geeks");            // Now creating an Enumeration object          //  to read elements          Enumeration e = hash.elements();            // Condition holds true till there is          // single key remaining            // Printing elements of hashtable          // using enumeration  **while** (e.hasMoreElements()) {                // Printing the current element              System.out.println(e.nextElement());          }      }  } |

**Output**

Geeks

for

Geeks

### 34. Write a program in java to get the collection view of the values present in a HashMap.

Java’s HashMap class has the java.util.HashMap.values() method for creating collections out of HashMap values. It basically returns a Collection view of HashMap values.

* Java

|  |
| --- |
| // Java code to illustrate the values() method  **import** java.util.\*;    **public** **class** Hash\_Map\_Demo {  **public** **static** **void** main(String[] args)      {            // Creating an empty HashMap          HashMap<Integer, String> hash\_map              = **new** HashMap<Integer, String>();            // Mapping string values to int keys          hash\_map.put(0, "Welcome");          hash\_map.put(1, "to");          hash\_map.put(2, "Geeks");          hash\_map.put(3, "4");          hash\_map.put(4, "Geeks");            // Displaying the HashMap          System.out.println("Initial Mappings are: "                             + hash\_map);            // Using values() to get the set view of values          System.out.println("The collection is: "                             + hash\_map.values());      }  } |

**Output**

Initial Mappings are: {0=Welcome, 1=to, 2=Geeks, 3=4, 4=Geeks}

The collection is: [Welcome, to, Geeks, 4, Geeks]

### 35. Write a program to join two ArrayList into one single ArrayList.

Given two ArrayLists in Java, our task is to join these ArrayLists.

Java

|  |
| --- |
| // Java program to demonstrate  // How to join ArrayList    **import** java.util.\*;    **public** **class** GFG {  **public** **static** **void** main(String args[])      {        ArrayList<String> list\_1 = **new** ArrayList<String>();          list\_1.add("Geeks");          list\_1.add("For");          list\_1.add("ForGeeks");            // Print the ArrayList 1          System.out.println("ArrayList 1: " + list\_1);            ArrayList<String> list\_2 = **new** ArrayList<String>();           list\_2.add("GeeksForGeeks");          list\_2.add("A computer portal");            // Displaying the ArrayList 2          System.out.println("ArrayList 2: " + list\_2);           // using Collection.addAll() method to join two          // arraylist          list\_1.addAll(list\_2);            // Print the joined ArrayList          System.out.println("Joined ArrayLists: " + list\_1);      }  } |

**Output**

ArrayList 1: [Geeks, For, ForGeeks]

ArrayList 2: [GeeksForGeeks, A computer portal]

Joined ArrayLists: [Geeks, For, ForGeeks, GeeksForGeeks, A computer portal]

For more information, refer to the article – [Join two ArrayLists in Java](https://www.geeksforgeeks.org/join-two-arraylists-in-java/#:~:text=Approach%3A%20ArrayLists%20can%20be%20joined,end%20of%20the%20first%20ArrayList.)

### 36. How can you synchronize an ArrayList in Java?

Using the Collections.synchronizedList() method, we can synchronize our collections in Java. SynchronizedList() returns a synchronized (thread-safe) list backed by a selection.

* Java

|  |
| --- |
| // Java program to show synchronization of ArrayList  **import** java.io.\*;  **import** java.util.\*;    **class** GFG {  **public** **static** **void** main(String[] args)      {          // Non Synchronized ArrayList          List<String> list = **new** ArrayList<String>();            list.add("Eat");          list.add("Coffee");          list.add("Code");          list.add("Sleep");          list.add("Repeat");            // Synchronizing ArrayList in Java          list = Collections.synchronizedList(list);            // we must use synchronize block to avoid          // non-deterministic behavior  **synchronized** (list)          {              Iterator<String> it = list.iterator();  **while** (it.hasNext()) {                  System.out.println(it.next());              }          }      }  } |

**Output**

Eat

Coffee

Code

Sleep

Repeat

### 37. What is a Properties Class in Java?

The properties class is a subclass of Hashtable. The properties class stores a list of values whose key is a string and whose value is also a string. Properties can define other properties class lists, but the default is properties.

**Features of Properties class:**

* Property is a subclass of Hashtable.
* The Properties file is used to store and retrieve string data type for a list of values where the key is a string and the value is also a string.
* If the original properties list does not contain a certain key property, the default properties list will be searched instead.
* Objects can be shared by multiple threads without external synchronization.
* The properties class can be used to retrieve the properties of the system.

### 38. ****What will happen if you use HashMap in a multithreaded Java application?****

In a multi-threaded environment, if multiple threads alter the map structurally, such as adding, removing, or modifying mappings, the internal data structure of HashMap may become corrupted and there may be some missing links, incorrect entries, and the map itself may become completely useless. Thus, you should not use HashMap in a concurrent application; instead, use ConcurrentHashMap or Hashtable which is thread-safe. The ConcurrentHashMap includes all the Hashtable’s methods as well as full concurrency of retrievals and updates.

**How did ThreadSafeConcurrentHashMap become thread-safe?**

* java.util.Concurrent.ConcurrentHashMap class provides thread safety by dividing the map into segments, which allows the lock to be taken only once per segment, i.e, once for each thread.
* The read operation in ConcurrentHashMap does not require a lock.

### 39. ****What will happen if two different keys of HashMap return the same hashcode()?****

When two different keys of HashMap return the same hash code, they will end up in the same bucket; therefore, collisions will occur. n case of collision, i.e. index of two or more nodes is the same, nodes are joined by a link list i.e. the second node is referenced by the first node and the third by the second, and so on.

### 40. What is WeakHashMap?

### WeakHashMap implements the Map interface. Unlike HashMap, WeakHashMap allows garbage collection even if the object specified as the key doesn’t contain any references despite being associated with WeakHashMap. In other words, Garbage Collector is better than WeakHashMap.

### 41. What is UnsupportedOperationException?

In the context of APIs or list implementations, the UnsupportedOperationException is a common exception. The exception is thrown when the requested operation cannot be performed. This class is a member of the Java Collections Framework.

**Syntax:**

public class UnsupportedOperationException

extends RuntimeException

### 42. How to make a Collection Read-Only in Java?

Creating a Read-Only Collection involves restricting the object to only fetching the data and not adding or removing data. Java has different methods for different Collection types like unmodifiableCollection(), unmodifiableMap(), ununmodifiableSet(), etc. java.util.The collections class defines all methods. The unmodifiableCollection() method creates a Read-Only collection. It requires a reference to the Collection class. If we have an object of Set Interface, we can use **ununmodifiableSet()** to make Read-Only.

### 43. Difference between PriorityQueue and TreeSet in Java?

| **PriorityQueue** | **TreeSet** |
| --- | --- |
| PriorityQueue comes in JDK 1.5. | TreeSet comes in JDK 1.4. |
| The data structure used by PriorityQueue is Queue | The data structure used by TreeSet is Set. |
| Duplicate elements are allowed. | Duplicate elements are not allowed. |
| Except for the root element, the rest of the elements do not follow any particular order in PriorityQueue. | In TreeSet all the elements remain in the sorted order. |
| Using PriorityQueue, we can retrieve the largest or smallest element in O(1) time. | TreeSet doesn’t provide a way to retrieve the largest or smallest element in O(1) time, but since they are in sorted order it gets the first or last element in O(1) time. |

*For more information, refer to the article –*[*Difference Between PriorityQueue and TreeSet*](https://www.geeksforgeeks.org/difference-between-priorityqueue-and-treeset/)

### 44. What is the diamond operator in Java?

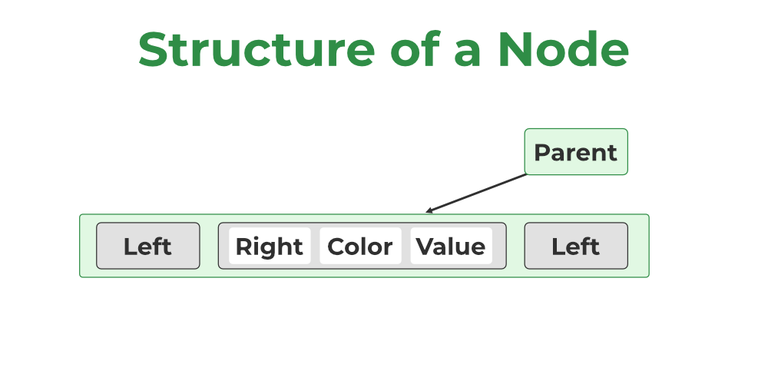
Diamond operators are used for simplifying the use of generics when creating objects while avoiding unchecked warnings in a program. When the Diamond operator was introduced in Java 7, we can create the object without mentioning the generic type on the right side of the expression as shown below.

**Syntax:**

List<String> list = new ArrayList<>();

### 45. How TreeMap works in Java?

[TreeMap](https://www.geeksforgeeks.org/treemap-in-java/)stores the key-value pairs, but TreeMap sorts the keys ascending rather than descending like HashMap. Depending on which constructor is used, TreeMap will be sorted either based on its keys, or by a Comparator. In TreeMap, the elements are sorted based on a Red-Black tree. A red-black tree is a self-balancing binary search tree where each node has an extra bit, and that bit is often interpreted as the color (red or black). These colors are used to ensure that the tree remains balanced during insertions and deletions.



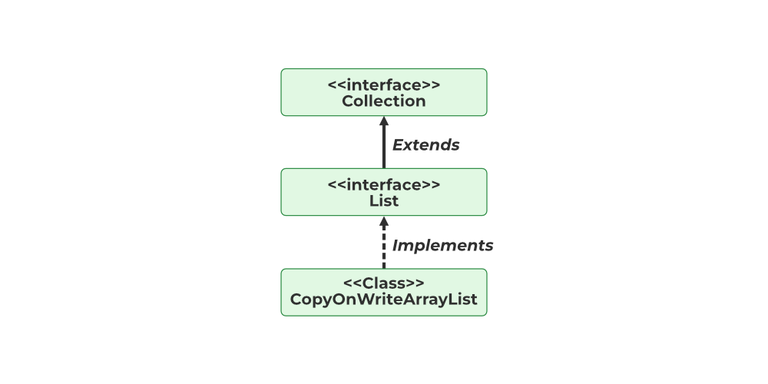
### 46. List down ways to iterate over Map in java?

The HashMap class provides Java’s Map interface by storing data in (Key, Value) pairs and accessing them by an index of another type. To use this class it is necessary to import java.util.HashMap package or its superclass.

There are numerous ways to iterate over HashMap of which 5 are listed below:

1. Iterate through a HashMap EntrySet using Iterators.
2. Iterate through HashMap KeySet using Iterator.
3. Iterate HashMap using for-each loop.
4. Iterating through a HashMap using Lambda Expressions.
5. Loop through a HashMap using Stream API.

### 47. What is CopyOnWriteArrayList in Java?



JDK 1.5 introduced an enhanced version of ArrayList called CopyOnWriteArrayList, where all modifications (add, set, remove, etc) are implemented by a new copy. It can be found in java.util.concurrent. It is a data structure created to be used in a concurrent environment. In a Thread-based environment, the CopyOnWriteArrayList is intended for frequent reading and infrequent updating. CopyOnWriteArrayList is a thread-safe version of  ArrayList.

### 48.  What is EnumMap in Java?

EnumMap is an implementation of the Map interface specific to enumeration types. EnumMap class is a member of the Java Collections Framework & is not synchronized. It extends AbstractMap and implements the Map interface in java. EnumMap belongs to java.util package.

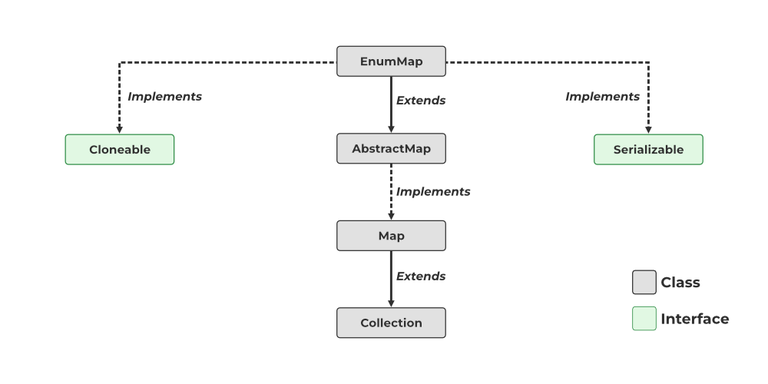
**Syntax:**

*public class EnumMap<K extends Enum<K>,​V> extends AbstractMap<K,​V> implements Serializable, Cloneable*

*// K must extend Enum, which enforces the requirement that the keys must be of the specified enum type.*

**Parameters:**

* Key object type
* Value object type



### 49. How does Hashmap internally Works?

HashMap works on the principle of Hashing. HashMap contains an array of Node and Node can represent a class having the following objects:

* int hash
* K key
* V value
* Node next

**The internal workings of HashMap:**

* Hashing
* Buckets
* Index Calculation in Hashmap

### 50. ****Why iterator in hashmap is considered fail-fast?****

fail-fast iterators immediately throw concurrent modification exceptions if any thread from outside attempts to modify the collection on which they are iterating. The fail-fast feature ensures that the iterator fails immediately if it detects that any modification of the collection will lead to anomalous behavior in the future.

[***Fail fast***](https://www.geeksforgeeks.org/fail-fast-fail-safe-iterators-java/)*feature ensures that if iterator feels that modification of collection would result in anomalous behaviour at any point of time in future, it fails immediately.*

**Example:**

* Java

|  |
| --- |
| **import** java.io.\*;  **import** java.util.ArrayList;  **import** java.util.Iterator;    **public** **class** GFG {  **public** **static** **void** main(String[] args)      {          ArrayList<Integer> arr = **new** ArrayList<>();          arr.add(1);          arr.add(2);          arr.add(3);          arr.add(4);          arr.add(5);            Iterator<Integer> it = arr.iterator();  **while** (it.hasNext()) {  **if** (it.next() == 2) {                  // will not throw Exception                  it.remove();              }          }            System.out.println(arr);            it = arr.iterator();  **while** (it.hasNext()) {  **if** (it.next() == 3) {                  // will throw Exception on                  // next call of next() method                  arr.remove(3);              }          }      }  } |

**Output:**

[1, 3, 4, 5]

Exception in thread "main" java.util.ConcurrentModificationException

at java.util.ArrayList$Itr.checkForComodification(ArrayList.java:901)

at java.util.ArrayList$Itr.next(ArrayList.java:851)

at FailFastExample.main(FailFastExample.java:28)

* **Constructor**

**What is a**[**Constructor?**](https://www.geeksforgeeks.org/constructors-in-java/)

 Constructors are used to initialize the object’s state. Like methods, a constructor also contains collection of statements(i.e. instructions) that are executed at time of Object creation.

**Do we have**[**Copy Constructor in Java**](https://www.geeksforgeeks.org/copy-constructor-in-java/)**?**

Like C++, Java also supports copy constructor. But, unlike C++, Java **doesn’t create a default copy constructor** if you don’t write your own. To copy the values of one object into another in java, you can use:

Constructor

Assigning the values of one object into another

[clone() method](https://www.geeksforgeeks.org/clone-method-in-java-2/) of Object class

**What is**[**Constructor Chaining**](https://www.geeksforgeeks.org/constructor-chaining-java-examples/)**?**

Constructor Chaining is a technique of calling another constructor from one constructor. this() is used to call same class constructor where as super() is used to call super class constructor.

|  |
| --- |
| **class** Temp  {      Temp()      {  **this**(5);          System.out.println("The Default constructor");      }         Temp(**int** x)      {  **this**(5, 15);          System.out.println(x);      }        Temp(**int** x, **int** y)      {          System.out.println(x \* y);      }  **public** **static** **void** main(String args[])      {  **new** Temp();      }  } |

**Can we call sub class constructor from super class constructor?**

 No. There is no way in java to call sub class constructor from a super class constructor.

**What happens if you keep a return type for a constructor?**

Ideally, Constructor must not have a return type. By definition, if a method has a return type, it’s not a constructor.([JLS8.8 Declaration](http://docs.oracle.com/javase/specs/#8.8)) It will be treated as a normal method. But compiler gives a warning saying that method has a constructor name.Example:

class GfG

{

int GfG()

{

return 0; // Warning for the return type

}

}

**What is No-arg constructor?**

Constructor without arguments is called no-arg constructor. Default constructor in java is always a no-arg constructor.

class GfG

{

public GfG()

{

//No-arg constructor

}

}

**How a no – argument constructor is different from**[**default Constructor**](https://www.geeksforgeeks.org/g-fact-50/)**?**

If a class contains no constructor declarations, then a default constructor with no formal parameters and no throws clause is implicitly declared. If the class being declared is the primordial class Object, then the default constructor has an empty body. Otherwise, the default constructor simply invokes the superclass constructor with no arguments.

**What are**[**private constructors**](https://www.geeksforgeeks.org/private-constructors-and-singleton-classes-in-java/)**and where are they used?**

 Like any method we can provide access specifier to the constructor. If it’s made private, then it can only be accessed inside the class. The major scenarios where we use private constructor:

Internal Constructor chaining

Singleton class design pattern

**When do we need**[**Constructor Overloading**](https://www.geeksforgeeks.org/constructor-overloading-java/)**?**

 Sometimes there is a need of initializing an object in different ways. This can be done using constructor overloading. Different constructors can do different work by implementing different line of codes and are called based on the type and no of parameters passed. According to the situation , a constructor is called with specific number of parameters among overloaded constructors.

**Do we have destructors in Java?** No, Because Java is a garbage collected language you cannot predict when (or even if) an object will be destroyed. Hence there is no direct equivalent of a destructor.

* **Inheritance**

### ****1. What does Java's inheritance mean?****

Inheritance is the term used in Java to describe the process of building a new class utilizing the features of an existing class. In other words, inheritance is how a child class acquires each parent class's traits.

### ****2. What are superclass and subclass, respectively?****

**Superclasses** are classes from which features are inherited by subclasses. It is also known as the parent class or foundation class.

**Subclasses** are classes that contain all the properties, methods, and nested classes of another class. A derived class, kid class, or extended class are other names.

### ****3. How does Java implement or achieve inheritance?****

Two keywords can be used to implement or accomplish inheritance:

* **extends:**The keyword extends is used to create an inheritance relationship between two classes and two interfaces.
* **implements:** The term "implements" establishes the line of descent between a class and an interface.

### ****4. Write the syntax for a class's subclass creation.****

The "extends" keyword can be used to build a subclass. The following syntax is used to declare a class subclass:

class subName extends superName

{

 // SubclassVariables

 // SubclassMethods

}

You can also try this code with **Online Java Compiler**

[**Run Code**](https://www.codingninjas.com/studio/online-compiler/online-java-compiler)

### ****5. Can a class extends on its own?****

Unable to extend itself.

### ****6. Are the instance initialization block and function Object() { [native code] } inherited by subclasses?****

No, the superclass's constructor and instance initialization block cannot be passed down to its subclass; nonetheless, they are used when creating an object of the subclass.

### ****7. In Java, are static members passed down to subclasses?****

Static blocks can't be passed down to their subclasses.

A static method from the superclass is inherited as a static member by the subclass, while non-static methods are only inherited as non-static members.

Must Read [Static Blocks In Java](https://www.codingninjas.com/studio/library/static-blocks-in-java).

### ****8. Can you override a final method?****

A final method cannot be overridden, unfortunately.

### ****9. How are constructors called in the case of inheritance?****

When there is inheritance, constructors are called in a hierarchical order.

### ****10. Why do Java programmers use inheritance??****

An inheritance is utilized to leverage polymorphism and reuse code by establishing a type hierarchy. For type declaration, inheritance is preferable, but because the composition is more versatile, it's a better choice for code reuse.

### ****11. What benefits does inheritance offer in Java?****

The following are some benefits of inheritance in Java:

* By placing the common code in the superclass and distributing it among various subclasses, we can reduce the amount of duplicate code in an application.
* The application's redundancy is decreased as a result of shorter code.
* Application code may become more adaptable through inheritance.

### ****12. What kinds of inheritance are there in Java?****

The different inheritance kinds are as follows:

a. Multi-level Inheritance

b. Single Inheritance

c. Multiple Inheritance

d. Hybrid Inheritance and,

e. Hierarchical Inheritance

Know about [Single Inheritance in Java](https://www.codingninjas.com/studio/library/single-inheritance-in-java) in detail.

### ****13. Why is inheritance necessary?****

One of the fundamental elements of the OOPs paradigm is inheritance. Some objects have similar traits and characteristics. A child class can inherit all of the traits and actions of the parent class.

The usage of inheritance in Java is justified for the reasons listed below.

* The basic class's code is reusable.
* By overriding, we can extend the functionality of a class or method via inheritance.
* The features of a class that already exist are utilized through inheritance.
* It is used to implement method overriding, often known as runtime polymorphism.

### ****14. Is the code going to compile successfully? In that case, what is the result?****

public class C1 {

  int x = 50;

}

public class C2 extends C1 {

  int x = 40;

}

public class Check {

public static void main(String[] args)

{

  C2 c2 = new C2();

  System.out.println(c2.x);

  C1 c1 = new C1();

  System.out.println(c1.x);

  C1 c3 = new C2();

  System.out.println(c3.x);

 }

}

You can also try this code with **Online Java Compiler**

[**Run Code**](https://www.codingninjas.com/studio/online-compiler/online-java-compiler)

Yes, the compilation of the code will succeed, and 40, 50, 50 is the output.

### ****15. Why isn't Java's class-based multiple inheritance support available?****

[Multiple Inheritance in Java](https://www.codingninjas.com/studio/library/multiple-inheritance-in-java) refers to when a class extends two base classes or superclasses, although a class cannot simultaneously extend more than one class in Java. One class may extend no more than one other class.

Because of this, Java does not permit multiple inheritance through classes, which reduces ambiguity, complexity, and confusion.

### ****16. What does Java's Super Keyword mean?****

A reference variable used to refer to the immediate parent class object in Java is the super keyword.

When you create an instance of a subclass, an implicit instance of the parent class is also produced and referred to by the super reference variable.

Three objectives serve the super keyword:

* The term "super" refers to an instant parent class variable.
* The instant parent class constructor is called using the super() function.
* The immediate parent class function is called using super.

### ****17. What does Java method overloading mean?****

The act of declaring two methods with the same name but different method signatures is known as overloading.

For example, System.out, an object of the PrintStream class, contains several println() methods that can be used to print various data types, including byte, short, int, char, float, and double.

They are all referred to as overloaded methods. In Java, overloaded method calls are resolved at compile time and require distinct method signatures.

### ****18. What are the Java guidelines for overloading a method?****

The only requirement for method overloading is that each overloaded method's method signature is unique.

For example, the method signature can be altered by altering the amount or type of method arguments. System.out.println() is overloaded to accept various primitive types, including int, short, byte, float, and others.

All of them accept the same reasoning, but they all have different types.

Changing the order of the method arguments is another way to change the method signature; however, doing so frequently results in unclear code and should be avoided.

### ****19. What does Java's method overriding mean?****

Another option to define a method with the same name but a different code is to override it; this method must be in a subclass.

The basis for overriding is run-time. Method calls with polymorphism are resolved at runtime based on the actual object.

For example, if a variable of type Parent contains an object of the Child class, the method called will, unless it is overridden, be from the Child class and not the Parent class.

You must go by the rules of method overriding, which include declaring a method with the same signature in a subclass to override a method.

### ****20. What does Java's method hiding mean?****

Because Java's static methods' method calls are resolved at compile time, they cannot be modified. However, this did not stop you from declaring a method with the same name in a subclass.

In this instance, we say that a static method in the parent class was hidden by a method in the subclass.

Because overloading is handled at compile time, if a variable in the parent class points to an object in the child class, the parent class's static method will also be called.

Let us now discuss the hard type of java inheritance interview questions.

### ****21. Is it possible to block method overriding without the final modifier?****

In Java, there are indeed some odd ways to prevent method overriding. Although the final modifier is only for that purpose, method overriding can be avoided using the private keyword.

How? If you recall, an extensible class is required to override a method. Because the parent class's constructor won't be available to subclasses if you make it private, that class cannot be extended.

This cannot be overridden because it is automatically called by the subclass's constructor.

This method is applied in the Singleton design pattern, where the constructor is purposely made secret, and an accessible singleton instance is made available via a static getInstance() method.

### ****22. Does Java permit multiple inheritance or can a class extend multiple classes? Why not, then?****

Java does not enable multiple inheritance, nor can a class extend more than one class. Java does not enable multiple inheritance since it can lead to ambiguity, complexity, and confusion. For example, Class C will have two methods with the same name if it extends Classes A and B, each of which has a method with the same name. This makes it unclear and difficult to decide which approach to take. Java does not permit multiple inheritance in order to prevent this.

class C1

{

   void funOne()

   {

       System.out.println("From methodOfClassC1");

   }

}

class C2

{

   void funOne()

   {

       System.out.println("From methodOfClassC2");

   }

}

class C3 extends C1, C2 ()

{

   //Class C3 will inherit two identical methods.

   //This leads to uncertainty and misunderstanding.

}

You can also try this code with **Online Java Compiler**

[**Run Code**](https://www.codingninjas.com/studio/online-compiler/online-java-compiler)

### ****23. List the Java access specifiers that are available?****

Here is a list of the access specifiers that Java supports:

* **private:** A private modifier is only accessible to members of the class. Outside of class, it can't be accessed.
* **default:**A default modifier is only available within the package. Outside of the package, it cannot be accessed. If no access level is specified, the default will apply.
* **protected:** A protected modifier can be accessed inside and outside the package via a child class. Only a child class can access it; nevertheless, it cannot be accessed outside the package.
* **public:** A public modifier is available everywhere. It may be accessible both inside and outside of the class and inside and outside the package.

### ****24. What does "co-variant Method Overriding" means?****

One of the rules of method overriding is that the return type of the overriding method must be the same as the overridden method; however, starting with Java 1.5, this requirement has been slightly eased and the overridden method can now return a subclass of the return type of the original method.

Casting at the client end can be eliminated thanks to a relaxation technique called co-variant method overriding.

The clone() method overriding is one of the best examples. The Object.clone() method returns an Object that needs to be cast, but by overriding the co-variant method, you can return the appropriate type immediately.

**Example:** The Date class returns an object of type java.util.Date rather than java.lang.Object.

### ****25. What distinguishes composition from inheritance?****

In Java, composition and inheritance differ in a number of ways, including the following:

* While inheritance cannot be modified, i.e. you cannot ask a class to implement another class at runtime, the composition is more flexible since you may change the implementation at runtime by executing the setXXX() function.
* A Room HAS A Fan, yet a Mango IS-A Fruit is an example of how composition builds HAS-A relationship while inheritance builds IS-A relationship.
* Inheritance represents the parent-child relationship the best. However, the composition can also be used if you just need one class.

### ****26. What distinguishes inheritance from encapsulation?****

A parent-child relationship is created via the object-oriented idea of inheritance. Although it serves as the foundation for polymorphism, it is one approach to reuse code created for parent classes.

On the other hand, encapsulation is an object-oriented notion used to conceal a class's internal details, such as How to store elements and generate hash values are both covered by HashMap.

### ****27. What distinguishes overriding from overloading methods?****

The primary distinction between overloading and overriding is that the former occurred during compile time while the latter occurred during run time. This is the cause.

Only virtual methods in Java can be overloaded. Methods resolved at compile time, such as private, static, and final methods in Java, cannot be overridden.

Additionally, the conditions for overloading and overriding methods differ. For instance, a method must have a different method signature to be considered overloading, whereas it must have the same method signature to be considered overriding.

### ****28. What are the super() and this() methods in Java?****

**super()**

* To call the constructor in the superclass, use the super keyword.
* Super always refers to the current class's parent.
* You can use Super to access the parent class's public and protected methods and properties. The parent's private methods and properties are hidden from you.
* Only the constructors of the class can access constructors using super.

**this()**

* A reference to the current class is made by this.
* It enables access to the current class's methods and attributes, including private ones.
* It is used to access the current object's methods and fields. Because of this, it, for instance, has no significance in static methods. this keyword was previously used to call the class's constructor (other overloaded constructor)

### ****29. Three classes, C1, C2, and C3, have been established in the code below. Both Class C2 and Class C1 are extended by Class C3.****

**Is there a way for Class C3 to invoke the method f1() of Class C1, which exists in each class?**

public class C1

{

 void f1(){

   System.out.println("f1 in class C1");

}

}

public class C2 extends C1

{

 void f1() {

    System.out.println("f1 in class C2");

}

}

public class C3 extends C2

{

 void f1() {

System.out.println("f1 in class C3");

  }

}

public class Test {

public static void main(String[] args)

{

  C3 c3 = new C3();

   c3.f1();

 }

}

[**Run Code**](https://www.codingninjas.com/studio/online-compiler/online-java-compiler)

Every class in the code above has a method called f1() with the same signature; as a result, Class C2 overrides Class C1's f1() method, and Class C3 overrides Class C2's f1() method.

Using the super.f1() call classes C2 and C3 can now call their superclass's f1() function.

However, in this case, invoking C1's f1() method from Class C3 is not possible because it violates Java's OOPs notion and does not employ super.super.

Since multiple inheritance is not permitted in Java, C3 can only see one superclass, which will have just one f1() method implementation. The f1() method of C1 is hidden from C3.

This situation is sometimes referred to as the Diamond Problem of Multiple Inheritance. Class C2 and Class C3 must both call the super.f1() method for class C3 to call class C1's f1() implementation.

### ****30. What distinguishes aggregation and composition as two distinct concepts?****

The terms "aggregation" and "composition" refer to different association links in the OOPs paradigm. Between classes, a composition forges a solid connection.

All components of a composite item are destroyed if it is destroyed. A car, as an illustration, has a steering wheel. The steering wheel's presence is meaningless if the car is destroyed.

A weaker association between classes is established via aggregation than by composition. As an illustration, libraries have students. Students still exist even if a library is destroyed. As a result, the library and the student are related.

**🡪**

* **JDBC**

# **JDBC Interview Questions**

A list of top frequently asked JDBC interview questions and answers is given below.

### 1) What is JDBC?

JDBC is a Java API that is used to connect and execute the query to the database. JDBC API uses JDBC drivers to connect to the database. JDBC API can be used to access tabular data stored into any relational database.

 [More details.](https://www.javatpoint.com/jdbc-tutorial)

### 2) 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:** The JDBC-ODBC bridge driver uses the ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of the thin driver. It is easy to use and can be easily connected to any database.
2. **Native-API driver (partially java driver):** The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in Java. Its performance is better than JDBC-ODBC bridge driver. However, the native driver must be installed on each client machine.
3. **Network Protocol driver (fully java driver):** The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is entirely written in Java. There is no requirement of the client-side library because of the application server that can perform many tasks like auditing, load balancing, logging, etc.
4. **Thin driver (fully java driver):** The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as the thin driver. It is entirely written in Java language. Its performance is better than all other drivers however these drivers depend upon the database.

[More details.](https://www.javatpoint.com/jdbc-driver)

### 3) What are the steps to connect to the database in java?

The following steps are used in database connectivity.

* **Registering the driver class:**

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

* 1. Class.forName("oracle.jdbc.driver.OracleDriver");
* **Creating connection:**

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

* 1. 1) **public** **static** Connection getConnection(String url)**throws** SQLException
  2. 2) **public** **static** Connection getConnection(String url,String name,String password)
  3. **throws** SQLException

Consider the following example to establish the connection with the Oracle database.

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

The createStatement() method of Connection interface is used to create the Statement. The object of the Statement is responsible for executing queries with the database.

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

consider the following example to create the statement object

* 1. Statement stmt=con.createStatement();
* **Executing the queries:**

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

Syntax of executeQuery() method is given below.

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

Example to execute the query

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

However, to perform the insert and update operations in the database, executeUpdate() method is used which returns the boolean value to indicate the successful completion of the operation.

* **Closing connection:**

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

Syntax of close() method is given below.

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

Consider the following example to close the connection.

* 1. con.close();

[More details.](https://www.javatpoint.com/steps-to-connect-to-the-database-in-java)

### 4) What are the JDBC API components?

The java.sql package contains following interfaces and classes for JDBC API.

**Interfaces:**

* **Connection:** The Connection object is created by using getConnection() method of DriverManager class. DriverManager is the factory for connection.
* **Statement:** The Statement object is created by using createStatement() method of Connection class. The Connection interface is the factory for Statement.
* **PreparedStatement:** The PrepareStatement object is created by using prepareStatement() method of Connection class. It is used to execute the parameterized query.
* **ResultSet:** The object of ResultSet maintains a cursor pointing to a row of a table. Initially, cursor points before the first row. The executeQuery() method of Statement interface returns the ResultSet object.
* **ResultSetMetaData:** The object of ResultSetMetaData interface cotains the information about the data (table) such as numer of columns, column name, column type, etc. The getMetaData() method of ResultSet returns the object of ResultSetMetaData.
* **DatabaseMetaData:** DatabaseMetaData interface provides methods to get metadata of a database such as the database product name, database product version, driver name, name of the total number of tables, the name of the total number of views, etc. The getMetaData() method of Connection interface returns the object of DatabaseMetaData.
* **CallableStatement:** CallableStatement interface is used to call the stored procedures and functions. We can have business logic on the database through the use of stored procedures and functions that will make the performance better because these are precompiled. The prepareCall() method of Connection interface returns the instance of CallableStatement.

**Classes:**

* **DriverManager:** The DriverManager class acts as an interface between the user and drivers. It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver. It contains several methods to keep the interaction between the user and drivers.
* **Blob:** Blob stands for the binary large object. It represents a collection of binary data stored as a single entity in the database management system.
* **Clob:** Clob stands for Character large object. It is a data type that is used by various database management systems to store character files. It is similar to Blob except for the difference that BLOB represent binary data such as images, audio and video files, etc. whereas Clob represents character stream data such as character files, etc.
* **SQLException** It is an Exception class which provides information on database access errors.

### 5) What are the JDBC statements?

In JDBC, Statements are used to send SQL commands to the database and receive data from the database. There are various methods provided by JDBC statements such as execute(), executeUpdate(), executeQuery, etc. which helps you to interact with the database.

There is three type of JDBC statements given in the following table.

|  |  |
| --- | --- |
| **Statements** | **Explanation** |
| Statement | Statement is the factory for resultset. It is used for general purpose access to the database. It executes a static SQL query at runtime. |
| PreparedStatement | The PreparedStatement is used when we need to provide input parameters to the query at runtime. |
| CallableStatement | CallableStatement is used when we need to access the database stored procedures. It can also accept runtime parameters. |

### 6) What is the return type of Class.forName() method?

The Class.forName() method returns the object of java.lang.Class object.

### 7) What are the differences between Statement and PreparedStatement interface?

|  |  |
| --- | --- |
| **Statement** | **PreparedStatement** |
| The Statement interface provides methods to execute queries with the database. The statement interface is a factory of ResultSet; i.e., it provides the factory method to get the object of ResultSet. | The PreparedStatement interface is a subinterface of Statement. It is used to execute the parameterized query. |
| In the case of Statement, the query is compiled each time we run the program. | In the case of PreparedStatement, the query is compiled only once. |
| The Statement is mainly used in the case when we need to run the static query at runtime. | PreparedStatement is used when we need to provide input parameters to the query at runtime. |

[More details.](https://www.javatpoint.com/PreparedStatement-interface)

### 8) How can we set null value in JDBC PreparedStatement?

By using setNull() method of PreparedStatement interface, we can set the null value to an index. The syntax of the method is given below.

1. **void** setNull(**int** parameterIndex, **int** sqlType) **throws** SQLException

### 9) What are the benefits of PreparedStatement over Statement?

The benefits of using PreparedStatement over Statement interface is given below.

* The PreparedStatement performs faster as compare to Statement because the Statement needs to be compiled everytime we run the code whereas the PreparedStatement compiled once and then execute only on runtime.
* PreparedStatement can execute Parameterized query whereas Statement can only run static queries.
* The query used in PreparedStatement is appeared to be similar every time. Therefore, the database can reuse the previous access plan whereas, Statement inline the parameters into the String, therefore, the query doesn't appear to be same everytime which prevents cache reusage.

### 10) What are the differences between execute, executeQuery, and executeUpdate?

|  |  |  |
| --- | --- | --- |
| **execute** | **executeQuery** | **executeUpdate** |
| The execute method can be used for any SQL statements(Select and Update both). | The executeQuery method can be used only with the select statement. | The executeUpdate method can be used to update/delete/insert operations in the database. |
| The execute method returns a boolean type value where true indicates that the ResultSet s returned which can later be extracted and false indicates that the integer or void value is returned. | The executeQuery() method returns a ResultSet object which contains the data retrieved by the select statement. | The executeUpdate() method returns an integer value representing the number of records affected where 0 indicates that query returns nothing. |

### 11) What are the different types of ResultSet?

ResultSet is categorized by the direction of the reading head and sensitivity or insensitivity of the result provided by it. There are three general types of ResultSet.

|  |  |
| --- | --- |
| Type | Description |
| ResultSet.TYPE\_Forward\_ONLY | The cursor can move in the forward direction only. |
| ResultSet.TYPE\_SCROLL\_INSENSITIVE | The cursor can move in both the direction (forward and backward). The ResultSet is not sensitive to the changes made by the others to the database. |
| ResultSet.TYPE\_SCROLL\_SENSITIVE | The cursor can move in both the direction. The ResultSet is sensitive to the changes made by the others to the database. |

### 12) What are the differences between ResultSet and RowSet?

|  |  |
| --- | --- |
| ResultSet | RowSet |
| ResultSet cannot be serialized as it maintains the connection with the database. | RowSet is disconnected from the database and can be serialized. |
| ResultSet object is not a JavaBean object | ResultSet Object is a JavaBean object. |
| ResultSet is returned by the executeQuery() method of Statement Interface. | Rowset Interface extends ResultSet Interface and returned by calling the RowSetProvider.newFactory().createJdbcRowSet() method. |
| ResultSet object is non-scrollable and non-updatable by default. | RowSet object is scrollable and updatable by default. |

### 13) How can we execute stored procedures using CallableStatement?

Following are the steps to create and execute stored procedures. Here, we are creating a table user420 by using a stored procedure and inserting values into it.

* **Create the procedure in the database.**

To call the stored procedure, you need to create it in the database. Here, we are assuming that the stored procedure looks like this.

* 1. create or replace procedure "INSERTR"
  2. (id IN NUMBER,
  3. name IN VARCHAR2)
  4. is
  5. begin
  6. insert into user420 values(id,name);
  7. end;
  8. /

The table structure is given below:

* 1. create table user420(id number(10), name varchar2(200));
* **Establish a network connection.**
  1. Class.forName("oracle.jdbc.driver.OracleDriver");
  2. Connection con=DriverManager.getConnection(
  3. "jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
* **Create the Object of CallableStatement.**
  1. CallableStatement stmt=con.prepareCall("{call insertR(?,?)}");
* **Provide the values and execute the query by using the following syntax.**
  1. stmt.setInt(1,1011);
  2. stmt.setString(2,"Amit");
  3. stmt.execute();
* **Check the database; the values will be found there. However, the complete code will look like the following.**
  1. **import** java.sql.\*;
  2. **public** **class** Proc {
  3. **public** **static** **void** main(String[] args) **throws** Exception{
  5. Class.forName("oracle.jdbc.driver.OracleDriver");
  6. Connection con=DriverManager.getConnection(
  7. "jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
  9. CallableStatement stmt=con.prepareCall("{call insertR(?,?)}");
  10. stmt.setInt(1,1011);
  11. stmt.setString(2,"Amit");
  12. stmt.execute();
  14. System.out.println("success");
  15. }
  16. }

### 14) What is the role of the JDBC DriverManager class?

The DriverManager class acts as an interface between user and drivers. It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver. The DriverManager class maintains a list of Driver classes that have registered themselves by calling the method DriverManager.registerDriver().

[More details.](https://www.javatpoint.com/DriverManager-class)

### 15) What are the functions of the JDBC Connection interface?

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

[More details.](https://www.javatpoint.com/Connection-interface)

### 16) What does the 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. By default, ResultSet object can move in the forward direction only and is not updatable. However, we can make this object to move the forward and backward direction by passing either TYPE\_SCROLL\_INSENSITIVE or TYPE\_SCROLL\_SENSITIVE in createStatement(int, int) method.

[More details.](https://www.javatpoint.com/ResultSet-interface)

### 17) What does the JDBC ResultSetMetaData interface?

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

[More details.](https://www.javatpoint.com/ResultSetMetaData-interface)

### 18) What does the 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. Consider the following example.

1. **import** java.sql.\*;
2. **class** Dbmd{
3. **public** **static** **void** main(String args[]){
4. **try**{
5. Class.forName("oracle.jdbc.driver.OracleDriver");
7. Connection con=DriverManager.getConnection(
8. "jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
9. DatabaseMetaData dbmd=con.getMetaData();
11. System.out.println("Driver Name: "+dbmd.getDriverName());
12. System.out.println("Driver Version: "+dbmd.getDriverVersion());
13. System.out.println("UserName: "+dbmd.getUserName());
14. System.out.println("Database Product Name: "+dbmd.getDatabaseProductName());
15. System.out.println("Database Product Version: "+dbmd.getDatabaseProductVersion());
17. con.close();
18. }**catch**(Exception e){ System.out.println(e);}
19. }
20. }

**Output**

Driver Name: Oracle JDBC Driver

Driver Version: 10.2.0.1.0XE

Database Product Name: Oracle

Database Product Version: Oracle Database 10g Express Edition Release 10.2.0.1.0 -Production

[More details.](https://www.javatpoint.com/DatabaseMetaData-interface)

### 19) Which interface is responsible for transaction management in JDBC?

The **Connection interface** provides methods for transaction management such as commit(), rollback() etc.

[More details.](https://www.javatpoint.com/transaction-management-in-jdbc)

### 20) What is batch processing and how to perform batch processing in JDBC?

By using the batch processing technique in JDBC, we can execute multiple queries. It makes the performance fast. The java.sql.Statement and java.sql.PreparedStatement interfaces provide methods for batch processing. The batch processing in JDBC requires the following steps.

* Load the driver class
* Create Connection
* Create Statement
* Add query in the batch
* Execute the Batch
* Close Connection

ADVERTISEMENT

ADVERTISEMENT

Consider the following example to perform batch processing using the Statement interface.

1. **import** java.sql.\*;
2. **class** FetchRecords{
3. **public** **static** **void** main(String args[])**throws** Exception{
4. Class.forName("oracle.jdbc.driver.OracleDriver");
5. Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
6. con.setAutoCommit(**false**);
8. Statement stmt=con.createStatement();
9. stmt.addBatch("insert into user420 values(190,'abhi',40000)");
10. stmt.addBatch("insert into user420 values(191,'umesh',50000)");
12. stmt.executeBatch();//executing the batch
14. con.commit();
15. con.close();
16. }}

[More details.](https://www.javatpoint.com/batch-processing-in-jdbc)

### 21) What are CLOB and BLOB data types in JDBC?

**BLOB:** Blob can be defined as the variable-length, binary large object which is used to hold the group of Binary data such as voice, images, and mixed media. It can hold up to 2GB data on MySQL database and 128 GB on Oracle database. BLOB is supported by many databases such as MySQL, Oracle, and DB2 to store the binary data (images, video, audio, and mixed media).

**CLOB:** Clob can be defined as the variable-length, character-large object which is used to hold the character-based data such as files in many databases. It can hold up to 2 GB on MySQL database, and 128 GB on Oracle Database. A CLOB is considered as a character string.

### 22) What are the different types of lockings in JDBC?

A lock is a certain type of software mechanism by using which, we can restrict other users from using the data resource. There are four type of locks given in JDBC that are described below.

* **Row and Key Locks:** These type of locks are used when we update the rows.
* **Page Locks:** These type of locks are applied to a page. They are used in the case, where a transaction remains in the process and is being updated, deleting, or inserting some data in a row of the table. The database server locks the entire page that contains the row. The page lock can be applied once by the database server.
* **Table locks:** Table locks are applied to the table. It can be applied in two ways, i.e., shared and exclusive. Shared lock lets the other transactions to read the table but not update it. However, The exclusive lock prevents others from reading and writing the table.
* **Database locks:** The Database lock is used to prevent the read and update access from other transactions when the database is open.

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

By using the PreparedStatement interface, we can store and retrieve images. Create a table which contains two columns namely NAME and PHOTO.

1. CREATE TABLE  "IMGTABLE"
2. (    "NAME" VARCHAR2(4000),
3. "PHOTO" BLOB
4. )

Consider the following example to store the image in the database.

1. **import** java.sql.\*;
2. **import** java.io.\*;
3. **public** **class** InsertImage {
4. **public** **static** **void** main(String[] args) {
5. **try**{
6. Class.forName("oracle.jdbc.driver.OracleDriver");
7. Connection con=DriverManager.getConnection(
8. "jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
10. PreparedStatement ps=con.prepareStatement("insert into imgtable values(?,?)");
11. ps.setString(1,"sonoo");
13. FileInputStream fin=**new** FileInputStream("d:\\g.jpg");
14. ps.setBinaryStream(2,fin,fin.available());
15. **int** i=ps.executeUpdate();
16. System.out.println(i+" records affected");
18. con.close();
19. }**catch** (Exception e) {e.printStackTrace();}
20. }
21. }

Consider the following example to retrieve the image from the table.

1. **import** java.sql.\*;
2. **import** java.io.\*;
3. **public** **class** RetrieveImage {
4. **public** **static** **void** main(String[] args) {
5. **try**{
6. Class.forName("oracle.jdbc.driver.OracleDriver");
7. Connection con=DriverManager.getConnection(
8. "jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
10. PreparedStatement ps=con.prepareStatement("select \* from imgtable");
11. ResultSet rs=ps.executeQuery();
12. **if**(rs.next()){//now on 1st row
14. Blob b=rs.getBlob(2);//2 means 2nd column data
15. **byte** barr[]=b.getBytes(1,(**int**)b.length());//1 means first image
17. FileOutputStream fout=**new** FileOutputStream("d:\\sonoo.jpg");
18. fout.write(barr);
20. fout.close();
21. }//end of if
22. System.out.println("ok");
24. con.close();
25. }**catch** (Exception e) {e.printStackTrace();  }
26. }
27. }

[More details.](https://www.javatpoint.com/storing-image-in-oracle-database)

### 24) How can we store the file in the Oracle database?

The setCharacterStream() method of PreparedStatement interface is used to set character information into the parameterIndex. For storing the file into the database, CLOB (Character Large Object) datatype is used in the table. For example:

1. CREATE TABLE  "FILETABLE"
2. (    "ID" NUMBER,
3. "NAME" CLOB
4. )

**Java Code**

1. **import** java.io.\*;
2. **import** java.sql.\*;
4. **public** **class** StoreFile {
5. **public** **static** **void** main(String[] args) {
6. **try**{
7. Class.forName("oracle.jdbc.driver.OracleDriver");
8. Connection con=DriverManager.getConnection(
9. "jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
11. PreparedStatement ps=con.prepareStatement(
12. "insert into filetable values(?,?)");
14. File f=**new** File("d:\\myfile.txt");
15. FileReader fr=**new** FileReader(f);
17. ps.setInt(1,101);
18. ps.setCharacterStream(2,fr,(**int**)f.length());
19. **int** i=ps.executeUpdate();
20. System.out.println(i+" records affected");
22. con.close();
24. }**catch** (Exception e) {e.printStackTrace();}
25. }
26. }

### 25) How can we retrieve the file in the Oracle database?

The getClob() method of PreparedStatement is used to get file information from the database. Let's see the table structure of the example to retrieve the file.

1. CREATE TABLE  "FILETABLE"
2. (    "ID" NUMBER,
3. "NAME" CLOB
4. )

The example to retrieve the file from the Oracle database is given below.

1. **import** java.io.\*;
2. **import** java.sql.\*;
4. **public** **class** RetrieveFile {
5. **public** **static** **void** main(String[] args) {
6. **try**{
7. Class.forName("oracle.jdbc.driver.OracleDriver");
8. Connection con=DriverManager.getConnection(
9. "jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
11. PreparedStatement ps=con.prepareStatement("select \* from filetable");
12. ResultSet rs=ps.executeQuery();
13. rs.next();//now on 1st row
15. Clob c=rs.getClob(2);
16. Reader r=c.getCharacterStream();
18. FileWriter fw=**new** FileWriter("d:\\retrivefile.txt");
20. **int** i;
21. **while**((i=r.read())!=-1)
22. fw.write((**char**)i);
24. fw.close();
25. con.close();
27. System.out.println("success");
28. }**catch** (Exception e) {e.printStackTrace();  }
29. }
30. }

### 26) What are the differences between stored procedure and functions?

The differences between stored procedures and functions are given below:

|  |  |
| --- | --- |
| **Stored Procedure** | **Function** |
| Is used to perform business logic. | Is used to perform the calculation. |
| Must not have the return type. | Must have the return type. |
| May return 0 or more values. | May return only one value. |
| The procedure supports input and output parameters. | The function supports only input parameter. |
| Exception handling using try/catch block can be used in stored procedures. | Exception handling using try/catch can't be used in user-defined functions. |

### 27) How can we maintain the integrity of a database by using JDBC?

To maintain the integrity of a database, we need to ensure the ACID properties. ACID properties mean Atomicity, Consistency, Isolation, and durability. In JDBC, Connection interface provides methods like setAutoCommit(), commit(), and rollback() which can be used to manage transaction. Let's see an example of transaction management in JDBC.

1. **import** java.sql.\*;
2. **class** FetchRecords{
3. **public** **static** **void** main(String args[])**throws** Exception{
4. Class.forName("oracle.jdbc.driver.OracleDriver");
5. Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
6. con.setAutoCommit(**false**);
8. Statement stmt=con.createStatement();
9. stmt.executeUpdate("insert into user420 values(190,'abhi',40000)");
10. stmt.executeUpdate("insert into user420 values(191,'umesh',50000)");
12. con.commit();
13. con.close();
14. }}

### 28) What is the JDBC Rowset?

JDBC Rowset is the wrapper of ResultSet. It holds tabular data like ResultSet, but it is easy and flexible to use. The implementation classes of RowSet interface are as follows:

* JdbcRowSet
* CachedRowSet
* WebRowSet
* JoinRowSet
* FilteredRowSet

### 29) What is the major difference between java.util.Date and java.sql.Date data type?

The major difference between java.util.Date and java.sql.Date is that, java.sql.Date represents date without time information whereas, java.util.Date represents both date and time information.

### 30) What does JDBC setMaxRows method do?

The setMaxRows(int i) method limits the number of rows the database can return by using the query. This can also be done within the query as we can use the limit cause in MySQL.

* **Interface and Abstraction**

### 1. Can abstract classes have constructors in Java?

Yes, an abstract class can declare and define a constructor in Java. Since you can not create an instance of an abstract class, a constructor can only be called during [constructor chaining](http://javarevisited.blogspot.com/2012/12/constructor-chaining-in-java-calling-another-constructor.html), i.e. when you create an instance of the concrete implementation class.

Now some interviewer, ask what is the purpose of a constructor if you can not instantiate abstract class? Well, it can still be used to initialize common variables, which are declared inside an abstract class, and used by the various implementation.   
  
Also even if you don’t provide any constructor, the compiler will add a [default no-argument constructor](http://javarevisited.blogspot.com/2012/12/what-is-constructor-in-java-example-chainning-overloading.html) in an abstract class, without that your subclass will not compile, since the first statement in any constructor implicitly calls super(), default superclass constructor in Java.

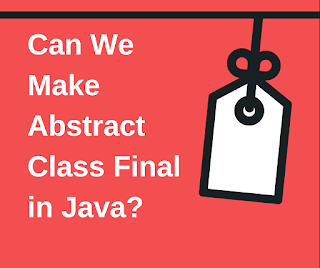
### 2. Can abstract class implement interface in Java? do they require to implement all methods?

Yes, an abstract class can implement an interface by using the implements keyword. Since they are abstract, they don’t need to implement all methods. It’s good practice to provide an abstract base class, along with an interface to declare Type. One example of this is java.util.List interface and corresponding java.util.AbstractList abstract class.   
  
Since AbstractList implements all common methods,  concrete implementations like [LinkedList](http://javarevisited.blogspot.com/2012/02/difference-between-linkedlist-vs.html) and [ArrayList](http://javarevisited.blogspot.com/2012/03/how-to-loop-arraylist-in-java-code.html) are free from the burden of implementing all methods, had they implemented List interface directly.   
  
It’s the best of both worlds, you can get the advantage of interface for declaring type, and flexibility of abstract class to implement common behavior in one place. Effective Java has a nice chapter on how to use interface and abstract class in Java, which is worth reading.

### 3. Can an abstract class be final in Java?

No, an abstract class can not be final in Java. Making them final will stop the abstract class from being extended, which is the only way to use an abstract class. They are also opposite of each other, abstract keyword enforces to extend a class, for using it, on the other hand, [final keyword](http://javarevisited.blogspot.com/2011/12/final-variable-method-class-java.html) prevents a class from being extended.

In real-world also, abstract signifies incompleteness, while final is used to demonstrate completeness. The bottom line is, you can not make your class abstract and final in Java, at the same time, it’s a compile-time error.

[](https://1.bp.blogspot.com/-MgOITF81UKA/YM12aY0I5CI/AAAAAAAAoQY/ifpENUDBCWsFVwiQjaayhIoSNUbbRzH9wCLcBGAsYHQ/s1287/Can%2Bwe%2Bmake%2Babstract%2Bclass%2Bfinal%2Bin%2BJava.png)

### 4. Can abstract classes have static methods in Java?

Yes, an abstract class can declare and define [static methods](http://javarevisited.blogspot.com/2011/11/static-keyword-method-variable-java.html), nothing prevents doing that. But, you must follow guidelines for making a method static in Java, as it’s not welcomed in an object-oriented design, because [static methods can not be overridden in Java](http://javarevisited.blogspot.com/2013/03/can-we-overload-and-override-static-method-java.html). It’s very rare, you see static methods inside an abstract class, but as I said, if you have a very good reason for doing it, then nothing stops you.

### 5. Can you create an instance of an abstract class?

No, you can not create instances of abstract class in Java, they are incomplete. Even though, if your abstract class doesn’t contain any abstract method, you can not create an instance of it. By making a class abstract,  you told the compiler that, it’s incomplete and should not be instantiated. Java compiler will throw an error when a code tries to instantiate an abstract class.

### 6. Is it necessary for an abstract class to have an abstract method?

[Java abstract class and interface interview Question answers](https://3.bp.blogspot.com/-K6q0DQ1v-tw/TWu8owBtc2I/AAAAAAAAADA/oBoHDBiJ8ag/s1600/17.jpg)No, It’s not mandatory for an abstract class to have any abstract method. You can make a class abstract in Java, by just using the abstract keyword in the class declaration. The compiler will enforce all structural restrictions, applied to an abstract class, like,  not allowing to create of any instance.

By the way, it’s debatable whether you should have an abstract method inside the abstract class or interface. In my opinion, the abstract class should have abstract methods, because that’s the first thing programmer assumes when he sees that class. That would also go nicely along with the principle of least surprise.

### 7. Difference between abstract class and interface in Java?

This is the most important and one of the classic Java Interview questions. I don’t know, how many times I have seen this question at all most all levels of Java interviews. One reason, which makes this question interesting is the ability to produce examples.

It’s easy to answers questions on core OOP concepts like [Abstraction](http://javarevisited.blogspot.com/2010/10/abstraction-in-java.html), [Encapsulation](http://javarevisited.blogspot.com/2012/03/what-is-encapsulation-in-java-and-oops.html), [Polymorphism](http://javarevisited.blogspot.com/2011/08/what-is-polymorphism-in-java-example.html), and [Inheritance](http://javarevisited.blogspot.com/2012/10/what-is-inheritance-in-java-and-oops-programming.html), but when it comes to subtle points like this, a candidate more often fumbled. You can see this post for all syntactical differences between abstract class and interface, but it deserves a post on its own.

### 8. When do you favor abstract class over interface?

This is the follow-up of previous interview questions on abstract class and interface. If you know the syntactical difference, you can answer this question quite easily, as they are the one, which drives the decision. Since it’s almost impossible to add a new method on a published interface, it’s better to use abstract class, when evolution is a concern.

Abstract class in Java evolves better than the interface. Similarly, if you have too many methods inside the interface, you are creating pain for all its implementation, consider providing an abstract class for default implementation. This is the pattern followed in the Java collection package, you can see AbstractList provides a default implementation for the List interface.

### 9. What is the abstract method in Java?

An abstract method is a method without a body. You just declare the method, without defining it and use abstract keywords in the method declaration.  All methods declared inside [Java Interface](http://javarevisited.blogspot.com/2012/04/10-points-on-interface-in-java-with.html) is by default abstract. Here is an example of an abstract method in Java

                public void abstract printVersion();

Now, In order to implement this method, you need to extend the abstract class and [override](http://javarevisited.blogspot.com/2011/12/method-overloading-vs-method-overriding.html) this method.

### 10. Can an abstract class contains the main method in Java?

Yes, an abstract class can contain the [main method](http://javarevisited.blogspot.sg/2011/12/main-public-static-java-void-method-why.html), it just another static method and you can execute the Abstract class with the main method until you don’t create any instance.

* **Object & Class**

**What is Object class in Java programming language?**

**FAQKey Concept**

\* Similar Java Interview Question Recently Asked @ Accenture, Citi, Expeditors, GlobalLogic, Google, Morgan Stanley, Oracle, Software AG, Vanguard,

Object class defined in *java.lang* package is the superclass of all other classes defined in Java programming language. Every class extends from the Object class either directly or indirectly. All classes inherit the instance methods defined in the Object class.

**What are the non-static methods defined in the Object class that are inherited by all classes?**

**FAQKey Concept**

\* Similar Java Interview Question Recently Asked @ Amadeus, Amdocs, Campus Labs, Deloitte, Ducat, Expeditors, FieldEZ, Goldman Sachs, Google, PWC, Rakuten, VirtUSA, Yodlee,

Object class defines eight non-static methods that are inherited by all other classes. A Java class can override any of these eight methods.

* **clone()** - If a class implements *cloneable* interface, then calling clone() method on its object returns a copy of the object. If a class does not implement the *cloneable* interface, and clone() method is called on its object, then the method throws a *CloneNotSupportedException* exception.

* **toString()** - You can get a string representation of any Java object by calling its toString() method. The toString() method is defined in the Object class and is inherited by all Java classes. The toString() method is usually overridden so that it returns a meaningful string representation of the object.

* **equals()** - You can check if an object equals another object by calling its equals() method and passing another object as a parameter to compare for equality.  
    
  The equals() method defined in the Object class uses the identity operator (==) to determine if two objects are equal. Hence this method returns correct result for primitive data types; but returns incorrect result for objects since the identity operator checks if the object references are equal instead of checking the logical equality of objects.  
    
  Hence equals() method is usually overridden to compare the logical equality of objects rather than their references.

* **hashcode()** - You get the hash code of an object by calling its hashcode() method. An objects hash code determines its memory address.  
    
  As a rule, if two objects are equal then their hash codes must also be equal. If you override equals() method, then you also have to override the hashcode() method to ensure that objects that are equal will have the same hash codes.

* **finalize()** - The Java garbage collector calls the finalize() method of an object just before the object is garbage collected. But you cannot control when, or if, an object is garbage collected. Hence, you should not rely on this method to perform critical tasks.

* **getClass()** - Calling getClass() method on an object returns a Class object which is a runtime class of this object. The Class class defines a number of methods which helps to find the metadata of a class - such as the class name, methods defined in the class, check if the class is an interface, check if the class is an interface etc.

* **wait()** - wait() is one of the three methods defined in Object class that facilitates thread to thread communication in multi-threaded Java programs. Calling wait() on an object of type Thread stops the execution of current thread until the processing of the other thread is complete.

* **notify()** - notify() is the second method defind in Java object class that facilitates thread to thread communication in multi-thread programming. notify() method will send an event notification or signal to a thread that is waiting in that object's waiting pool.

* **notifyAll()** - notifyAll() is the third method defind in Java object class that facilitates thread to thread communication in multi-thread programming. notifyAll() is similar to notify(), except that it sends notification to all threads that are waiting in that objects waiting pool

A screenshot of a computer

Description automatically generated

**What is the difference between method Overloading and method Overriding?**

**FAQKey Concept**

\* Similar Java Interview Question Recently Asked @ ACL Worldwide, Amadeus, American Express, Atos, CA Technologies, Campus Labs, Capital One, FGM Group, General Dynamics, Home Depot, IBM, Keysight Technologies, Northrop Grumman, Revaturte, TCS, Tech Mahindra, Techlogix, Thomson Reuters, TMC, Trustwave,

Methods having same name and return types but different arguments types can be declared within a class. These methods are called overloaded methods. This concept is known as method overloading

**\*\*\* See complete answer and code snippet in the Java Interview Guide.**

**What are final classes and final methods Java programming language?**

**FAQKey Concept**

\* Similar Java Interview Question Recently Asked @ Acceltree, Alcatel-Lucent, Antra, Apple, Aricent, Capgemini, Capttal One, C-Edge, ConnectWise, Faithlife, FDM, Financial Engines, GE, Goldman Sach, Group, Guidewire, HCL Technologies, IG, Incessant Technologies, Intuit, MathWorks, OpenEye, Oracle, Orbitz, Palo Alto Networks, Priceline, Rackspace, Realex Payments, Revature, Rolta, Sears, ServiceNow, SonicWall, TechFlow, Virtusa, VMWare, Wealthfront,

Final classes are Java classes that cannot be sub-classed. Final classes are declared using the keyword 'final'.

Final methods are methods that cannot be overridden by sub-classes. Final methods are declared using the keyword 'final'...

**\*\*\* See complete answer and code snippet in the Java Interview Guide.**

**What are Access and Non-Access modifiers that can be added to a class or class-member declaration?**

**FAQ**

\* Similar Java Interview Question Recently Asked @ ADP, Argonne National Laboratory, BAE Systems, Guidewire, Impetus Technologies, Polaris, Virtusa,

Access modifiers control how a class or its members can be accessed. Non-access modifiers control other aspests of a class and its members.

For your interview you have to know the following combination of modifiers.

1. Class access modifiers  
Class non-access modifiers  
3. Class-member access modifiers  
4. Class-member non-access modifiers.

The next four questions addresses these combinations.

**What are the access modifiers that can be added to a class?**

**FAQ**

Following access modifiers can be added to a class declaration.

**public** - A class declared with public access modifier is accessible to all classes.

**\*\*\* See complete answer and code snippet in the Java Interview Guide.**

**What are the Non-access modifiers that can be added to a class declaration?**

**FAQ**

Following non-access modifiers can be added to a class declaration.

**final** - A class declared as final cannot be sub-classed...

* **Access Modifiers**

**1. How many types of modifiers in Java?**

Ans: There are two types of modifiers in Java:

* Access modifiers
* Non- access modifiers

**2. What are Access modifiers in java?**

Ans: [Access Modifiers](https://www.scientecheasy.com/2020/06/access-modifiers-in-java.html/) are those modifiers that are used to restrict the visibility of classes, fields, methods, and constructors.

Java supports four types of access modifiers:

* Private
* Default
* Protected
* Public

a) **Private:** Private members of a class can be accessed only within the class. It cannot be accessed from outside the class.

b) **Default:** Default members of a class are accessible within the same package due to visible only within the package. They cannot be accessed from outside the package.

c) **Protected:** Protected members of a class are visible within the package. Therefore, we can only access within the package but can be accessed to the subclasses outside the package through the inheritance only.

d) **Public:** Public members are visible anywhere. So, we can access it anywhere within or outside the package.

**3. Can we have a private constructor in Java?**

Ans: Yes, we can have a private constructor in Java.

* The private constructor is used when we do not want to create the object of that class.
* We cannot create a subclass of that class.
* It is also used in Singleton design and Factory method design patterns.

**4. Which access modifiers can be used with a class?**

Ans: Public and Default access modifiers can be used with a class.

**5. Can we instantiate the object of derived class if the parent constructor is protected?**

Ans: No, we cannot instantiate the object of derived class if the parent constructor is protected.

**6. What are non-access modifiers in Java?**

Ans: There are four [non-access modifiers in Java](https://www.scientecheasy.com/2020/06/non-access-modifiers-in-java.html/). They are as follows:

* Static
* Final
* Abstract
* Synchronized

a) **Static:** This modifier is used to check that a member is a class member or instance member. If you declare a class as static, this class will be executed first.

b) **Final:** Final is a keyword that is used to restrict the users. In other words, it is used to restrict further modification of a class, field, or method. If a class is declared as ‘final’, the class cannot be subclassed.

c) **Abstract:** Abstract is a keyword that is used with a class or a method. An abstract class or abstract method is used for further modification. If a class is declared as ‘abstract’, the class cannot be instantiated.

d) **Synchronized:** It is used to achieve thread safeness. Only one thread can enter in a synchronized method or block at a given time.

**7. Can we declare a top-level class as private?**

Ans: No, we cannot declare a top-level or outer class as private. It can have either “public” or no modifier.

If you declare a top-level class as a private, the compiler will complain that the “modifier private is not allowed here” but an inner class can be private.

Inner class means class as a member of another class. The same is the case with protected.

**8. Can we declare an abstract method as private?**

Ans: No, an abstract method cannot be private. They must be declared as public, protected, or default so that they can be further modified.

**9. Can we declare a top-level class as protected?**

Ans: No, we cannot declare a class as protected. An inner class can be protected but not an outer class.

**10. Can a method or a class be final and abstract at the same time?**

Ans: No, it is not possible. A class or a method cannot be final or abstract at the same time because the final method or final class cannot be further modified whereas an abstract class or an abstract method must be modified further.

**11. Why are access modifiers used?**

Ans: The access modifiers are used to restrict the access of a class and its members. Access modifiers are used to reduce the visibility of the members of a class.

**12. Which is the default access modifier?**

Ans: ‘Internal’ is the default access modifier if no access modifier is mentioned with a class or its members. Internal is a keyword that is used for declaration.

**13. What is the default access modifier for Interface?**

Ans: The public is the default access modifier for the interface. No other access modifier is allowed for them

**14. Can we define struct members as protected?**

Ans: No, we cannot define struct members as protected because struct does not support inheritance.

**15. What is the default access specifier for a class, an interface, and struct declared directly with a namespace?**

Ans: Internal

**16. What is access modifier for enumeration?**

Ans: Enumeration members are always public. No other access modifiers are allowed.

**17. What is the role of private constructor in Java?**

Ans: If you declare any constructor of a class as private, we cannot create the object of a class from outside the class. In other words, we cannot create the subclass of that class.

**18. Which is the least restrictive access modifier in Java?**

Ans: Public

**19. Which is the most restrictive access modifier in Java?**

Ans: Private

**20. Which access modifier is also known as Universal access modifier?**

Ans: Public

**21. Explain visibility control in Java.**

Ans: Visibility control in Java is implemented by the access modifiers.

* **Arrays**

1) On which memory arrays are created in Java?

Arrays are created on dynamic memory by JVM. There is no question of static memory in Java everything (variable, array, object, etc.) is created on dynamic memory only.

2) Can we call the main() method of a class from another class?

Yes! We can call the main() method of a class from another class using Classname.main(). At the time of calling the main() method, we should pass a string type array to it.

3) What is an array in Java?

An array is a finite and ordered collection of homogeneous data elements. It is finite because it contains a limited number of elements. It is ordered because all the elements are stored one by one in a contiguous location of computer memory (heap) in a linear fashion. It is homogeneous because all elements of an array are of the same data type only. We can store either primitive types or object references into it.

4) What are the types of an array?

Arrays are generally categorized into two parts as described below:

* Single Dimensional Array
* Multi-Dimensional Array (2D and 3D arrays)

5) Is it possible to declare array size as negative?

No, it is not possible to declare array size as negative. Still, if we declare the negative size, there will be no compile-time error. But we get the NegativeArraySizeException at run-time.

6) What is the difference between int array[] and int[] array?

There is no difference between array[] and []array. Both array[] and []array are the ways to declare an array. The only difference between them is that if we are declaring more than one array in a line, we should use prefix []. If we are declaring a single array in a line, we should use postfix []. For example, consider the following declaration:

1. **int** array1[], array2;   //array1[] is an array while array2 is just a variable of type int
2. **int**[] arr1, arr2;  //both arr1 and arr2 are arrays of int type

7) How to copy an array in Java?

We can create a copy of an array in two ways, first one is manually by iterating over the array and the second one is by using the arrayCopy() method. Using the arrayCopy() method of the System class is the fastest way to copy an array and also allows us to copy a part of the array. These two methods are the popular ways to copy an array.

The other two methods to copy an array is to use the Arrays.copyOf() method and using clone() method.

8) What is the default value of the array?

When we create a new array, it always initialized with the default values. The default values of the array are:

* If an array is of byte, short, int, and long type, the default value is **0**.
* If an array is of float and double type, the default value is **0**.
* If an array is of Boolean type, the default value is **false**.
* If an array is of an Object type, the default value is **null**.

9) What do you understand by the jagged array?

A jagged array is a multidimensional array in which member arrays are of different sizes. For example, int array[][]=new int[3][]. The statement creates a two-dimensional jagged array.

10) What is an anonymous array also give an example?

Array reference that is not stored in a variable. It is used in the construction of other objects. Java's Polygon class has a constructor that parses anonymous array as a parameter.

1. Polygon(**int**[] xvalues, **int**[] yvalues, **int** n)

For example:

1. Polygon triangle=**new** Polygon(**new** **int**[] {0, 10, 5}, **new** **int**[] {10, 0, 5}, 3)

The above statement creates a triangle that an anonymous array.

11) How many ways to find the duplicate elements in an array?

There are the following five ways to find the duplicate array in Java.

* **Brute Force Method:** In this method, we compare each element of an array with the other elements. If any of the two elements are found equal, we consider them as duplicates. The method has time complexity O(n2).
* **Using HashSet:** We can also use the HashSet class to find the duplicate elements in an array. To find the duplicate elements, iterate over the array elements and insert them into HashSet by invoking add() method of the HashSet class. If the method returns false it means that the element is already present in the Set. It takes O(n) time to find the duplicate elements.
* **Using HashMap:** We know that HashMap uses key-value pair to store an element. When we use HashMap to find the duplicate array, we store the elements of the array as keys and the frequency of the elements as values. If the value of any key is greater than 1, the key is a duplicate element. Its time and space complexity is O(n). Using this method, we can also find the number of occurrences of duplicates.

12) Which operations can be performed on an array?

On an array, we can perform the searching, sorting, traversal, deletion, and insertion operation.

13) Consider the following statements and tell that the declaration is true or not. Also, specify the reason?

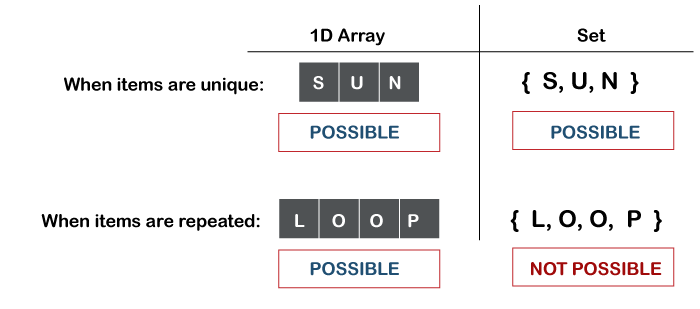
1. **int** y=56;
2. **int**[] number={12, 34, 90, y, 65};

The declaration of the above array is true. Because Java allows us to use two different array-specific syntax shortcuts both to initialize (put explicit values into an array's elements) and construct (instantiate the array object itself) in a single statement. The first statement is used to declare, create and initialize in one statement. The second statement does the following four things:

* **Declares** an int array reference variable named number.
* **Creates** an int array with a length of five (five elements).
* **Populates** the array's elements with the values 12, 34, 90, 56, and 65.
* **Assigns** the new array object to the reference variable number.

14) Can a Set be an array?

In Java, a Set is an array, but an array is not necessarily a Set. Because repetition is allowed in array but in Set. For example, consider the following figure:



15) Is it possible to make an array volatile?

Yes, we can make an array volatile in Java. But we only make the variable that is pointing to array volatile. If an array is changed by replacing individual elements that happen before the guarantee provided by volatile variables will not hold.

ADVERTISEMENT

ADVERTISEMENT

16) What happens if we declare an array without assigning the size?

It is not possible to declare an array without size. When we declare an array without assigning the size, it throws the compile-time error. For example, height=new int[].

17) Can we declare array size as negative?

No, the array size cannot be negative. If we declare an array with a negative size, it throws NegativeArraySizeException at run time.

18) When ArrayIndexOutOfBoundsException occurs?

ADVERTISEMENT

ADVERTISEMENT

The ArrayIndexOutOfBoundsException occurs when the program tries to access the index of an array. The exception also occurs when the index is higher than the size of the array or the index is negative.

19) Which method of the Arrays class can be used to search a specific element in an array?

The binarySearch() method of the Arrays class is used to search a specific element in an array. The method uses the binary search algorithm. The array must be in natural ordering before making this call. It is the simplest and most efficient method to find an element in a sorted array.

20) How to retrieve the class name of an array?

An array is an object. From the object we can retrieve the class name. We invoke the getClass() and getName() method that retrieves the class name of an array. The getClass() is the method of the Object class that returns the runtime class of the object. While the getName() is the method of the Class class that returns the name of the class/array class.

21) What is the difference between Array and ArrayList?

**Array:** Array is static. It is of fixed size. Its size cannot be changed once it is declared. It contains both primitive data types and objects of a class. Array does not have generic features.

**ArrayList:** ArrayList is dynamic in size. Its size or capacity automatically grows when we add element into it. It contains only the object entries. It has a generic feature.

22) How can we check an array contains values or not?

Java Arrays class provides two methods isExists() and contains() to check an array has elements or not. Both the methods return true if an array has elements else returns false.

23) What is the equilibrium index of an array also give an example?

If the sum of lower indices is equal to the sum of higher indices is called an equilibrium index of an array. For example, consider the following array: [-7, 1, 5, 2, -4, 3, 0], where:

a[0]=-7, a[1]=1, a[2]=5, a[3]=2, a[4]=-4, a[5]=3, a[6]=0

Let's find the equilibrium index. According to the definition:

The sum of lower indices is = a[0]+a[1]+a[2] = -7+1+5 = -1

The sum of higher indices is = a[4]+a[5]+a[6] = -4+3+0 = -1

We observe that the sum of lower indices is equal to the sum of higher indices. Hence, the equilibrium index is 3.

In the above array, 6 is also an equilibrium index because the sum of a[0] to a[5] is 0, and the value of index a[6] is 0.

24) What is left-rotation in an array?

Left-rotation is an operation that can be performed over an array. In this operation, each element of an array shifts 1 unit to the left. Therefore, the lowest index's value moves to the highest index. We can perform any number of rotations over an array. It is also known as a circular array. Let's perform left-rotation twice over an array [7, 8, 9, 2, 5, 6].

After one rotation, we get the array **[8, 9, 2, 5, 6, 7],** after the second rotation, we get **[9, 2, 5, 6, 8]**.

25) what are the advantages and disadvantages of an array?

**Advantages of Array**

* We can store multiple elements of the same type under a single variable.
* We can implement other data structures such as Stack, Queue, Tree, etc. with the help of an array.
* We can fetch data elements using the index at run time.

**Disadvantages of Array**

* Before using an array, it is mandatory to declare its size.
* It is a static structure, so we cannot increase or decreases memory allocation.
* Insertion and deletion operations are difficult because elements stored at the contiguous memory location. It also increases the cost.
* Allocate more memory than required is the wastage of memory.
* **Encapsulation**

**1. What is Encapsulation in Java? Why is it called Data hiding?**

Ans: The process of binding data and corresponding methods (behavior) together into a single unit is called encapsulation in Java.

In other words, encapsulation is a programming technique that binds the class members (variables and methods) together and prevents them from being accessed by other classes, thereby we can keep variables and methods safes from outside interference and misuse.

If a field is declared private in the class then it cannot be accessed by anyone outside the class and hides the fields within the class. Therefore, Encapsulation is also called data hiding.

**2. What are the important features of Encapsulation?**

Ans: Encapsulation means combining the data of our application and its manipulation in one place. It allows the state of an object to be accessed and modified through behavior. It reduces the coupling of modules and increases the cohesion inside them.

**3. What is the advantage of Encapsulation?**

Ans: There are the following advantages of encapsulation in Java. They are as follows:

* The encapsulated code is more flexible and easy to change with new requirements.
* It prevents the other classes to access the private fields.
* Encapsulation allows modifying implemented code without breaking other code who have implemented the code.
* It keeps the data and codes safe from external inheritance. Thus, Encapsulation helps to achieve security.
* It improves the maintainability of the application.

**4. What are the main benefits of using encapsulation in Java?**

Ans: The main benefits of using encapsulation are:

* The main benefit of encapsulation is the ability to modify the implemented code without breaking the others code who have implemented the code.
* It also provides us with maintainability, flexibility, and extensibility to our code.
* The fields of a class can be made read-only or write-only.
* A class can have total control over what is stored in its fields.

**5. How to achieve encapsulation in Java? Give an example.**

Ans: There are two key points that should be kept in mind to achieve the encapsulation in Java. They are as follows:

* Declare the variable of the class as private.
* Provide public setter and getter methods to modify the values of variables.

Let’s understand it with the help of an example program.

public class EncapsulationTest{

private String name;

private String idNum;

private int age;

public int getAge() {

return age;

}

public String getName() {

return name;

}

public String getIdNum() {

return idNum;

}

public void setAge( int newAge) {

age = newAge;

}

public void setName(String newName) {

name = newName;

}

public void setIdNum( String newId) {

idNum = newId;

}

}

**6. What is data hiding in Java?**

Ans: An outside person cannot access our internal data directly or our internal data should not go out directly. This oops feature is called data hiding n Java. After validation or authentication, the outside person can access our internal data.

**7. How to achieve Data hiding programmatically?**

Ans: By declaring data members (variables) as private, we can achieve or implement data hiding. If the variables are declared as private in the class, nobody can access them from outside the class.

The biggest advantage of data hiding is we can achieve security.

**8. What is a Tightly encapsulated class in Java?**

Ans: If each variable is declared as private in the class, it is called tightly encapsulated class in Java. For tightly encapsulated class, we are not required to check whether class contains getter and setter method or not and whether these methods are declared as public or not.

**9. What is the difference between Abstraction and Encapsulation?**  
Or, how abstraction is different from encapsulation in Java?

Ans: There are the following differences between Abstraction and Encapsulation:

a) Abstraction solves the problem at the design level whereas encapsulation solves the problem at the implementation level.

b) Abstraction is implemented in Java using Interface and Abstract class whereas encapsulation is implemented using private and protected access modifiers.

c) Abstraction is used to hide the unwanted data and giving relevant data whereas encapsulation is used for hiding data and code in a single unit to prevent access from outside.

d) The real-time example of Abstraction is TV Remote Button whereas the real-time example of Encapsulation is medical medicine.

**10. Can we achieve abstraction without encapsulation in Java?**

Ans: Yes, we can achieve abstraction without encapsulation because both are different things and different concepts.

**11. What are getter and setter methods in Java?**

Ans: In Java, setter method is a method that is used for updating the values of a variable. This method is also known as mutator method.

Getter method is a method that is used to retrieve the value of a variable or return the value of the private member variable. This method is also known as an accessor method.

**12. In the following code, radius is declared as private in the class Circle, and myCircle is an instance of class Circle. Does the code cause any error problems? If so, explain why?**

class Circle {

private double radius = 1;

/\*\* Find area of the circle \*/

public double getArea() {

return radius \* radius \* Math.PI;

}

public static void main(String[] args) {

Circle myCircle = new Circle();

System.out.println("Radius is " + myCircle.radius);

System.out.println("Area of cirle: " +myCircle.getArea());

}

}

Ans: No, the above code will not create any problem. The code will be compiled successfully. The output is: Radius is 1.0, Area of cirle: 3.141592653589793.

**13. Does reflection violates encapsulation in Java?**

Ans: Reflection violates encapsulation because it reveals the internal data structures.

**14. Explain design pattern based on encapsulation in java?**

Ans: In many design patterns, Java uses the encapsulation technique and one of them is Factory pattern which is used to create the objects.

Factory pattern is a better choice in creating the object of those classes whose creation logic can vary. It is also used for creating different implementations of the same interface.

‘BorderFactory class’ of JDK is a good example of encapsulation in Java which creates different types of ‘border’ and encapsulates creation logic of border.

**15. How can the variable of the EncapsulationTest be accessed by using getter and setter methods?**

Ans: The public setXXX() and getXXX() methods are access points of the instance variable of EncapsulationTest class. Basically, these methods are known as getter and setter methods.

Therefore, any class that wants to access variable should access them through these getters and setters. The variables of the EncapsulationTest class can be accessed  as shown in the following code:

public class RunEncapTest {

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

{

EncaptulationTest encap = new EncapsulationTest();

encap.setName("John");

encap.setAge(22);

encap.setId(123456);

System.out.println("Name: " +encap.getName()); System.out.println("Age: " +encap.getAge());

System.out.println("Id: " +encap.getId());

}

}

This would produce following output: Name: John Age: 22 Id: 123456.

* **Operators**

**1. What is an Operator in Java?**

Ans: A operator is a special symbol that tells the compiler to perform specific mathematical or logical operations. It is generally used in a program to perform a particular function on operands.

**2. What are the types of operators based on the number of operands?**

Ans: There are three types of operators in java based on the number of operands. They are:

* Unary operator
* Binary operator
* Ternary operator

**3. What is Unary operator in Java?**

Ans: A operator that acts on a single operand is called unary operator. It uses a single variable.

**4. What is a Binary operator in Java?**

Ans: A operator that acts on two operands is called binary operator. It uses two variables.

**5. What is Ternary operator in Java?**

Ans: A operator that acts on three operands is called ternary operator. It uses three variables.

**6. What is Expression in Java?**

Ans: An expression in java is a combination of operators and operands that is used to perform mathematical or logical calculations. In simple words, it is a combination of variables, constants, and operators.

For example, an expression is x+5. Here, the operand x is a variable, operand 5 is a constant, and + is an operator that acts on these two operands and produces the desired result.

**7. What is a symbolic operator in Java? What are the types of operators based on symbols?**

Ans: If a symbol like +, -, \*, etc is used as an operator, it is called symbolic operator.

There are mainly eight types of operators based on symbols. They are as follows:

* Arithmetic operators ⇒ +, -, \*, /, etc.
* Relational operators ⇒ <, >, <=, >=, = =, !=.
* Logical operators ⇒ &&, ||, !.
* Assignment operators ⇒ =,
* Increment and decrement operators ⇒ + +, – –
* Conditional operators ⇒ ?:
* Bitwise operators ⇒ &, !, ^, ~, <<, >>, >>>
* Shift operators ⇒ <<, >>, >>>.

**8. What is named operator in Java?**

Ans: If a keyword is used as an operator, it is called named operator. The named operator is instanceof.

## Interview Questions based Arithmetic Operator in Java

**1. What is Arithmetic operators in Java?**

Ans: Operators that are used to performing fundamental arithmetic operations such as addition, subtraction, multiplication, and division on numeric data types are called arithmetic operators.

**2. What are the types of arithmetic operators?**

Ans: Java supports five types of arithmetic operators. They are:

* + Addition
* – Subtraction
* \* Multiplication
* / Division
* % Modulo division (Remainder)

**3. What are the priority levels of arithmetic operation in Java?**

Ans: There are two priority levels of arithmetic operation in java. They are as follows:

* High priority ⇒ \* / %
* Low priority ⇒ + –

**4. What will be the output of the following program?**

a)

public class Test {

public static void main(String[] args)

{

int x = 2, y = 5;

int exp1 = (x \* y / x);

int exp2 = (x \* (y / x));

System.out.println(exp1);

System.out.println(exp2);

}

}

Ans: Output is 5, 4.

b)

public class Test {

public static void main(String[] args)

{

int x = 10, y = 5;

int exp1 = (y \* (x / y + x / y));

int exp2 = (y \* x / y + y \* x / y);

System.out.println(exp1);

System.out.println(exp2);

}

}

Ans: Output is 20, 20.

**5. What is the result of the following code?**

public class Test {

public static void main(String[] args)

{

double exp1 = (3.0 \* 2 / 4);

int exp2 = (3 \* -2 % 4);

System.out.println(exp1);

System.out.println(exp2);

}

}

Ans: Results are 1.5, -2.

**6. What is the output of following code?**

public class Test {

public static void main(String[] args)

{

int x = 9, y = 12, z = 3;

int exp1 = x - y/3 + z \* 2 - 1;

int exp2 = (x - y)/3 + ((z \* 2) - 1);

System.out.println(exp1);

System.out.println(exp2);

}

}

Ans: Outputs are 10, 4.

**7. Is the arithmetic expression valid? If yes, what is the output of the following expression?**

public class Test {

public static void main(String[] args)

{

int x = 9, y = 12;

int a = 2, b = 4, c = 6;

int exp = (3 + 4 \* x)/5 - 10 \* (y - 5) \* (a + b + c)/x + 9 \* (4/x + (9 + x)/y);

System.out.println(exp);

}

}

Ans: Yes, the arithmetic expression is valid. The output is -77.

**8. What would be the result of the following arithmetic expression?**

public class Test {

public static void main(String[] args)

{

int x = 9, y = 12;

int a = 2, b = 4, c = 6;

int exp = 4/3 \* (x + 34) + 9 \* (a + b \* c) + (3 + y \* (2 + a)) / (a + b\*y);

System.out.println(exp);

}

}

Ans: 278

For explanation of answer, follow this tutorial: [Java Operators | Arithmetic Operators Example](https://www.scientecheasy.com/2020/05/java-operators-arithmetic.html/)

## Interview Questions based on Relational Operators

**1. What are Relational operators in Java?**

Ans: Relational operators in Java are those operators that are used to perform the comparison between two numeric values or two quantities. They are generally used in looping and branching statements to create conditions.

**2. How many types of relational operators are supported by Java?**

Ans: Java supports six types of relational operators. They are as follows:

* < Less than
* <= Less than or equal to
* > Greater than
* >= Greater than or equal to
* == Equal to
* != Not equal to

**3. What is the result of relational operator when it is used in looping or branching statement?**

Ans: The result of all relational operators is always of a boolean type. It returns always true or false.

**4. What will be the output of the following program?**

public class Test {

public static void main(String[] args)

{

int x = 9, y = 12;

int a = 2, b = 4;

boolean exp = 4/3 \* (x + 34) < 9 \* (3 + y \* (2 + a)) / (a + b\*y);

System.out.println(exp);

}

}

Ans: false

## Interview Questions based on Logical operators

**1. What are Logical operators in Java?**

Ans: Logical operators in Java are those operators that are used to form compound conditions by combining two or more conditions or relational expressions.

These operators are also called Boolean operators because they return a boolean value.

**2. How many types of logical operators are supported by Java?**

Ans: Java supports three types of logical operators. They are as follows:

* && AND operator
* || OR operator
* ! NOT operator

**3. What will be the output of the following program?**

public class Test {

public static void main(String[] args)

{

int x = 200, y = 50, z = 100;

if(x > y && y > z)

{

System.out.println("Hello");

}

if(z > y && z < x)

{

System.out.println("Java");

}

if((y+200) < x && (y+150) < z)

{

System.out.println("Hello Java");

}

}

}

Ans: Output: Java

**4. What will be the output of the following code?**

public class Test{

public static void main(String[] args)

{

int x = 1, y = 2, z = 5;

if(x == 1 || x > y || x > z)

{

System.out.println("One");

}

if(x == y || y == 2 || z == 5)

{

System.out.println("Two");

}

if(x == y || y == z || z == x)

{

System.out.println("Three");

}

}

}

Ans: Output: One, Two

**5. Will the following code compile successfully? If yes, What will be the output of the program?**

public class Test {

public static void main(String[] args)

{

int x = 1, y = 2, z = 5;

System.out.println("x: " +(!((x + 2) == (1 + 2))));

System.out.println("y: " +(!(y == z)));

System.out.println("z>x: " +(!(z > x)));

if(!(x == y) && ((y + 5) > z) && (!((z - 3) == 0)))

{

System.out.println("Hello");

}

}

}

Ans: Output: x: false, y: true, z>x: false, Hello.

For explanation, go to this tutorial: [Logical operators in Java](https://www.scientecheasy.com/2020/05/logical-operators-in-java.html/)

## Interview Questions Based on Assignment Operators

**1. What is an Assignment operator in Java?**

Ans: An operator which is used to store a value into a particular variable is called assignment operator in java.

**2. Is there any difference between x += y and x = x + y expressions?**

Ans: No, there is no any difference between x += y and x = x + y expressions. Both are equivalent to each other.

**3. What will be the result of the following code?**

public class Test {

public static void main(String[] args)

{

int x = 20, y = 30, z = 50;

x += y;

y -= x + z;

z \*= x \* y;

System.out.println("x = " +x );

System.out.println("y = " +y );

System.out.println("z = " +z );

}

}

Ans: Output: x = 50, y = -70, z = -175000

**4. Identify the errors in the following code.**

public class Test {

public static void main(String[] args)

{

int x = y = z = 10;

System.out.println(x + " " + y + " " + z);

}

}

Ans: Compile time error because we cannot assign a variable to variable. For example, int x = y;

**5. Will the code compile successfully? If yes, what will be the output of code?**

public class Test {

public static void main(String[] args)

{

int x, y, z;

x = y = z = 2;

x += y;

y -= z;

z /= (x + y);

System.out.println(x + " " + y + " " + z);

}

}

Ans: Yes, the code will be compiled successfully. No error. The output is 4, 0, 0.

**6. What will be the output of code if no error?**

public class Test {

public static void main(String[] args)

{

int x, y, z;

1 = x;

y = z = 2;

int a = x + y + z;

System.out.println(x + " " + y + " " + z);

}

}

Ans: Compile time error because the left-hand side of an assignment must be a variable.

* **Polymorphism**

**1. What is Polymorphism in Java OOPs?**

Ans: Polymorphism in java is one of the core concepts of object-oriented programming system. Polymorphism means “many forms” in Greek. That is one thing that can take many forms.

Polymorphism is a concept by which we can perform a single task in different ways. That is, when a single entity (object) behaves differently in different cases, it is called polymorphism.

In other words, if a single object shows multiple forms or multiple behaviors, it is called polymorphism.

**2. What are the types of Polymorphism in Java?**

Ans: There are two types of polymorphism in java. They are:

* Static polymorphism (Compile time Polymorphism)
* Dynamic polymorphism (Runtime Polymorphism)

**3. What are different ways to achieve or implement polymorphism in Java?**

Ans: Polymorphism in Java can be primarily achieved by subclassing or by implementing an interface. The subclasses can have their own unique implementation for certain features and can also share some of the functionality through inheritance.

**4. How is Inheritance useful to achieve Polymorphism in Java?**

Ans: Inheritance represents the parent-child relationship between two classes and polymorphism take the advantage of that relationship to add dynamic behavior in the code (or to make the program more dynamic).

**5. What are the advantages of Polymorphism?**  
Or what is the use of polymorphism?

Ans: There are the following advantages of polymorphism in java:

a. Using polymorphism, we can achieve flexibility in our code because we can perform various operations by using methods with the same names according to requirements.

b. The main benefit of using polymorphism is when we can provide implementation to an abstract base class or an interface.

**6. What are the differences between Polymorphism and Inheritance in Java?**

Ans: The differences between polymorphism and inheritance in java are as follows:

a. Inheritance represents the parent-child relationship between two classes. On the other hand, polymorphism takes the advantage of that relationship to make the program more dynamic.

b. Inheritance helps in code reusability in child class by inheriting behavior from parent class. On the other hand, polymorphism enables child class to redefine already defined behavior inside parent class.

Without polymorphism, it is not possible for a child class to execute its own behavior.

**7. What is Compile time polymorphism (Static polymorphism)?**

Ans: A polymorphism where object binding with methods happens at compile time is called static polymorphism or compile-time polymorphism.

In static polymorphism, the behavior of method is decided at compile-time based on the parameters or arguments of method.

8. How to achieve or implement static polymorphism in Java?

Ans: Static polymorphism can be achieved by method overloading. Other examples of compile time polymorphism are constructor overloading and method hiding.

**9. What is the output of the following program if no error?**

class A {

void sum(int x, int y) {

System.out.println("Sum of two numbers: " +(x+y));

}

void sum(int x, int y, int z) {

System.out.println("Sum of three numbers: " +(x+y+z));

}

public static void main(String[] args){

A a = new A();

a.sum(20, 30);

a.sum(30, 40, 50);

}

}

Ans: The output is:

Sum of two numbers: 50  
Sum of three numbers: 120

**10. Identify the errors in the following code.**

class A {

void sum(int x, int y) {

System.out.println("Sum of two numbers: " +(x+y));

}

void sum(int y, int x) {

System.out.println("Sum of three numbers: " +(x+y));

}

public static void main(String[] args){

A a = new A();

a.sum(20, 30);

}

}

Ans: Duplicate method error.

**11. What is the output of the following program if no errors?**

class A {

void m1(String x){

System.out.println("One");

}

protected void m1(String x, String y){

System.out.println("Two");

}

public static void main(String[] args){

A a = new A();

a.m1("ABC");

a.m1("PQR", "XYZ");

}

}

Ans: The output is One, Two.

**12. How Java compiler differentiate between methods in Compile time Polymorphism?**

Ans: During compilation, Java compiler differentiates multiple methods having the same name by their signatures.

**13. Is there any errors in the following code? Will the code compile successfully?**

class A {

String m1(String x){

System.out.println("One");

return "ABC";

}

String m1(String y){

System.out.println("Two");

return "PQR";

}

public static void main(String[] args){

A a = new A();

a.m1("ABC");

}

}

Ans: Duplicate method error in the above code but the code will be compiled successfully. The output is One.

**14. What is Runtime Polymorphism (Dynamic Polymorphism)?**

Ans: A polymorphism where object binding with methods happens at runtime is called runtime polymorphism. In runtime polymorphism, the behavior of a method is decided at runtime.

JVM (Java Virtual Machine) binds the method call with method definition/body at runtime and invokes the relevant method during runtime when the method is called. This happens because objects are created at runtime and the method is called using an object of the class.

**15. How to achieve/implement dynamic polymorphism in Java?**

Ans: Dynamic or runtime polymorphism can be achieved or implemented via method overriding in java. It is another way to implement polymorphism and a common approach when we have an IS-A relationship.

**16. Is it possible to implement runtime polymorphism by data members in Java?**

Ans: No, we cannot implement runtime polymorphism by data members in java.

**17. Will the code compile successfully? If yes, what will be the output of the program?**

class A {

void m1(A a) {

System.out.println("m1 method in class A");

}

}

class B extends A {

public void m1(A a) {

System.out.println("m1 method in class B");

}

}

public class Test{

public static void main(String[] args){

A a = new A();

a.m1(a);

a.m1(new B());

B b = new B();

b.m1(null);

a = b;

a.m1(null);

a.m1(new A());

}

}

Ans: Yes, the code will be compiled successfully. The output is given below:

m1 method in class A  
m1 method in class A  
m1 method in class B  
m1 method in class B  
m1 method in class B

**17. What is the output of the following program if no errors?**

class A {

void m1(String x){

System.out.println("One");

}

}

class B extends A {

public void m1(String x){

System.out.println("Two");

super.m1(null);

}

}

public class Test{

public static void main(String[] args){

A a = new B();

a.m1(null);

}

}

Ans: The output is Two, One.

**18. Identify the errors in the following code.**

class A {

void m1(String x){

System.out.println("One");

}

}

class B extends A {

}

public class Test{

public static void main(String[] args){

A a = new B();

a.m1(new A());

}

}

Ans: The method m1(String) in type A is not applicable for the arguments (A).

**19. What is the output of the below program if no errors?**

class A {

void m1(Object obj){

System.out.println("One");

}

}

class B extends A {

void m1(Object obj){

super.m1(null);

System.out.println("Two");

}

void m2(Object obj){

System.out.println("Three");

this.m1(null);

}

}

public class Test{

public static void main(String[] args){

A a = new B();

a.m1(new A());

B b = new B();

b.m2(new B());

}

}

Ans: No error. The above code will be compiled successfully. The output is given below:

One  
Two  
Three  
One  
Two

**20. What are the differences between compile-time polymorphism and runtime polymorphism in java?**

Ans: There are three main differences between compile-time polymorphism and runtime polymorphism that are as follows:

a) In the compile-time polymorphism, the behavior of a method is decided at compile-time. Hence, Java compiler binds method calls with method definition/body during compilation.

In runtime polymorphism, the behavior of a method is decided at runtime, JVM binds the method call with method definition at runtime and invokes the relevant method during runtime when the method is called.

b) Compile time polymorphism is also known as early binding because the binding is performed at compile time.

Runtime polymorphism is also known as late binding because the binding is performed at runtime.

c) Compile time polymorphism can be achieved via method overloading.

Runtime polymorphism can be achieved via method overriding.

**21. What is Binding in Java?**

Ans: The connecting (linking) between a method call and method definition is called [binding in java](https://www.scientecheasy.com/2020/02/static-and-dynamic-binding-in-java.html/).

**22. What are the types of binding in Java?**

Ans: There are two types of binding in java. They are as follows:

a. Static Binding (also known as Early Binding).  
b. Dynamic Binding (also known as Late Binding).

**23. What is Static binding in Java?**

Ans: The binding that happens during compilation is called static binding in java. This binding is resolved at the compiled time by the compiler.

**24. How Java compiler performs static binding?**

Ans: Java compiler just checks which method is going to be called through reference variable and method definition exists or not.

It does not check the type of object to which a particular reference variable is pointing to it.

**25. Why static binding is also called early binding in Java?**

Ans: Static binding is also called early binding because it takes place before the program actually runs.

**26. Give an example of static binding.**

Ans: An example of static binding is method overloading.

**27. What is Dynamic binding in Java?**

Ans: The binding which occurs during runtime is called dynamic binding in java. This binding is resolved based on the type of object at runtime.

**28. How JVM performs dynamic binding in Java?**

Ans: In the dynamic binding, the actual object is used for binding at runtime. JVM resolved the method calls based on the type of object at runtime. The type of object cannot be determined by the compiler.

**29. Why Dynamic binding is also called late binding in java?**

Ans: Dynamic binding is also called late binding or runtime binding because binding occurs during runtime.

**30. Give an example of dynamic binding in Java.**

Ans: An example of dynamic binding is method overriding.

**31. What is the difference between static binding and dynamic binding in Java?**

Ans: Refer to this tutorial: [Polymorphism in Java OOPs with Example](https://www.scientecheasy.com/2020/07/polymorphism-in-java.html/)

**32. Why binding of private, static, and final methods are always static binding in Java?**

Ans: Static binding is better performance-wise because java compiler knows that all such methods cannot be overridden and will always be accessed by object reference variable.

Hence, the compiler doesn’t have any difficulty in binding between a method call and method definition. That’s why binding for such methods is always static.

* **Synchronization**

### What is synchronization of threads?

Since Threads run in parallel, a new problem arises. What if thread1 modifies data which is being accessed by thread2? How do we ensure that different threads don’t leave the system in an inconsistent state? This problem is usually called synchronization problem.

Let’s first look at an example where this problem can occur. Consider the code in the setAndGetSum method.

int setandGetSum(int a1, int a2, int a3) {

cell1 = a1;

sleepForSomeTime();

cell2 = a2;

sleepForSomeTime();

cell3 = a3;

sleepForSomeTime();

return cell1 + cell2 + cell3;

}

If following method is running in two different threads, funny things can happen. After setting the value to each cell, there is a call for the Thread to sleep for some time. After Thread 1 sets the value of cell1, it goes to Sleep. So, Thread2 starts executing. If Thread 2 is executing “return cell1 + cell2 + cell3;”, it uses cell1 value set by Thread 1 and cell2 and cell3 values set by Thread 2. This results in the unexpected results that we see when the method is run in parallel. What is explained is one possible scenario. There are several such scenarios possible.

The way you can prevent multiple threads from executing the same method is by using the synchronized keyword on the method. If a method is marked synchronized, a different thread gets access to the method only when there is no other thread currently executing the method.

Let’s mark the method as synchronized:

synchronized int setandGetSum(int a1, int a2, int a3) {

cell1 = a1;

sleepForSomeTime();

cell2 = a2;

sleepForSomeTime();

cell3 = a3;

sleepForSomeTime();

return cell1 + cell2 + cell3;

}

### Can you give an example of a synchronized block?

All code which goes into the block is synchronized on the current object.

void synchronizedExample2() {

synchronized (this){

//All code goes here..

}

}

### Can a static method be synchronized?

Yes. Static methods and block are synchronized on the class. Instance methods and blocks are synchronized on the instance of the class i.e. an object of the class. Static synchronized methods and instance synchronized methods don’t affect each other. This is because they are synchronized on two different things. Consider the example below.

synchronized static int getCount(){

return count;

}

static int getCount2(){

synchronized (SynchronizedSyntaxExample.class) {

return count;

}

}

### What is the use of join method in threads?

Join method is an instance method on the Thread class. Let's see a small example to understand what join method does. Let’s consider the thread's declared below: thread2, thread3, thread4

ThreadExample thread2 = new ThreadExample();

ThreadExample thread3 = new ThreadExample();

ThreadExample thread4 = new ThreadExample();

Let’s say we would want to run thread2 and thread3 in parallel but thread4 can only run when thread3 is finished. This can be achieved using join method.Look at the example code below: thread3.join() method call forces the execution of main method to stop until thread3 completes execution. After that, thread4.start() method is invoked, putting thread4 into a Runnable State.

thread3.start();

thread2.start();

thread3.join();//wait for thread 3 to complete

System.out.println("Thread3 is completed.");

thread4.start();

##### **Overloaded Join method**

Join method also has an overloaded method accepting time in milliseconds as a parameter. In below example, main method thread would wait for 2000 ms or the end of execution of thread4, whichever is minimum.

thread4.join(2000);

### Describe a few other important methods in Threads?

##### **Thread yield method**

Yield is a static method in the Thread class. It is like a thread saying " I have enough time in the limelight. Can some other thread run next?".

A call to yield method changes the state of thread from RUNNING to RUNNABLE. However, the scheduler might pick up the same thread to run again, especially if it is the thread with highest priority.

Summary is yield method is a request from a thread to go to Runnable state. However, the scheduler can immediately put the thread back to RUNNING state.

##### **Thread sleep method**

sleep is a static method in Thread class. sleep method can throw a InterruptedException. sleep method causes the thread in execution to go to sleep for specified number of milliseconds.

### What is a deadlock?

Let’s consider a situation where thread1 is waiting for thread2 ( thread1 needs an object whose synchronized code is being executed by thread1) and thread2 is waiting for thread1. This situation is called a Deadlock. In a Deadlock situation, both these threads would wait for one another for ever.

### What are the important methods in java for inter-thread communication?

Important methods are wait, notify and notifyAll.

### What is the use of wait method?

Below snippet shows how wait is used. wait method is defined in the Object class. This causes the thread to wait until it is notified.

synchronized(thread){

thread.start();

thread.wait();

}

### What is the use of notify method?

Below snippet shows how notify is used. notify method is defined in the Object class. This causes the object to notify other waiting threads.

synchronized (this) {

calculateSumUptoMillion();

notify();

}

### What is the use of notifyAll method?

If more than one thread is waiting for an object, we can notify all the threads by using notifyAll method.

thread.notifyAll();

### Can you write a synchronized program with wait and notify methods?

package com.rithus.threads;

class Calculator extends Thread {

long sumUptoMillion;

long sumUptoTenMillion;

public void run() {

synchronized (this) {

calculateSumUptoMillion();

notify();

}

calculateSumUptoTenMillion();

}

private void calculateSumUptoMillion() {

for (int i = 0; i < 1000000; i++) {

sumUptoMillion += i;

}

System.out.println("Million done");

}

private void calculateSumUptoTenMillion() {

for (int i = 0; i < 10000000; i++) {

sumUptoTenMillion += i;

}

System.out.println("Ten Million done");

}

}

public class ThreadWaitAndNotify {

public static void main(String[] args) throws InterruptedException {

Calculator thread = new Calculator();

synchronized(thread){

thread.start();

thread.wait();

}

System.out.println(thread.sumUptoMillion);

}

}

##### **Output**

Million done

499999500000

Ten Million done

* **Abstraction**

**1. What is Abstraction in Java?**

Ans: [Abstraction in Java](https://www.scientecheasy.com/2020/05/java-abstraction.html/) is a technique by which we can hide the data that is not required to users. It hides all unwanted data so that users can work only with the required data.

**2. How to achieve or implement Abstraction in Java?**

Ans: There are two ways to implement abstraction in java. They are as follows:

a) Abstract class (0 to 100%)  
b) Interface (100%)

**3. What is Abstract class in Java? How to define it?**

Ans: An abstract class in java is a class that is declared with an abstract keyword.

Example of Abstract class:

abstract class Test {

abstract void show();

}

**4. What is the difference between abstract class and concrete class?**

Ans: There are mainly two differences between an abstract class and concrete class. They are:

a) We cannot create an object of abstract class. Only objects of its non-abstract (or concrete) sub classes can be created.

b) It can have zero or more abstract methods that are not allowed in a non-abstract class (concrete class).

**5. What is Abstract in Java?**

Ans: Abstract is a non-access modifier in java that is applicable for classes, interfaces, methods, and inner classes.

**6. Can abstract modifier applicable for variables?**

Ans: No.

**7. What is Abstract method in Java?**

Ans: A method which is declared with abstract modifier and has no implementation (means no body) is called abstract method in java.

It does not contain any body. It has simply a signature declaration followed by a semicolon. It has the following general form as given below.

Syntax:

abstract type MethodName(arguments); // No body

For example:

abstract void msg(); // No body.

**8. Can an abstract method be declared as static?**

Ans: No.

**9. Can an abstract method be declared with private modifier?**

Ans: No, it cannot be private because the abstract method must be implemented in the child class. If we declare it as private, we cannot implement it from outside the class.

**10. What is Concrete method in Java?**

Ans: A concrete method in Java is a method which has always the body. It is also called a complete method in java.

**11. When to use Abstract class in Java?**

Ans: An abstract class can be used when we need to share the same method to all non-abstract sub classes with their own specific implementations.

**12. When to use Abstract method in Java?**

Ans: An abstract method can be used

a) When the same method has to perform different tasks depending on the object calling it.  
b) When you need to be overridden in its non-abstract subclasses.

**13. Is abstract class a pure abstraction in Java?**

Ans: No, It provides 0 to 100% abstraction.

**14. Is it possible to create an object of abstract class in Java?**

Ans: No. It is not possible but we can create an object of its subclass.

**15. Is it possible that an abstract class can have without any abstract method?**

Ans: Yes.

**16. Can an abstract class have constructor?**

Ans: Yes.

**17. Is abstract class allow to define private, final, static, and concrete methods?**

Ans: Yes.

**18. Is it possible to achieve multiple inheritance through abstract class?**

Ans: No.

**19. Can we make an abstract class without abstract keyword?**

Ans: No, a class must be declared with abstract keyword to make an abstract class.

**20. Can we define an abstract method inside non-abstract class (concrete class)?**

Ans: No, we cannot define an abstract method in non-abstract class.

For example:

class Test {

abstract void show();

}

The above code will generate a compile-time error.

**21. Which among the following code have errors?**

a) abstract class A {

void m1();

}

b) public class A {

abstract void m1();

}

c) abstract public class A {

abstract void m1();

}

d) abstract public class A {

void m1() { }

}

e) public abstract class A {

abstract void m1();

A(){ }

void m2() { }

}

f) public abstract class A {

abstract int x = 100;

abstract void m1();

abstract void m2();

}

g) public abstract class A {

abstract void m1();

}

public class Test {

public static void main(String[] args) {

A a = new A();

}

}

h) public abstract class A {

abstract void m1();

A(){ }

static void m2() {System.out.println("Hello Java!"); }

}

public class B extends A {

void m1(){

A.m2();

}

}

i) public abstract class A {

abstract void m1();

private A(){ }

}

public class B extends A { }

Ans: a, b, f, g, i.

**22. Will the following code compile successfully? If yes, what will be the output of program?**

public abstract class A {

abstract void m1(A a);

}

public class B extends A {

void m1(A a) {

System.out.println("One");

}

}

public class C extends B {

void m1(B b) {

System.out.println("Two");

super.m1(new B());

}

}

public class Test {

public static void main(String[] args) {

C c = new C();

c.m1(new B());

}

}

Ans: Yes, the above code will be compiled successfully. The output of above program is Two, One.

**23. What will happen if we do not override all abstract methods in subclass?**  
Or, what will happen if we do not provide implementation for all abstract methods in subclass?

Ans: Java compiler will generate compile time error. We will have to override all abstract methods in subclass.

**24. What is the difference between Abstraction and Encapsulation?**

Ans: Abstraction hides the implementation details from users whereas, [encapsulation](https://www.scientecheasy.com/2020/07/encapsulation-in-java.html/) wraps (binds) data and code into a single unit.

**25. Why abstract class has constructor even though you cannot create object?**

Ans: We cannot create an object of abstract class but we can create an object of subclass of abstract class. When we create an object of subclass of an abstract class, it calls the constructor of subclass.

This subclass constructor has a super keyword in the first line that calls constructor of an abstract class. Thus, the constructors of an abstract class are used from constructor of its subclass.

If the abstract class doesn’t have constructor, a class that extends that abstract class will not get compiled.

**26. Why should we create reference to superclass (abstract class reference)?**

Ans: We should create a reference of the superclass to access subclass features because superclass reference allows only to access those features of subclass which have already declared in superclass.

If you create an individual method in subclass, the superclass reference cannot access that method. Thus, any programmer cannot add their own additional features in subclasses other than whatever is given in superclass.

**27. What is the advantage of Abstract class in Java?**

Ans: The main advantages of using abstract class are as follows:

* Abstract class makes programming better and more flexible by giving the scope of implementing abstract methods.
* Programmer can implement abstract method to perform different tasks depending on the need.
* We can easily manage code.

**28. Will the code compile successfully? If yes, what will be the output?**

public abstract class A {

abstract void m1();

}

public class B extends A {

void m1(){

System.out.println("m1 in class B");

}

}

public class Test {

public static void main(String[] args) {

B b = new B();

b.m1();

}

}

Ans: Yes, the code will be compiled successfully. The output is m1 in class B.

**29. Consider the below given code.**

public abstract class A {

abstract void m1();

void m2(){

System.out.println("One”);

}

}

**How to call m2() method in the above code?**

Ans: Make it static and call as A.m2();

**30. Identify the errors in the following code.**

public abstract class A {

abstract void m1();

void m2(){

System.out.println("One");

}

}

public class B extends A {

void m2(){

System.out.println("Two");

}

}

public class Test {

public static void main(String[] args) {

B b = new B();

b.m2();

}

}

Ans: The abstract method m1() has been not implemented (overridden) in subclass B. Therefore, we will get a compile-time error.

**31. Will the given code compile successfully? If yes, how?**

a)

public abstract class A {

abstract void m1();

}

public class B extends A {

private void m1(){

System.out.println("One");

}

}

public class Test {

public static void main(String[] args) {

B b = new B();

b.m1();

}

}

Ans: No, the code will not be compiled successfully because we cannot reduce the visibility of the inherited method from A.

b)

public abstract class A {

abstract void m1();

}

public class B extends A {

protected void m1(){

System.out.println("One");

}

}

public class Test {

public static void main(String[] args) {

B b = new B();

b.m1();

}

}

Ans: Yes, the code will be compiled successfully. The output is One.

c)

public abstract class A {

public abstract void m1();

}

public class B extends A {

protected void m1() {

System.out.println("One");

}

}

public class Test {

public static void main(String[] args) {

B b = new B();

b.m1();

}

}

Ans: No, compile-time error because we cannot reduce the visibility of inherited method m1() from A.

d)

public abstract class A {

private abstract void m1();

}

public class B extends A {

protected void m1(){

System.out.println("One");

}

}

public class Test {

public static void main(String[] args) {

B b = new B();

b.m1();

}

}

Ans: Abstract method m1() in class A cannot be private.

e)

public abstract class A {

protected abstract void m1();

}

public class B extends A {

void m1(){

System.out.println("One");

}

}

public class Test {

public static void main(String[] args) {

B b = new B();

b.m1();

}

}

Ans: No, compile time error.

* **Annotations**

### ****Q1. What Are Annotations? What Are Their Typical Use Cases?****

Annotations are metadata bound to elements of the source code of a program and have no effect on the operation of the code they operate.

Their typical uses cases are:

* **Information for the compiler** – with annotations, the compiler can detect errors or suppress warnings
* **Compile-time and deployment-time processing** – software tools can process annotations and generate code, configuration files, etc.
* **Runtime processing** – annotations can be examined at runtime to customize the behavior of a program

### ****Q2. Describe Some Useful Annotations from the Standard Library.****

There are several annotations in the java.lang and java.lang.annotation packages, the more common ones include but not limited to:

* @Override – marks that a method is meant to override an element declared in a superclass. If it fails to override the method correctly, the compiler will issue an error
* @Deprecated – indicates that element is deprecated and should not be used. The compiler will issue a warning if the program uses a method, class, or field marked with this annotation
* @SuppressWarnings – tells the compiler to suppress specific warnings. Most commonly used when interfacing with legacy code written before generics appeared
* @FunctionalInterface – introduced in Java 8, indicates that the type declaration is a functional interface and whose implementation can be provided using a Lambda Expression

### ****Q3. How Can You Create an Annotation?****

Annotations are a form of an interface where the keyword interface is preceded by @, and whose body contains annotation type element declarations that look very similar to methods:

**public** @interface SimpleAnnotation {

String **value**();

**int**[] types();

}Copy

After the annotation is defined, yon can start using it in through your code:

@SimpleAnnotation(value = "an element", types = 1)

**public** **class** **Element** {

@SimpleAnnotation(value = "an attribute", types = { 1, 2 })

**public** Element nextElement;

}Copy

Note that, when providing multiple values for array elements, you must enclose them in brackets.

Optionally, default values can be provided as long as they are constant expressions to the compiler:

**public** @interface SimpleAnnotation {

String **value**() **default** "This is an element";

**int**[] types() **default** { 1, 2, 3 };

}Copy

Now, you can use the annotation without those elements:

@SimpleAnnotation

**public** **class** **Element** {

// ...

}Copy

Or only some of them:

@SimpleAnnotation(value = "an attribute")

**public** Element nextElement;Copy

### ****Q4. What Object Types Can Be Returned from an Annotation Method Declaration?****

The return type must be a primitive, String, Class, Enum, or an array of one of the previous types. Otherwise, the compiler will throw an error.

Here’s an example code that successfully follows this principle:

**enum** **Complexity** {

LOW, HIGH

}

**public** @interface ComplexAnnotation {

Class<? **extends** **Object**> value();

**int**[] types();

Complexity **complexity**();

}Copy

The next example will fail to compile since Object is not a valid return type:

**public** @interface FailingAnnotation {

Object **complexity**();

}Copy

### ****Q5. Which Program Elements Can Be Annotated?****

Annotations can be applied in several places throughout the source code. They can be applied to declarations of classes, constructors, and fields:

@SimpleAnnotation

**public** **class** **Apply** {

@SimpleAnnotation

**private** String aField;

@SimpleAnnotation

**public** **Apply**() {

// ...

}

}Copy

Methods and their parameters:

@SimpleAnnotation

**public** **void** **aMethod**(@SimpleAnnotation String param) {

// ...

}Copy

Local variables, including a loop and resource variables:

@SimpleAnnotation

**int** i = 10;

**for** (@SimpleAnnotation **int** j = 0; j < i; j++) {

// ...

}

**try** (@SimpleAnnotation **FileWriter** writer = getWriter()) {

// ...

} **catch** (Exception ex) {

// ...

}Copy

Other annotation types:

@SimpleAnnotation

**public** @interface ComplexAnnotation {

// ...

}Copy

And even packages, through the package-info.java file:

@PackageAnnotation

**package** com.baeldung.interview.annotations;Copy

As of Java 8, they can also be applied to the use of types. For this to work, the annotation must specify an @Target annotation with a value of ElementType.USE:

@Target(ElementType.TYPE\_USE)

**public** @interface SimpleAnnotation {

// ...

}Copy

Now, the annotation can be applied to class instance creation:

**new** @SimpleAnnotation Apply();Copy

Type casts:

aString = (@SimpleAnnotation String) something;Copy

Implements clause:

**public** **class** **SimpleList**<T>

implements @SimpleAnnotation List<@SimpleAnnotation T> {

// ...

}Copy

And throws clause:

**void** **aMethod**() **throws** @SimpleAnnotation Exception {

// ...

}Copy

### ****Q6. Is There a Way to Limit the Elements in Which an Annotation Can Be Applied?****

Yes, the @Target annotation can be used for this purpose. If we try to use an annotation in a context where it is not applicable, the compiler will issue an error.

Here’s an example to limit the usage of the @SimpleAnnotation annotation to field declarations only:

@Target(ElementType.FIELD)

**public** @interface SimpleAnnotation {

// ...

}Copy

We can pass multiple constants if we want to make it applicable in more contexts:

@Target({ ElementType.FIELD, ElementType.METHOD, ElementType.PACKAGE })Copy

We can even make an annotation so it cannot be used to annotate anything. This may come in handy when the declared types are intended solely for use as a member type in complex annotations:

@Target({})

**public** @interface NoTargetAnnotation {

// ...

}Copy

### ****Q7. What Are Meta-Annotations?****

Are annotations that apply to other annotations.

All annotations that aren’t marked with @Target, or are marked with it but include ANNOTATION\_TYPE constant are also meta-annotations:

@Target(ElementType.ANNOTATION\_TYPE)

**public** @interface SimpleAnnotation {

// ...

}Copy

### ****Q8. What Are Repeating Annotations?****

These are annotations that can be applied more than once to the same element declaration.

For compatibility reasons, since this feature was introduced in Java 8, repeating annotations are stored in a container annotation that is automatically generated by the Java compiler. For the compiler to do this, there are two steps to declared them.

First, we need to declare a repeatable annotation:

@Repeatable(Schedules.class)

**public** @interface Schedule {

String **time**() **default** "morning";

}Copy

Then, we define the containing annotation with a mandatory value element, and whose type must be an array of the repeatable annotation type:

**public** @interface Schedules {

Schedule[] value();

}Copy

Now, we can use @Schedule multiple times:

@Schedule

@Schedule(time = "afternoon")

@Schedule(time = "night")

**void** **scheduledMethod**() {

// ...

}Copy

### ****Q9. How Can You Retrieve Annotations? How Does This Relate to Its Retention Policy?****

You can use the Reflection API or an annotation processor to retrieve annotations.

The @Retention annotation and its RetentionPolicy parameter affect how you can retrieve them. There are three constants in RetentionPolicy enum:

* RetentionPolicy.SOURCE – makes the annotation to be discarded by the compiler but annotation processors can read them
* RetentionPolicy.CLASS – indicates that the annotation is added to the class file but not accessible through reflection
* RetentionPolicy.RUNTIME –Annotations are recorded in the class file by the compiler and retained by the JVM at runtime so that they can be read reflectively

Here’s an example code to create an annotation that can be read at runtime:

@Retention(RetentionPolicy.RUNTIME)

**public** @interface Description {

String **value**();

}Copy

Now, annotations can be retrieved through reflection:

**Description** description

= AnnotatedClass.class.getAnnotation(Description.class);

System.out.println(description.value());Copy

An annotation processor can work with RetentionPolicy.SOURCE, this is described in the article [Java Annotation Processing and Creating a Builder](https://www.baeldung.com/java-annotation-processing-builder).

RetentionPolicy.CLASS is usable when you’re writing a Java bytecode parser.

### ****Q10. Will the Following Code Compile?****

@Target({ ElementType.FIELD, ElementType.TYPE, ElementType.FIELD })

**public** @interface TestAnnotation {

**int**[] value() **default** {};

}Copy

No. It’s a compile-time error if the same enum constant appears more than once in an @Target annotation.

Removing the duplicate constant will make the code to compile successfully:

@Target({ ElementType.FIELD, ElementType.TYPE})Copy

### ****Q11. Is It Possible to Extend Annotations?****

No. Annotations always extend java.lang.annotation.Annotation, as stated in the [Java Language Specification](http://docs.oracle.com/javase/specs/jls/se7/html/jls-9.html#jls-9.6).

If we try to use the extends clause in an annotation declaration, we’ll get a compilation error:

**public** @interface AnAnnotation **extends** **OtherAnnotation** {

// Compilation error

}

* **Log4j**

### 1) Explain what is Log4j?

Log4j is a fast, flexible and reliable logging framework  written in Java developed in early 1996. It is distributed under the Apache software license and can be used for small to large scale projects.  It has been ported to the languages like C, C++, C#, Python, etc.

[Free PDF Download: Log4j Interview Questions and Answers](https://career.guru99.com/top-25-log4j-interview-questions/#cb7fd47d7b)

### 2) Explain why to use Apache Log4j?

* Being open-source its completely free to use.
* You can easily save log information into either files or even databases.
* Can be used for projects of any sizes small or large.

### 3) Mention what are the three principal components of Log4j?

The three principal components of Log4j are

* Loggers
* Appenders
* Layout

**List of the Most Frequently Asked Log4j Interview Questions:**

symbol

logo

00:09

03:59

EXPLORE MOREA white background with black text

Description automatically generatedTop 10 Behavioral Interview Questions and Answers04:00A white background with black text

Description automatically generatedTop 10 Interview Do's and Don'ts in a Job Interview02:11A white background with black text

Description automatically generatedTop 5 Powerpoint Interview Questions and Answers02:11A black text on a white background

Description automatically generatedWhat to bring to a JOB Interview: Interview Tips01:50A black text on a white background

Description automatically generatedWhat to wear in interview for Women | Working women01:44A black text on a white background

Description automatically generatedHow To Dress for an Job Interview for Men01:36A white background with black text

Description automatically generated10 Most Common Interview Questions and Answers 🔝03:54A cartoon of a person with his hand up

Description automatically generated5 Secrets Tips to Win an Interview01:42

### 4) Inside logger component what are the different log levels?

Different log levels inside logger components are

* All
* Debug
* Info
* Warn
* Error
* Fatal
* Off

### 5) Explain what are Appenders in Log4j?

Log4j Interview Questions

Appenders are used to deliver LogEvents to their destination. In simple words, it is used to write the logs in the file.

### 6) Mention what are the different types of Appenders?

Some of the Appenders type include

* ConsoleAppender logs to standard output
* FileAppender prints logs to some file
* Rolling file appender to a file with maximum size

### 7) Explain what is layouts in log4j?

Layout in log4j is responsible for formatting logging information in different styles.

### 8) Mention what are the two static methods for obtaining a logger object?

The two static methods for obtaining a logger object are

* Public static Logger getRootLogger()
* Public static Logger getLogger(String name)

### 9) Mention what are the other support objects in Log4j?

There are other support objects in Log4j framework they are

* Level Object
* Filter Object
* Object Renderer
* Log Manager

### 10) How log4j file is defined?

Log4j file is defined by the name log4j.properties, it keeps properties in key-value pairs. By default, the log manager looks for a file name log4j.properties in the **CLASSPATH.**

### 11) Explain what is the command to write your logging information into a file?

To write your logging information into a file, you would need to use a command **org.apache.log4j.FileAppender**

### 12) Mention what are the logging methods provided by logger class?

Logger class provides a variety of methods to handle logging activities.  To obtain a logger object it provides two static methods

* Public static logger getRootLogger();
* Public static logger getLogger(String name);

### 13) In log4j how can you log into the database?

The log4j [API](https://career.guru99.com/top-20-questions-on-api-testing/) provides the object **org.apache.log4j.**[**jdbc**](https://career.guru99.com/top-50-jdbc-interview-questions-and-answers/)**.**JDBCAppender object can put logging information in a particular database.

### 14) Explain whether log4j is a thread safe?

Log4j is a thread safe, log4j components are built to be used in multithread systems.

### 15) Explain whether a log output format can be customized?

Yes, you can extend the layout class to create your own customized log format. Appenders can be parameterized to use the layout of your choice.

### 16) Explain what are the system properties checked by log4j?

The system properties checked by log4j are

* **Log4j debug**, if true, log4j will show internal debugging messages to the console
* **defaultInitOverride**, if true, log4j will not execute default initialization
* **configuration**, URL for default initialization configuration file
* **configurationClass**, Class name for configurator to execute default initialization configuration file
* **ignoreTCL,**if true, the thread class loader will be overlooked when loading classes

### 17) Mention what is the role of filter in log4j?

Filter objects in log4j decide whether logging request should be handled by a particular Appender or ignored.

### 18) Explain how can you get multiple processes to log to the same file?

You may have each process log to a socket Appender. The receiving socket server can receive all the events and send them to a single log file.

### 19) Mention what is the difference between Threshold and LevelRangeFilter in log4j?

Both Threshold and LevelRangeFilter does the same thing. However threshold should be faster. Filters enable you to implement your own logic, and you can also link them together if required. If you need a basic threshold functionality, then “threshold” function will be enough.

### 20) Mention what does .class mean in log4j context?

In log4j context, .class is used to get the full name of your class and that string is used to configure this logger object.

**For example,**

logger.getlogget (Myclass.class)

### 21) Explain what is package level logging in log4j?

Package level logging is the standard logging of log4j, with this you would determine the package and the associated level.

### 22) Explain what does WARN and TRACE level indicates in log4j?

Log4j level WARN gives a warning about an unpredicted event to the user. The messages coming out of this level may not stop the progress of the system. The TRACE level provides more detailed information than the DEBUG level, and it will stay on the top of the hierarchy.

### 23) Explain what are the format characters used in log4j?

The format characters used in log4j are

* **L –** it is used to output the line number from where the logging request was processed or issued
* **m –** It is used to output the application supplied message related with the logging event
* **p –** It is used to output the priority of the logging event
* **C –** It is used to output the class name of the caller issuing the logging request

When any number is used along with the character it means the priority of the logging event should be justified to a width of 4 characters.

### 24) Mention what is the best way to migrate from java.util logging to log4j?

The best way to migrate from java.util  logged to log4j is to use global file search/replace method.  It will replace with “org.apache.log4j.Logger”

### 25) Explain why do you get multiple copies of the message in log file sometime?

There could be two reasons why this may happen

* Repeated configuration of log4j
* Attaching the same appenders to multiple loggers

**What is the use of volatile keywords?**

Volatile keyword is used to modify the value of a variable by different threads. It is also used to make classes thread safe.

**Can we make array volatile in Java?**

Yes, you can make an array volatile in Java but only the reference, which is pointing to an array, not the whole array.   
  
**Can volatile make a non-atomic operation to atomic?**

 volatile only makes read/write operations atomic, which means that one thread can change a volatile variable while another thread is reading it. Therefore, volatile variables are only used in simple scenarios where read/write operations need to be atomic.  
  
**What are practical uses of volatile modifier?**  
One of the practical uses of the volatile variable is to make reading double and long atomic. Both double and long are 64-bit wide and they are read in two parts, first 32-bit first time and next 32-bit second time, which is non-atomic but volatile double and long read is atomic in Java.

Another use of the volatile variable is to provide a memory barrier, just like it is used in Disruptor framework. Basically, Java Memory model inserts a write barrier after you write to a volatile variable and a read barrier before you read it.

Which means, if you write to volatile field then it's guaranteed that any thread accessing that variable will see the value you wrote and anything you did before doing that right into the thread is guaranteed to have happened and any updated data values will also be visible to all threads, because the memory barrier flushed all other writes to the cache.  
  
  
**4) What guarantee volatile variable provides?**([answer](http://java67.blogspot.sg/2012/08/what-is-volatile-variable-in-java-when.html))  
volatile variables provide the guarantee about ordering and visibility e.g. volatile assignment cannot be re-ordered with other statements but in the absence of any synchronization instruction compiler, JVM or JIT are free to reorder statements for better performance. volatile also provides the happens-before guarantee which ensures changes made in one thread is visible to others.

In some cases volatile also provide atomicity e.g. reading 64-bit data types like long and double are not atomic but read of volatile double or long is atomic.

**5) Which one would be easy to write? synchronization code for 10 threads or 2 threads?**  
In terms of writing code, both will be of same complexity because synchronization code is independent of a number of threads. Choice of synchronization though depends upon a number of threads because the number of thread present more contention, so you go for advanced synchronization technique e.g. lock stripping, which requires more complex code and expertise.  
  
  
**6) How do you call wait() method? using if block or loop? Why?**([answer](http://javarevisited.blogspot.sg/2015/07/how-to-use-wait-notify-and-notifyall-in.html))  
wait() method should always be called in loop because it's possible that until thread gets CPU to start running again the condition might not hold, so it's always better to check condition in loop before proceeding.

Here is the standard idiom of using wait and notify method in Java:

// The standard idiom for using the wait method

synchronized (obj) {

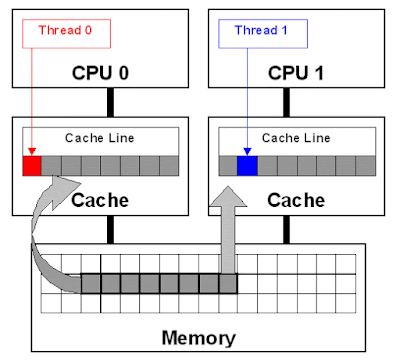
while (condition does not hold)

obj.wait(); // (Releases lock, and reacquires on wakeup)

... // Perform action appropriate to condition

}

See [Effective Java Item 69](https://javarevisited.blogspot.com/2017/10/effective-java-3rd-edition-coming-soon.html#axzz5Nf1J69kw) to learn more about why wait method should call in the loop.  
  
  
**7)  What is false sharing in the context of multi-threading?**  
false sharing is one of the well-known performance issues on multi-core systems, where each process has its local cache. false sharing occurs when threads on different processor modify variables that reside on same cache line as shown in the following image:

[](https://javarevisited.blogspot.com/2020/04/top-10-advanced-core-java-courses-for-experienced-developers.html)

False sharing is very hard to detect because the thread may be accessing completely different global variables that happen to be relatively close together in memory. Like many concurrency issues, the primary way to avoid false sharing is careful code review and aligning your data structure with the size of a cache line.  
  
  
**8) What is busy spin? Why should you use it?**  
Busy spin is one of the technique to wait for events without releasing CPU. It's often done to avoid losing data in CPU cached which is lost if the thread is paused and resumed in some other core. So, if you are working on low latency system where your order processing thread currently doesn't have any order, instead of sleeping or calling wait(), you can just loop and then again check the queue for new messages.

It's only beneficial if you need to wait for a very small amount of time e.g. in microseconds or nanoseconds. [LMAX Disrupter](http://lmax-exchange.github.io/disruptor/) framework, a high-performance inter-thread messaging library has a BusySpinWaitStrategy which is based on this concept and uses a busy spin loop for EventProcessors waiting on the barrier.  
  
  
**9) How do you take thread dump in Java?**  
You can take a thread dump of Java application in Linux by using **kill -3 PID**, where PID is the process id of Java process. In Windows, you can press **Ctrl + Break**. This will instruct JVM to print thread dump in standard out or err and it could go to console or log file depending upon your application configuration. If you have used Tomcat then when  
  
  
  
**10) is Swing thread-safe?**([answer](http://javarevisited.blogspot.sg/2013/08/why-swing-is-not-thread-safe-in-java-Swingworker-Event-thread.html))  
No, Swing is not thread-safe. You cannot update Swing components e.g. JTable, JList or JPanel from any thread, in fact, they must be updated from GUI or AWT thread. That's why swings provide invokeAndWait() and invokeLater() method to request GUI update from any other threads.

This methods put update request in AWT threads queue and can wait till update or return immediately for an asynchronous update. You can also check the detailed answer to learn more.  
  
  
**11) What is a thread local variable in Java?** ([answer](http://javarevisited.blogspot.sg/2012/05/how-to-use-threadlocal-in-java-benefits.html))  
Thread-local variables are variables confined to a thread, its like thread's own copy which is not shared between multiple threads. Java provides a ThreadLocal class to support thread-local variables. It's one of the many ways to achieve thread-safety.

Though be careful while using thread local variable in managed environment e.g. with web servers where worker thread out lives any application variable. Any thread local variable which is not removed once its work is done can potentially cause a memory leak in Java application.  
  
  
**12) Write wait-notify code for producer-consumer problem?** ([answer](http://java67.blogspot.sg/2012/12/producer-consumer-problem-with-wait-and-notify-example.html))  
Please see the answer for a code example. Just remember to call wait() and notify() method from synchronized block and test waiting for condition on the loop instead of if block.  
  
  
**13) Write code for thread-safe Singleton in Java?** ([answer](http://javarevisited.blogspot.in/2012/12/how-to-create-thread-safe-singleton-in-java-example.html))  
Please see the answer for a code example and step by step guide to creating thread-safe singleton class in Java. When we say thread-safe, which means Singleton should remain singleton even if initialization occurs in the case of multiple threads. Using Java enum as Singleton class is one of the easiest ways to create a thread-safe singleton in Java.  
  
  
**14) The difference between sleep and wait in Java?**([answer](http://java67.blogspot.sg/2012/08/what-are-difference-between-wait-and.html))  
Though both are used to pause currently running thread, sleep() is actually meant for short pause because it doesn't release lock, while wait() is meant for conditional wait and that's why it release lock which can then be acquired by another thread to change the condition on which it is waiting.  
  
  
**15) What is an immutable object? How do you create an Immutable object in Java?** ([answer](http://javarevisited.blogspot.sg/2013/03/how-to-create-immutable-class-object-java-example-tutorial.html))  
Immutable objects are those whose state cannot be changed once created. Any modification will result in a new object e.g. String, Integer, and other wrapper class. Please see the answer for step by step guide to creating Immutable class in Java.  
  
  
**16) Can we create an Immutable object, which contains a mutable object?**  
Yes, its possible to create an Immutable object which may contain a mutable object, you just need to be a little bit careful not to share the reference of the mutable component, instead, you should return a copy of it if you have to. Most common example is an Object which contain the reference of java.util.Date object.

## Date types and Basic Java Interview Questions

Now, let's take a look at Java Interview questions which are based upon core Java and basic Java concepts like data types int, float, long, double and conditional and logical operator including the ternary operator

**17) What is the right data type to represent a price in Java?**([answer](http://javarevisited.blogspot.sg/2012/02/java-mistake-1-using-float-and-double.html))  
BigDecimal if memory is not a concern and Performance is not critical, otherwise double with predefined precision.  
  
  
**18) How do you convert bytes to String?** ([answer](http://javarevisited.blogspot.sg/2014/08/2-examples-to-convert-byte-array-to-String-in-Java.html))  
you can convert bytes to the string using string constructor which accepts byte[], just make sure that right character encoding otherwise platform's default character encoding will be used which may or may not be same.  
  
  
**19) How do you convert bytes to long in Java?** (answer)  
This questions if for you to answer :-)  
  
  
**20) Can we cast an int value into byte variable? what will happen if the value of int is larger than byte?**  
Yes, we can cast but int is 32 bit long in java while byte is 8 bit long in java so when you cast an int to byte higher 24 bits are lost and a byte can only hold a value from -128 to 128.  
  
  
**21) There are two classes B extends A and C extends B, Can we cast B into C e.g. C = (C) B;**([answer](http://javarevisited.blogspot.sg/2012/12/what-is-type-casting-in-java-class-interface-example.html))  
  
  
**22) Which class contains clone method? Cloneable or Object?** ([answer](http://javarevisited.blogspot.sg/2015/01/java-clone-tutorial-part-2-overriding-with-mutable-field-example.html))  
java.lang.Cloneable is marker interface and doesn't contain any method clone method is defined in the object class. It is also knowing that clone() is a native method means it's implemented in C or C++ or any other native language.  
  
  
**23) Is ++ operator is thread-safe in Java?** (answer)  
 No it's not a thread safe operator because its involve multiple instructions like reading a value, incriminating it and storing it back into memory which can be overlapped between multiple threads.  
  
  
**24) Difference between a = a + b and a += b ?** (answer)  
The += operator implicitly cast the result of addition into the type of variable used to hold the result. When you add two integral variable e.g. variable of type byte, short, or int then they are first promoted to int and them addition happens. If result of addition is more than maximum value of a then a + b will give compile time error but a += b will be ok as shown below

byte a = 127;

byte b = 127;

b = a + b; *// error : cannot convert from int to byte*

b += a; *// ok*

**25) Can I store a double value in a long variable without casting?**([answer](http://java67.blogspot.com/2014/11/how-to-convert-double-to-long-in-java-example.html))  
No, you cannot store a double value into a long variable without casting because the range of double is more  that long and you we need to type cast. It's not dificult to answer this question but many develoepr get it wrong due to confusion on which one is bigger between double and long in Java.  
  
  
**26) What will this return 3\*0.1 == 0.3? true or false?**(answer)  
This is one of the really tricky questions. Out of 100, only 5 developers answered this question and only of them have explained the concept correctly. The short answer is false because some floating point numbers can not be represented exactly.  
  
  
**27) Which one will take more memory, an int or Integer?**(answer)  
An Integer object will take more memory an Integer is the an object and it  store meta data overhead about the object and int is primitive type so its takes less space.  
  
  
**28) Why is String Immutable in Java?** ([answer](http://java67.blogspot.sg/2014/01/why-string-class-has-made-immutable-or-final-java.html))  
One of my favorite Java interview question. The String is Immutable in java because java designer thought that string will be heavily used and making it immutable allow some optimization easy sharing same String object between multiple clients.

See the link for the more detailed answer. This is a great question for Java programmers with less experience as it gives them food for thought, to think about how things works in Java, what Jave designers might have thought when they created String class etc.  
  
**29) Can we use String in the switch case?** ([answer](http://javarevisited.blogspot.sg/2011/08/string-switch-case-jdk7-example.html))  
Yes from Java 7 onward we can use String in switch case but it is just syntactic sugar. Internally string hash code is used for the switch. See the detailed answer for more explanation and discussion.  
  
**30) What is constructor chaining in Java?** ([answer](http://java67.blogspot.sg/2012/12/how-constructor-chaining-works-in-java.html))  
When you call one constructor from other than it's known as constructor chaining in Java. This happens when you have multiple, overloaded constructor in the class.

### JVM Internals and Garbage Collection Interview Questions

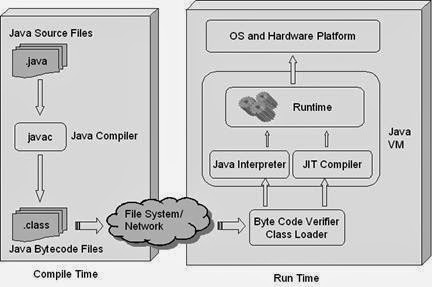
In last a couple of years I have seen increased focus on JVM internal and Garbage collection tuning, monitoring Java application, dealing with Java performance issues on various Java interviews. This is actually become the prime topic for interviewing any experienced Java developer for senior position e.g. technical lead, VP or team lead.

If you feel you are short of experience and knowledge in this area then you should read at least one book mentioned in my list of [Java Performance books](http://javarevisited.blogspot.com/2014/07/top-5-java-performance-tuning-books.html). I vote goes to Java Performance, The Definitive guide by Scott.  
  
**31) What is the size of int in 64-bit JVM?**  
The size of an int variable is constant in Java, it's always 32-bit irrespective of platform. Which means the size of primitive int is same in both 32-bit and 64-bit Java virtual machine.  
  
**32) The difference between Serial and Parallel Garbage Collector?** ([answer](http://javarevisited.blogspot.sg/2011/04/garbage-collection-in-java.html))  
Even though both the serial and parallel collectors cause a stop-the-world pause during Garbage collection. The main difference between them is that a serial collector is a default copying collector which uses only one GC thread for garbage collection while a parallel collector uses multiple GC threads for garbage collection.  
  
**33) What is the size of an int variable in 32-bit and 64-bit JVM?**(answer)  
The size of int is same in both 32-bit and 64-bit JVM, it's always 32 bits or 4 bytes.  
  
**34) A difference between WeakReference and SoftReference in Java?**([answer](http://javarevisited.blogspot.sg/2014/03/difference-between-weakreference-vs-softreference-phantom-strong-reference-java.html))  
Though both WeakReference and SoftReference helps garbage collector and memory efficient, WeakReference becomes eligible for garbage collection as soon as last strong reference is lost but SoftReference even thought it can not prevent GC, it can delay it until JVM absolutely need memory.  
  
**35) How do WeakHashMap works?**(answer)  
WeakHashMap works like a normal HashMap but uses WeakReference for keys, which means if the key object doesn't have any reference then both key/value mapping will become eligible for garbage collection.  
  
**36) What is -XX:+UseCompressedOops JVM option? Why use it?**([answer](http://javarevisited.blogspot.com/2012/06/what-is-xxusecompressedoops-in-64-bit.html))  
When you go migrate your Java application from 32-bit to 64-bit JVM, the heap requirement suddenly increases, almost double, due to increasing size of ordinary object pointer from 32 bit to 64 bit.

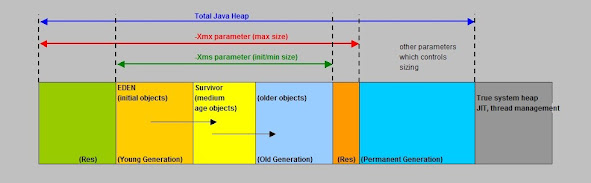
This also adversely affect how much data you can keep in CPU cache, which is much smaller than memory. Since main motivation for moving to 64-bit JVM is to specify large heap size, you can save some memory by using compressed OOP. By using -XX:+UseCompressedOops, JVM uses 32-bit OOP instead of 64-bit OOP.  
  
  
**37) How do you find if JVM is 32-bit or 64-bit from Java Program?**([answer](http://javarevisited.blogspot.sg/2012/01/find-jvm-is-32-or-64-bit-java-program.html))  
You can find that by checking some system properties like sun.arch.data.model or os.arch  
  
  
**38) What is the maximum heap size of 32-bit and 64-bit JVM?**([answer](http://javarevisited.blogspot.sg/2013/04/what-is-maximum-heap-size-for-32-bit-64-JVM-Java-memory.html))  
Theoretically, the maximum heap memory you can assign to a 32-bit JVM is 2^32 which is 4GB but practically the limit is much smaller. It also varies between operating systems e.g. form 1.5GB in Windows to almost 3GB in Solaris.

64-bit JVM allows you to specify larger heap size, theoretically 2^64 which is quite large but practically you can specify heap space up to 100GBs. There are even JVM e.g. Azul where heap space of 1000 gigs is also possible.  
  
  
**39) What is the difference between JRE, JDK, JVM and JIT?** ([answer](http://javarevisited.blogspot.sg/2011/12/jre-jvm-jdk-jit-in-java-programming.html))  
JRE stands for Java run-time and it's required to run Java application. JDK stands for Java development kit and provides tools to develop Java program e.g. Java compiler. It also contains JRE. The JVM stands for Java virtual machine and it's the process responsible for running Java application.

The JIT stands for Just In Time compilation and helps to boost the performance of Java application by converting Java byte code into native code when the crossed certain threshold i.e. mainly hot code is converted into native code.

[](https://javarevisited.blogspot.com/2018/07/top-5-websites-to-learn-coding-in-java.html)

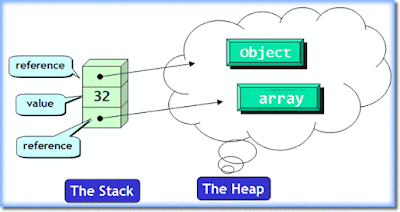
**40) Explain Java Heap space and Garbage collection?** ([answer](http://javarevisited.blogspot.sg/2011/05/java-heap-space-memory-size-jvm.html))  
When a Java process is started using java command, memory is allocated to it. Part of this memory is used to create heap space, which is used to allocate memory to objects whenever they are created in the program. Garbage collection is the process inside JVM which reclaims memory from dead objects for future allocation.

[](https://javarevisited.blogspot.com/2019/04/top-5-courses-to-learn-jvm-internals.html)

**41) Can you guarantee the garbage collection process?**(answer)  
No, you cannot guarantee the garbage collection, though you can make a request using System.gc() or Runtime.gc() method.  
  
  
**42) How do you find memory usage from Java program? How much percent of the heap is used?**  
You can use memory related methods from java.lang.Runtime class to get the free memory, total memory and maximum heap memory in Java.

By using these methods, you can find out how many percents of the heap is used and how much heap space is remaining. Runtime.freeMemory() return amount of free memory in bytes, Runtime.totalMemory() returns total memory in bytes and Runtime.maxMemory() returns maximum memory in bytes.  
  
  
**43) What is the difference between stack and heap in Java?**([answer](http://javarevisited.blogspot.com/2013/01/difference-between-stack-and-heap-java.html))  
Stack and heap are different memory areas in the JVM and they are used for different purposes. The stack is used to hold method frames and local variables while objects are always allocated memory from the heap.

The stack is usually much smaller than heap memory and also didn't shared between multiple threads, but heap is shared among all threads in JVM.

[](https://medium.com/javarevisited/top-5-java-online-courses-for-beginners-best-of-lot-1e1e240a758)

## Basic Java concepts Interview Questions

In this part, we will take a look at various Java keywords, and concepts like equals and hashcode, compile time and runtime behavior and much which are very important for beginner level Java interviews.

**44) What's the difference between "a == b" and "a.equals(b)"?** ([answer](http://javarevisited.blogspot.sg/2012/12/difference-between-equals-method-and-equality-operator-java.html))  
The a = b does object reference matching if both a and b are an object and only return true if both are pointing to the same object in the heap space, on the other hand, a.equals(b) is used for logical mapping and its expected from an object to override this method to provide logical equality.

For example, String class overrides this equals() method so that you can compare two Strings, which are the different object but contains same letters.  
  
  
**45) What is a.hashCode() used for? How is it related to a.equals(b)?**([answer](http://javarevisited.blogspot.sg/2011/10/override-hashcode-in-java-example.html))  
hashCode() method returns an int hash value corresponding to an object. It's used in hash based collection classes e.g Hashtable, HashMap, LinkedHashMap and so on. It's very tightly related to equals() method. According to Java specification, two objects which are equal to each other using equals() method must have same hash code.  
  
  
**46) Difference between final, finalize and finally?**([answer](http://javarevisited.blogspot.sg/2012/11/difference-between-final-finally-and-finalize-java.html))  
The final is a modifier which you can apply to variable, methods and classes. If you make a variable final it means its value cannot be changed once initialized. finalize is a method, which is called just before an object is a garbage collected, giving it last chance to resurrect itself, but the call to finalize is not guaranteed.

finally is a keyword which is used in exception handling along with try and catch. the finally block is always executed irrespective of whether an exception is thrown from try block or not.  
  
  
**47) What is a compile time constant in Java? What is the risk of using it?**  
public static final variables are also known as a compile time constant, the public is optional there. They are replaced with actual values at compile time because compiler know their value up-front and also knows that it cannot be changed during run-time.

One of the problem with this is that if you happened to use a public static final variable from some in-house or third party library and their value changed later than your client will still be using old value even after you deploy a new version of JARs. To avoid that, make sure you compile your program when you upgrade dependency JAR files.

## Java Collections Framework Interview Questions

It also contains Data structure and algorithm Interview question in Java, questions on array, linked list, HashMap, ArrayList, Hashtable, Stack, Queue, PriorityQueue, LinkedHashMap and ConcurrentHashMap.  
  
**48) The difference between List, Set, Map, and Queue in Java?** ([answer](http://java67.blogspot.sg/2013/01/difference-between-set-list-and-map-in-java.html))  
The list is an ordered collection which allows duplicate. It also has an implementation which provides constant time index based access, but that is not guaranteed by List interface. Set is unordered collection which  
  
  
**49) Difference between poll() and remove() method?**  
Both poll() and remove() take out the object from the Queue but if poll() fails then it returns null but if remove fails it throws Exception.  
  
  
**50) The difference between LinkedHashMap and PriorityQueue in Java?**([answer](http://javarevisited.blogspot.sg/2013/10/what-is-priorityqueue-data-structure-java-example-tutorial.html))  
PriorityQueue guarantees that lowest or highest priority element always remain at the head of the queue, but LinkedHashMap maintains the order on which elements are inserted. When you iterate over a PriorityQueue, iterator doesn't guarantee any order but iterator of LinkedHashMap does guarantee the order on which elements are inserted.  
  
  
**51) Difference between ArrayList and LinkedList in Java?** ([answer](http://java67.blogspot.sg/2012/12/difference-between-arraylist-vs-LinkedList-java.html))  
The obvious difference between them is that ArrayList is backed by array data structure, supports random access and LinkedList is backed by linked list data structure and doesn't support random access. Accessing an element with the index is O(1) in ArrayList but its O(n) in LinkedList. See the answer for more detailed discussion.  
  
  
**52) What is a couple of ways that you could sort a collection?** ([answer](http://java67.blogspot.sg/2012/07/sort-list-ascending-descending-order-set-arraylist.html))  
You can either use the Sorted collection like TreeSet or TreeMap or you can sort using the ordered collection like a list and using Collections.sort() method.  
  
  
**53) How do you print Array in Java?** ([answer](http://java67.blogspot.sg/2014/03/how-to-print-array-in-java-example-tutorial.html))  
You can print an array by using the Arrays.toString() and Arrays.deepToString() method. Since array doesn't implement toString() by itself, just passing an array to System.out.println() will not print its contents but Arrays.toString() will print each element.

**54) LinkedList in Java is doubly or singly linked list?** (answer)  
It's a doubly linked list, you can check the code in JDK. In Eclipse, you can use the [shortcut](http://javarevisited.blogspot.com/2010/10/eclipse-tutorial-most-useful-eclipse.html), Ctrl + T to directly open this class in Editor.  
  
**55) Which kind of tree is used to implement TreeMap in Java?** (answer)  
A Red Black tree is used to implement TreeMap in Java.

**56) What is the difference between Hashtable and HashMap?**([answer](http://java67.blogspot.sg/2012/08/5-difference-between-hashtable-hashmap-Java-collection.html))  
There are many differences between these two classes, some of them are following:  
a) Hashtable is a legacy class and present from JDK 1, HashMap was added later.  
b) Hashtable is synchronized and slower but HashMap is not synchronized and faster.  
c) Hashtable doesn't allow null keys but HashMap allows one null key.  
See the answer for more differences between HashMap and Hashtable in Java.  
  
  
**57) How HashSet works internally in Java?** ([answer](http://java67.blogspot.sg/2014/01/how-hashset-is-implemented-or-works-internally-java.html))  
HashSet is internally implemented using an HashMap. Since a Map needs key and value, a default value is used for all keys. Similar to HashMap, HashSet doesn't allow duplicate keys and only one null key, I mean you can only store one null object in HashSet.  
  
  
**58) Write code to remove elements from ArrayList while iterating?** ([answer](http://javarevisited.blogspot.sg/2014/01/ow-to-remove-objects-from-collection-arraylist-java-iterator-traversing.html))  
 Key here is to check whether candidate uses ArrayList's remove() or Iterator's remove(). Here is the [sample code](http://java67.blogspot.com/2015/10/how-to-solve-concurrentmodificationexception-in-java-arraylist.html) which uses right way o remove elements from ArrayList while looping over and avoids ConcurrentModificationException.  
  
  
**59) Can I write my own container class and use it in the for-each loop?**  
Yes, you can write your own container class. You need to implement the Iterable interface if you want to loop over advanced for loop in Java, though. If you implement Collection then you by default get that property.  
  
  
**60) What is default size of ArrayList and HashMap in Java?**([answer](http://javarevisited.blogspot.sg/2014/07/java-optimization-empty-arraylist-and-Hashmap-cost-less-memory-jdk-17040-update.html))  
As of Java 7 now, default size of ArrayList is 10 and default capacity of HashMap is 16, it must be power of 2. Here is code snippet from ArrayList  and HashMap class :

// from ArrayList.java JDK 1.7

private static final int DEFAULT\_CAPACITY = 10;

//from HashMap.java JDK 7

static final int DEFAULT\_INITIAL\_CAPACITY = 1 **<<** 4; // aka 16

**61) Is it possible for two unequal objects to have the same hashcode?**  
Yes, two unequal objects can have same hashcode that's why collision happen in a hashmap.  
the equal hashcode contract only says that two equal objects must have the same hashcode it doesn't say anything about the unequal object.  
  
**62) Can two equal object have the different hash code?**  
No, thats not possible according to hash code contract.  
  
  
**63) Can we use random numbers in the hashcode() method?** ([answer](http://javarevisited.blogspot.sg/2011/10/override-hashcode-in-java-example.html))  
No, because hashcode of an object should be always same. See the answer to learning more about things to remember while overriding hashCode() method in Java.  
  
  
**64) What is the difference between Comparator and Comparable in Java?**([answer](http://java67.blogspot.sg/2013/08/difference-between-comparator-and-comparable-in-java-interface-sorting.html))  
The Comparable interface is used to define the  natural order of object while Comparator is used to define custom order. Comparable can be always one, but we can have multiple comparators to define customized order for objects.  
  
**65) Why you need to override hashcode, when you override equals in Java?** ([answer](http://javarevisited.blogspot.sg/2015/01/why-override-equals-hashcode-or-tostring-java.html))  
 Because equals have code contract mandates to override equals and hashcode together .since many container class like HashMap or HashSet depends on hashcode and equals contract.

### Java IO and NIO Interview questions

IO is very important from Java interview point of view. You should have a good knowledge of old Java IO, NIO, and NIO2 alsong with some operating system and disk IO fundamentals. Here are some frequently asked questions form Java IO.  
  
**66) In my Java program, I have three sockets? How many threads I will need to handle that?**

The number of threads needed to handle three sockets in a Java program depends on the specific requirements and design of your application. Generally, a common approach is to use a multi-threaded model where each socket connection is handled by a separate thread.

This allows concurrent processing of multiple connections, preventing one slow or blocking operation from affecting others. Therefore, for three sockets, you might create three threads, each responsible for managing a single socket connection.

It's also essential to implement proper synchronization mechanisms if shared resources are accessed among these threads to avoid potential race conditions and ensure thread safety.

However, the optimal threading strategy can vary based on factors such as the nature of the tasks performed during socket handling, the expected number of concurrent connections, and the overall architecture of your application.

**67) How do you create ByteBuffer in Java? (**[**example**](https://javarevisited.blogspot.com/2020/03/bytebuffer-read-write-example-in-java.html)**)**

In Java, you can create a ByteBuffer using the ByteBuffer.allocate() method or ByteBuffer.allocateDirect() method. The allocate() method creates a non-direct buffer in the Java heap, while allocateDirect() creates a direct buffer outside of the Java heap

Here is the sample code:

// Creating a non-direct ByteBuffer with a specified capacity

int capacity = 1024;

ByteBuffer buffer = ByteBuffer.allocate(capacity);

// Creating a direct ByteBuffer with a specified capacity

ByteBuffer directBuffer = ByteBuffer.allocateDirect(capacity);

After creating a ByteBuffer, you can perform various operations such as reading and writing data, flipping, and compacting, depending on your specific use case.

**68) How do you write and read from ByteBuffer in Java?**

In Java, writing to and reading from a ByteBuffer involves using the put() methods to write data and the get() methods to read data. For instance, you can use put(byte), putInt(int), or putDouble(double) to write various data types into the buffer.

After writing, **it's essential to flip the buffer using the flip() method to prepare it for reading**.

Reading data involves using corresponding get() methods to retrieve the data from the buffer. Additionally, managing the buffer's position and limit is crucial; the position indicates the current read/write position, and the limit marks the end of valid data.

Depending on your needs, you may need to compact the buffer to remove read data or clear it to start writing from the beginning. Exception handling, such as handling BufferOverflowException and BufferUnderflowException, is important when dealing with dynamic data sizes to ensure robustness in your application.  
  
69) Is Java BIG endian or LITTLE endian?  
  
70) What is the byte order of ByteBuffer?  
  
71) The difference between direct buffer and non-direct buffer in Java? ([answer](http://javarevisited.blogspot.sg/2015/08/difference-between-direct-non-direct-mapped-bytebuffer-nio-java.html))  
  
72) What is the memory mapped buffer in Java? ([answer](http://javarevisited.blogspot.sg/2012/01/memorymapped-file-and-io-in-java.html))  
  
73) What is TCP NO DELAY socket option?  
  
74) What is the difference between TCP and UDP protocol? ([answer](http://javarevisited.blogspot.com/2014/07/9-difference-between-tcp-and-udp-protocol.html))  
  
75) The difference between ByteBuffer and StringBuffer in Java? (answer)

### Java Best Practices Interview question

Contains best practices from different parts of Java programming e.g. Collections, String, IO, Multi-threading, Error and Exception handling, design patterns etc. This section is mostly for experience Java developer, technical lead,  AVP, team lead and coders who are responsible for products. If you want to create quality products you must know and follow the best practices.  
  
**76) What best practices you follow while writing multi-threaded code in Java?** ([answer](http://javarevisited.blogspot.com/2015/05/top-10-java-multithreading-and.html))  
Here are couple of best practices which I follow while writing concurrent code in Java:  
a) Always name your thread, this will help in debugging.  
b) minimize the scope of synchronization, instead of making whole method synchronized, only critical section should be synchronized.  
c) prefer volatile over synchronized if you can can.  
e) use higher level concurrency utilities instead of wait() and notify for inter thread communication e.g. BlockingQueue, CountDownLatch and Semaphore.  
e) Prefer concurrent collection over synchronized collection in Java. They provide better scalability.  
  
  
**77) Tell me few best practices you apply while using Collections in Java?**(answer)  
Here are couple of best practices I follow while using Collection classes from Java:  
a) Always use the right collection e.g. if you need non-synchronized list then use ArrayList and not Vector.  
b) Prefer concurrent collection over synchronized collection because they are more scalable.  
c) Always use interface to a represent and access a collection e.g. use List to store ArrayList, Map to store HashMap and so on.  
d) Use iterator to loop over collection.  
e) Always use generics with collection.  
  
  
**78) Can you tell us at least 5 best practice you use while using threads in Java?** ([answer](http://java67.blogspot.com/2014/01/10-points-about-thread-and-javalangthread-in-java.html))  
This is similar to the previous question and you can use the answer given there. Particularly with thread, you should:  
a) name your thread  
b) keep your task and thread separate, use Runnable or Callable with thread pool executor.  
c) use thread pool  
d) use volatile to indicate compiler about ordering, visibility, and atomicity.  
e) avoid thread local variable because incorrect use of ThreadLocal class in Java can create a memory leak.  
Look there are many best practices and I give extra points to the developer which bring something new, something even I don't know. I make sure to ask this question to Java developers of 8 to 10 years of experience just to gauge his hands on experience and knowledge.  
  
  
**79) Name 5 IO best practices?** (answer)  
IO is very important for performance of your Java application. Ideally you should avoid IO in critical path of your application. Here are couple of Java IO best practices you can follow:

a) Use buffered IO classes instead of reading individual bytes and char.

b) Use classes from NIO and NIO2

c) Always close streams in finally block or use try-with-resource statements.

d) use memory mapped file for faster IO.

If a Java candidate doesn't know about IO and NIO, especially if he has at least 2 to 4 years of experience, he needs some reading.  
  
  
**80) Name 5 JDBC best practices your follow?** ([answer](http://javarevisited.blogspot.sg/2012/08/top-10-jdbc-best-practices-for-java.html))  
Another good Java best practices for experienced Java developer of 7 to 8 years experience. Why it's important? because they are the ones which set the trend in the code and educate junior developers. There are many best practices and you can name as per your comfort and convenience. Here are some of the more common ones:  
a) use batch statement for inserting and updating data.  
b) use PreparedStatement to avoid SQL exception and better performance.  
c) use database connection pool  
d) access resultset using column name instead of column indexes.  
e) Don't generate dynamic SQL by concatenating String with user input.  
  
  
**81) Name couple of method overloading best practices in Java?** ([answer](http://javarevisited.blogspot.sg/2013/01/java-best-practices-method-overloading-constructor.html))  
Here are some best practices you can follow while overloading a method in Java to avoid confusion with auto-boxing:  
a) Don't overload method where one accepts int and other accepts Integer.  
b) Don't overload method where number of argument is same and only order of argument is different.  
c) Use varargs after overloaded methods has more than 5 arguments.

### Date, Time and Calendar Interview questions in Java

**82) Does SimpleDateFormat is safe to use in the multi-threaded program?** ([answer](http://javarevisited.blogspot.sg/2012/03/simpledateformat-in-java-is-not-thread.html))  
No, unfortunately, DateFormat and all its implementations including SimpleDateFormat is not thread-safe, hence should not be used in the multi-threaded program until external thread-safety measures are applied e.g. confining SimpleDateFormat object into a ThreadLocal variable.

If you don't do that, you will get an incorrect result while parsing or formatting dates in Java. Though, for all practical date time purpose, I highly recommend **joda-time** library.  
  
  
**83) How do you format a date in Java? e.g. in the ddMMyyyy format?** ([answer](http://javarevisited.blogspot.com/2011/09/convert-date-to-string-simpledateformat.html))  
You can either use SimpleDateFormat class or joda-time library to format date in Java. DateFormat class allows you to format date on many popular formats. Please see the answer for code samples to format date into different formats e.g. dd-MM-yyyy or ddMMyyyy.  
  
  
84) How do you show timezone in formatted date in Java? ([answer](http://java67.blogspot.sg/2013/01/how-to-format-date-in-java-simpledateformat-example.html))  
  
85) The difference between java.util.Date and java.sql.Date in Java? ([answer](http://java67.blogspot.sg/2014/02/how-to-convert-javautildate-to-javasqldate-example.html))  
  
86) How to you calculate the difference between two dates in Java? ([program](http://javarevisited.blogspot.sg/2015/07/how-to-find-number-of-days-between-two-dates-in-java.html))  
  
87) How do you convert a String(YYYYMMDD) to date in Java? ([answer](http://java67.blogspot.sg/2014/12/string-to-date-example-in-java-multithreading.html))

### Unit testing JUnit Interview questions

89) How do you test static method? (answer)  
You can use PowerMock library to test static methods in Java.  
  
90) How to do you test a method for an exception using JUnit? ([answer](http://javarevisited.blogspot.sg/2013/04/JUnit-tutorial-example-test-exception-thrown-by-java-method.html))  
  
91) Which unit testing libraries you have used for testing Java programs? (answer)  
  
92) What is the difference between @Before and @BeforeClass annotation? ([answer](http://javarevisited.blogspot.sg/2013/04/JUnit-tutorial-example-test-exception-thrown-by-java-method.html))

### Programming and Coding Questions

93) How to check if a String contains only numeric digits? ([solution](http://java67.blogspot.com/2014/01/java-regular-expression-to-check-numbers-in-String.html))  
  
94) How to write LRU cache in Java using Generics? (answer)  
  
95) Write a Java program to convert bytes to long? (answer)  
  
96) How to reverse a String in Java without using StringBuffer? ([solution](http://java67.blogspot.com/2012/12/how-to-reverse-string-in-java-stringbuffer-stringbuilder.htm))  
  
97) How to find the word with the highest frequency from a file in Java? ([solution](http://java67.blogspot.com/2015/10/java-program-to-find-repeated-words-and-count.html))  
  
98) How do you check if two given String are anagrams? ([solution](http://javarevisited.blogspot.sg/2013/03/Anagram-how-to-check-if-two-string-are-anagrams-example-tutorial.html))  
  
99) How to print all permutation of a String in Java? ([solution](http://javarevisited.blogspot.com/2015/08/how-to-find-all-permutations-of-string-java-example.html))  
  
100) How do you print duplicate elements from an array in Java? ([solution](http://javarevisited.blogspot.com/2015/06/3-ways-to-find-duplicate-elements-in-array-java.html))  
  
101) How to convert String to int in Java? ([solution](http://java67.blogspot.com/2015/08/2-ways-to-parse-string-to-int-in-java.html))  
  
102) How to swap two integers without using temp variable? ([solution](http://java67.blogspot.com/2015/08/how-to-swap-two-integers-without-using.html))

### Java Interview questions from OOP and Design Patterns

It contains Java Interview questions from SOLID design principles, OOP fundamentals e.g. class, object, interface, Inheritance, Polymorphism, Encapsulation, and Abstraction as well as more advanced concepts like Composition, Aggregation, and Association. It also contains questions from GOF design patterns.  
  
**103) What is the interface? Why you use it if you cannot write anything concrete on it?**  
The interface is used to define API. It tells about the contract your classes will follow. It also supports abstraction because a client can use interface method to leverage multiple implementations e.g. by using List interface you can take advantage of [random access of ArrayList](http://javarevisited.blogspot.com/2015/07/java-arraylist-tutorial.html) as well as flexible insertion and deletion of LinkedList.

The interface doesn't allow you to write code to keep things abstract but from Java 8 you can declare static and default methods inside interface which are concrete.  
  
  
**104) The difference between abstract class and interface in Java?**([answer](http://javarevisited.blogspot.sg/2013/05/difference-between-abstract-class-vs-interface-java-when-prefer-over-design-oops.html))  
There are multiple differences between abstract class and interface in Java, but the most important one is Java's restriction on allowing a class to extend just one class but allows it to implement multiple interfaces.

An abstract class is good to define default behavior for a family of class, but the interface is good to define Type which is later used to leverage Polymorphism. Please check the answer for a more thorough discussion of this question.  
  
  
**105) Which design pattern have you used in your production code? apart from Singleton?**  
This is something you can answer from your experience. You can generally say about dependency injection, factory pattern, decorator pattern or observer pattern, whichever you have used. Though be prepared to answer follow-up question based upon the pattern you choose.  
  
  
**106) Can you explain Liskov Substitution principle?** ([answer](http://javarevisited.blogspot.com/2012/03/10-object-oriented-design-principles.html))  
This is one of the toughest questions I have asked in Java interviews. Out of 50 candidates, I have almost asked only 5 have managed to answer it. I am not posting an answer to this question as I like you to do some research, practice and spend some time to understand this confusing principle well.  
  
  
**107) What is Law of Demeter violation? Why it matters?** ([answer](http://javarevisited.blogspot.com/2014/05/law-of-demeter-example-in-java.html))  
Believe it or not, Java is all about application programming and structuring code. If  you have good knowledge of common coding best practices, patterns and what not to do than only you can write quality code.  Law of Demeter suggests you "talk to friends and not stranger", hence used to reduce coupling between classes.  
  
  
**108) What is Adapter pattern? When to use it?**  
Another frequently asked Java design pattern questions. It provides interface conversion. If your client is using some interface but you have something else, you can write an Adapter to bridge them together. This is good for Java software engineer having 2 to 3 years experience because the question is neither difficult nor tricky but requires knowledge of OOP design patterns.  
  
  
**109) What is "dependency injection" and "inversion of control"? Why would someone use it?**([answer](http://javarevisited.blogspot.sg/2012/12/inversion-of-control-dependency-injection-design-pattern-spring-example-tutorial.html))  
  
**110) What is an abstract class? How is it different from an interface? Why would you use it?**([answer](http://java67.blogspot.sg/2014/06/why-abstract-class-is-important-in-java.html))  
One more classic question from Programming Job interviews, it is as old as chuck Norris. An abstract class is a class which can have state, code and implementation, but an interface is a contract which is totally abstract. Since I have answered it many times, I am only giving you the gist here but you should read the article linked to answer to learn this useful concept in much more detail.  
  
  
**111) Which one is better constructor injection or setter dependency injection?**([answer](http://javarevisited.blogspot.sg/2012/11/difference-between-setter-injection-vs-constructor-injection-spring-framework.html))  
Each has their own advantage and disadvantage. Constructor injection guaranteed that class will be initialized with all its dependency, but setter injection provides flexibility to set an optional dependency.

Setter injection is also more readable if you are using an XML file to describe dependency. Rule of thumb is to use constructor injection for mandatory dependency and use setter injection for optional dependency.  
  
  
**112) What is difference between dependency injection and factory design pattern?** ([answer](http://javarevisited.blogspot.sg/2015/06/difference-between-dependency-injection.html))  
Though both patterns help to take out object creation part from application logic, use of dependency injection results in cleaner code than factory pattern. By using dependency injection, your classes are nothing but POJO which only knows about dependency but doesn't care how they are acquired.

In the case of factory pattern, the class also needs to know about factory to acquire dependency. hence, DI results in more testable classes than factory pattern. Please see the answer for a more detailed discussion on this topic.  
  
  
**113) Difference between Adapter and Decorator pattern?**([answer](http://javarevisited.blogspot.sg/2015/01/adapter-vs-decorator-vs-facade-vs-proxy-pattern-java.html))  
Though the structure of Adapter and Decorator pattern is similar, the difference comes on the intent of each pattern. The adapter pattern is used to bridge the gap between two interfaces, but Decorator pattern is used to add new functionality into the class without the modifying existing code.  
  
  
**114) Difference between Adapter and Proxy Pattern?**([answer](http://javarevisited.blogspot.sg/2015/01/adapter-vs-decorator-vs-facade-vs-proxy-pattern-java.html))  
Similar to the previous question, the difference between Adapter and Proxy patterns is in their intent. Since both Adapter and Proxy pattern encapsulate the class which actually does the job, hence result in the same structure, but Adapter pattern is used for interface conversion while the Proxy pattern is used to add an extra level of indirection to support distribute, controlled or intelligent access.  
  
  
**115) What is Template method pattern?** ([answer](https://javarevisited.blogspot.com/2022/10/template-method-pattern-in-java-example.html))  
Template pattern provides an outline of an algorithm and lets you configure or customize its steps. For examples, you can view a sorting algorithm as a template to sort object. It defines steps for sorting but let you configure how to compare them using Comparable or something similar in another language. The method which outlines the algorithms is also known as template method.  
  
  
**116) When do you use Visitor design pattern?**([answer](https://www.java67.com/2022/12/visitor-design-patterns-in-java.html))  
The visitor pattern is a solution of problem where you need to add operation on a class hierarchy but without touching them. This pattern uses double dispatch to add another level of indirection.  
  
  
**117) When do you use Composite design pattern?**([answer](https://javarevisited.blogspot.com/2018/02/composite-design-pattern-in-java-real.html))  
Composite design pattern arranges objects into tree structures to represent part-whole hierarchies. It allows clients treat individual objects and container of objects uniformly. Use Composite pattern when you want to represent part-whole hierarchies of objects.

**118) The difference between Inheritance and Composition?** ([answer](http://javarevisited.blogspot.sg/2015/06/difference-between-inheritance-and-Composition-in-Java-OOP.html))  
Though both allows code reuse, Composition is more flexible than Inheritance because it allows you to switch to another implementation at run-time. Code written using Composition is also easier to test than code involving inheritance hierarchies.  
  
  
**119) Describe overloading and overriding in Java?** ([answer](http://java67.blogspot.sg/2012/09/difference-between-overloading-vs-overriding-in-java.html))  
Both overloading and overriding allow you to write two methods of different functionality but with the same name, but overloading is compile time activity while overriding is run-time activity. Though you can overload a method in the same class, but you can only override a method in child classes. Inheritance is necessary for overriding.  
  
  
**120) The difference between nested public static class and a top level class in Java?** ([answer](http://javarevisited.blogspot.sg/2012/12/inner-class-and-nested-static-class-in-java-difference.html))  
You can have more than one nested public static class inside one class, but you can only have one top-level public class in a Java source file and its name must be same as the name of Java source file.  
  
  
**121) Difference between Composition, Aggregation and Association in OOP?** ([answer](http://javarevisited.blogspot.sg/2014/02/ifference-between-association-vs-composition-vs-aggregation.html))  
If two objects are related to each other, they are said to be associated with each other. Composition and Aggregation are two forms of association in object-oriented programming. The composition is stronger association than Aggregation.

In Composition, one object is OWNER of another object while in Aggregation one object is just USER of another object. If an object A is composed of object B then B doesn't exist if A ceased to exists, but if object A is just an aggregation of object B then B can exists even if A ceased to exist.  
  
  
**122) Give me an example of design pattern which is based upon open closed principle?** ([answer](http://javarevisited.blogspot.sg/2011/11/great-example-of-open-closed-design.html))  
This is one of the practical questions I ask experienced Java programmer. I expect them to know about OOP design principles as well as patterns. Open closed design principle asserts that your code should be open for extension but closed for modification.

Which means if you want to add new functionality, you can add it easily using the new code but without touching already tried and tested code.  There are several design patterns which are based upon open closed design principle e.g. [Strategy pattern](http://java67.blogspot.com/2014/12/strategy-pattern-in-java-with-sample.html) if you need a new strategy, just implement the interface and configure, no need to modify core logic.

One working example is Collections.sort() method which is based on Strategy pattern and follows the open-closed principle, you don't modify sort() method to sort a new object, what you do is just implement Comparator in your own way.  
  
  
**123) Difference between Abstract factory and Prototype design pattern?** (answer)  
This is the practice question for you, If you are feeling bored just reading and itching to write something, why not write the answer to this question. I would love to see an example the, which should answer where you should use the Abstract factory pattern and where is the Prototype pattern is more suitable.  
  
  
**124) When do you use Flyweight pattern?** (answer)  
This is another popular question from the design pattern. Many Java developers with 4 to 6 years of experience know the definition but failed to give any concrete example. Since many of you might not have used this pattern, it's better to look examples from JDK. You are more likely have used them before and they are easy to remember as well.

Now let's see the answer.

Flyweight pattern allows you to share object to support large numbers without actually creating too many objects. In order to use Flyweight pattern, you need to make your object Immutable so that they can be safely shared. String pool and pool of Integer and Long object in JDK are good examples of Flyweight pattern.

### Miscellaneous Java Interview Questions

It contains XML Processing in Java Interview question, JDBC Interview question, Regular expressions Interview questions, Java Error and Exception Interview Questions, Serialization,  
  
**125) The difference between nested static class and top level class?**([answer](http://java67.blogspot.sg/2012/10/nested-class-java-static-vs-non-static-inner.html))  
One of the fundamental questions from Java basics. I ask this question only to junior Java developers of 1 to 2 years of experience as it's too easy for an experience Java programmers. The answer is simple, a public top level class must have the same name as the name of the source file, there is no such requirement for nested static class.

A nested class is always inside a top level class and you need to use the name of the top-level class to refer nested static class e.g. HashMap.Entry is a nested static class, where HashMap is a top level class and Entry is nested static class.  
  
  
**126) Can you write a regular expression to check if String is a number?**([solution](http://javarevisited.blogspot.sg/2012/10/regular-expression-example-in-java-to-check-String-number.html))  
If you are taking Java interviews then you should ask at least one question on the regular expression. This clearly [differentiates an average programmer with a good programmer](http://javarevisited.blogspot.com/2015/05/how-to-differentiate-between-average.html). Since one of the traits of a good developer is to know tools, regex is the best tool for searching something in the log file, preparing reports etc.

Anyway, answer to this question is, a numeric String can only contain digits i.e. 0 to 9 and + and - sign that too at start of the String, by using this information you can write following regular expression to check if given String is number or not  
  
  
**127) The difference between checked and unchecked Exception in Java?**([answer](http://java67.blogspot.sg/2012/12/difference-between-runtimeexception-and-checked-exception.html))  
checked exception is checked by the compiler at compile time. It's mandatory for a method to either handle the checked exception or declare them in their throws clause. These are the ones which are a sub class of Exception but doesn't descend from RuntimeException.

The unchecked exception is the descendant of RuntimeException and not checked by the compiler at compile time. This question is now becoming less popular and you would only find this with interviews with small companies, both investment banks and startups are moved on from this question.  
  
  
**128) The difference between throw and throws in Java?** ([answer](http://javarevisited.blogspot.sg/2012/02/difference-between-throw-and-throws-in.html))  
the throw is used to actually throw an instance of java.lang.Throwable class, which means you can throw both Error and Exception using throw keyword e.g.

throw new IllegalArgumentException("size must be multiple of 2")

On the other hand, throws is used as part of method declaration and signals which kind of exceptions are thrown by this method so that its caller can handle them. It's mandatory to declare any unhandled checked exception in **throws** clause in Java. Like the previous question, this is another frequently asked Java interview question from errors and exception topic but too easy to answer.  
  
  
**129) The difference between Serializable and Externalizable in Java?** ([answer](http://javarevisited.blogspot.sg/2012/01/serializable-externalizable-in-java.html))  
This is one of the frequently asked questions from Java Serialization. The interviewer has been asking this question since the day Serialization was introduced in Java, but yet only a few good candidate can answer this question with some confidence and practical knowledge.

Serializable interface is used to make Java classes serializable so that they can be transferred over network or their state can be saved on disk, but it leverages default serialization built-in JVM, which is expensive, fragile and not secure.

Externalizable allows you to fully control the Serialization process, specify a custom binary format and add more security measure.  
  
  
**130) The difference between DOM and SAX parser in Java?**([answer](http://javarevisited.blogspot.sg/2011/12/difference-between-dom-and-sax-parsers.html))  
Another common Java question but from XML parsing topic. It's rather simple to answer and that's why many interviewers prefers to ask this question on the telephonic round. DOM parser loads the whole XML into memory to create a tree based DOM model which helps it quickly locate nodes and make a change in the structure of XML while SAX parser is an event based parser and doesn't load the whole XML into memory. Due to this reason DOM is faster than SAX but require more memory and not suitable to parse large XML files.  
  
  
**131) Tell me 3 features introduced on JDK 1.7?**([answer](http://javarevisited.blogspot.sg/2014/04/10-jdk-7-features-to-revisit-before-you.html))  
This is one of the good questions I ask to check whether the candidate is aware of recent development in Java technology space or not. Even though JDK 7 was not a big bang release like JDK 5 or JDK 8, it still has a lot of good feature to count on e.g. try-with-resource statements, which free you from closing streams and resources when you are done with that, Java automatically closes that.

Fork-Join pool to implement something like the Map-reduce pattern in Java. Allowing String variable and literal into switch statements. Diamond operator for improved type inference, no need to declare generic type on the right-hand side of variable declaration anymore, results in more readable and succinct code.

Another worth noting feature introduced was improved exception handling e.g. allowing you to catch multiple exceptions in the same catch block.  
  
  
**132) Tell me 5 features introduced in JDK 1.8?**([answer](http://javarevisited.blogspot.sg/2014/02/10-example-of-lambda-expressions-in-java8.html))  
This is the follow-up question of the previous one. Java 8 is path breaking release in Java's history, here are the top 5 features from JDK 8 release

* **Lambda expression**, which allows you pass an anonymous function as object.
* **Stream API**, take advantage of multiple cores of modern CPU and allows you to write succinct code.
* **Date and Time API**, finally you have a solid and easy to use date and time library right into JDK
* **Extension methods**, now you can have static and default method into your interface
* **Repeated annotation**, allows you apply the same annotation multiple times on a type

**133) What is the difference between Maven and ANT in Java?** ([answer](http://javarevisited.blogspot.sg/2015/01/difference-between-maven-ant-jenkins-and-hudson.html))  
Another great question to check the all round knowledge of Java developers. It's easy to answer questions from core Java but when you ask about setting things up, building Java artifacts, many Java software engineer struggles.

Coming back to the answer of this question, Though both are build tools and used to create Java application build, Maven is much more than that. It provides a standard structure for Java project based upon the "convention over configuration" concept and automatically manages dependencies (JAR files on which your application is dependent) for Java applications. Please see the answer for more differences between the Maven and ANT tools.

**SPRING BOOT**

**What is Spring Boot, and how does it differ from Spring Framework?**

Spring Boot is a framework designed to simplify the development of Spring-based applications. Spring Boot aims to minimize boilerplate code and provides defaults for various components. This makes it easier to get started with Spring-based applications.

**Explain the benefits of using Spring Boot for application development.**

• Simplified setup and configuration through auto-configuration and starter dependencies.

• Reduced boilerplate code, enabling developers to focus more on application logic.

• Embedded server support, allowing applications to be run as standalone JAR files.

• Enhanced testability through the provision of test utilities and annotations.

**What are the key features of Spring Boot?**

**Auto-configuration**: Automatically configures Spring-based applications based on dependencies and defaults.

**Starter dependencies**: Pre-packaged dependencies that simplify the setup of specific application features or technologies.

**Developer tools**: Tools that enhance developer productivity such as automatic application restarts and live reload.

**Actuator**: Provides endpoints for monitoring and managing applications at runtime.

**Explain the concept of Spring Boot starters and provide an example.**

Starters are a collection of dependency descriptors, which can help simplify your dependency management.

For instance, if you want to get started with Spring JPA, you just include the spring-boot-starter-data-jpa dependency and everything required for it (like Hibernate, Spring Data, etc.) will be added to your application.

**What is the purpose of the @SpringBootApplication annotation?**

The @SpringBootApplication annotation is a convenience annotation provided by Spring Boot. It serves as the entry point for the Spring Boot application. It combines three commonly used annotations:

* @Configuration
* @EnableAutoConfiguration
* @ComponentScan.6.

**What is the default port number for a Spring Boot application?**

The default port number for a Spring Boot application is 8080.

**How can you enable the auto-configuration feature in Spring Boot?**

Auto-configuration is enabled by default in Spring Boot.

If needed, you can disable specific auto-configuration classes or customize the configuration by providing your own beans.

**How does Spring Boot handle external configuration?**

Spring Boot provides multiple ways to handle external configurations. It supports property files (application.properties or application.yml) that can be placed in various locations including the classpath, file system, or external directories.

Spring Boot also supports environment variables, command-line arguments, and the use of profiles for different deployment environments. The configuration values can be accessed using the @Value annotation or by binding them to Java objects using the @ConfigurationProperties annotation

**What is the purpose of the application.properties (or application.yml) file?**

The application.properties or application.yml file is used for external configuration in a Spring Boot application. It allows developers to specify various properties and their values to configure the application.

These properties can control various aspects of the application such as server port, database connection details, logging configuration, and much more. The properties file can be placed in the classpath or other predefined locations, and Spring Boot will automatically load and apply the configuration during application startup.

**Explain the difference between @Component, @Repository, @Service, and @Controller annotations in Spring Boot.**

**@Component**: It is a generic stereotype annotation used to mark a class as a Spring-managed component. It is a broad and generic term that can be used for any type of Spring-managed component.

**@Repository**: It is a specialized form of @Component used to indicate that a class is a repository or data access component. It typically encapsulates database operations and exception translation.

**@Service**: It is a specialized form of @Component used to indicate that a class is a service component. It encapsulates business logic and is often used as an intermediate layer between controllers and repositories.

@**Controller**: It is a specialized form of @Component used to indicate that a class is a web controller component. It handles incoming requests, performs business logic, and prepares the response to be sent back to the client.

**What is the role of the @Autowired annotation in Spring Boot?**

The @Autowired annotation is used for dependency injection in Spring Boot. When applied to a field, setter method, or constructor, it allows Spring to automatically resolve and inject the required dependencies.

By using @Autowired, developers don't need to manually instantiate and wire dependencies. Spring Boot scans the application context for beans matching the required type and injects them automatically.

**How can you implement logging in a Spring Boot application?**

In a Spring Boot application, logging is typically implemented using a logging framework such as Logback or Log4j2. Spring Boot provides a default logging configuration out of the box.

You can configure logging levels, appenders, and log formats using the application.properties or application.yml file. Additionally, you can include the desired logging framework dependencies in your project's build configuration and use the framework's APIs to perform logging within your application code.

**What is the purpose of the SpringApplication.run() method?**

The SpringApplication.run() method is used to bootstrap and launch a Spring Boot application. It is typically invoked from the main method of the application's entry point class.

The run() method initializes the Spring application context, performs auto-configuration, starts the embedded server, and starts the application lifecycle. It returns an instance of the ApplicationContext, allowing access to the application context and its beans.

**What is Spring Boot CLI?**

Spring Boot Command Line Interface (CLI) is a command line tool that you can use to run and test Spring Boot applications from a command prompt. It provides a fast way to get Spring applications up and running.

Some of the advantages of using Spring Boot CLI are:

• It allows you to write your application using Groovy.

• It automatically includes useful external libraries whenever possible. For example, if you're writing a web application and importing classes such as @RestController, the CLI will automatically provide a dependency for Spring MVC.

• You can use various commands for different operations like run (to run the application), test (to test the application), jar (to create a jar file), init (to create a basic Java or Groovy project), etc.

**How does Spring Boot handle data validation?**

In Spring Boot, data validation can be performed using various mechanisms. One common approach is to use the validation annotations provided by the Bean Validation API, such as @NotNull, @Size, and @Pattern, on the fields of model objects.

**What is the purpose of the @RequestMapping annotation in Spring Boot?**

The @RequestMapping annotation is used to map HTTP requests to specific handler methods in a Spring Boot application. It is applied at the method or class level to define the URL patterns that should trigger the execution of the annotated method.

@RequestMapping allows developers to specify various attributes, such as the HTTP method (GET, POST, etc.), request parameters, headers, and more to further refine the mapping.

**How does Spring Boot integrate with containerization platforms like Docker and Kubernetes?**

Spring Boot integrates seamlessly with containerization platforms like Docker and Kubernetes. You can package a Spring Boot application as a Docker image by creating a Dockerfile that includes the necessary dependencies and configurations.

The image can be built and deployed to a containerization platform like Docker Swarm or Kubernetes. Spring Boot also provides features like externalized configuration and health indicators which can be leveraged by container orchestration platforms for efficient management and scaling of the application.

**Explain the concept of message-driven microservices using Spring Boot and Apache Pulsar.**

Message-driven microservices using Spring Boot and Apache Pulsar leverage the publish-subscribe messaging pattern to enable loosely coupled and scalable communication between microservices. Apache Pulsar acts as the messaging system, and Spring Boot provides the necessary abstractions for consuming and producing messages.

**What is the purpose of the @Value annotation in Spring Boot?**

The @Value annotation is used to inject values from properties files, environment variables, or other sources into Spring-managed beans. It can be applied to fields, methods, or constructor parameters.

With @Value, developers can easily access and use configuration properties or other values within their application code. The values can be specified directly or referenced using SpEL (Spring Expression Language) expressions.

**Describe the role of the CommandLineRunner and ApplicationRunner interfaces in Spring Boot.**

In Spring Boot, the CommandLineRunner and ApplicationRunner interfaces are used for performing specific tasks during the application startup process. When implemented, these interfaces provide a callback method (run()) that gets executed once the application context is initialized.

They are particularly useful for performing tasks like data initialization, cache population, or other one-time setup operations. The main difference between them is that CommandLineRunner receives the application's command-line arguments as a parameter, while ApplicationRunner receives an ApplicationArguments object.

**How can you implement pagination in a Spring Boot application?**

To implement pagination in a Spring Boot application, you can utilize features provided by libraries like Spring Data JPA or Spring Data MongoDB. They offer built-in support for pagination through the use of Pageable objects and repository methods.

**Explain the concept of bean scopes in Spring Boot.**

Bean scopes define the lifecycle and visibility of Spring-managed beans in a Spring Boot application. The following are the commonly used bean scopes:

Singleton (default): Only one instance of the bean is created and shared across the entire application context.

Prototype: A new instance of the bean is created each time it is requested.

Request: A new instance of the bean is created for each HTTP request. It is only applicable in a web application context.

Session: A new instance of the bean is created for each user session. It is only applicable in a web application context.

Custom scopes: Spring Boot allows defining custom bean scopes by implementing the Scope interface and registering them in the application context.

**What is the purpose of the @Qualifier annotation in Spring Boot?**

The @Qualifier annotation in Spring is used to disambiguate bean references when we have multiple beans of the same type defined in the Spring container. It is used in scenarios where a given type has more than one implementation and we need to inject a specific implementation.

The @Qualifier annotation can be used in conjunction with @Autowired to specify which exact bean should be wired, by providing the name of the bean as the qualifier value.

**How does Spring Boot handle exception logging and error handling?**

In Spring Boot, exception logging and error handling can be configured using various mechanisms. Spring Boot automatically provides a default error page that displays a standardized error message for unhandled exceptions.

However, you can customize the error-handling behavior by implementing exception handlers using the @ControllerAdvice annotation and handling specific exceptions in dedicated methods.

Additionally, you can configure logging frameworks to capture and log exceptions with desired levels of detail and appenders.

**Describe the purpose and usage of the @RestControllerAdvice annotation.**

The @RestControllerAdvice annotation is a specialized form of the @ControllerAdvice annotation in Spring Boot. It combines the functionality of @ControllerAdvice and @ResponseBody, making it convenient for implementing global exception handling in RESTful APIs.

By using @RestControllerAdvice, you can define exception handlers that handle exceptions thrown by any @RequestMapping or @RestController method within the application. The exception handlers can return error responses in JSON or other supported formats.

**What is the purpose of the @ConfigurationProperties annotation in Spring Boot?**

The @ConfigurationProperties annotation is used to bind external configuration properties to Spring-managed beans. By annotating a bean class with @ConfigurationProperties and specifying a prefix, you can map properties with matching names to the fields or setter methods of the bean.

Spring Boot will automatically bind the values from the configuration sources to the corresponding bean properties. The annotation simplifies the retrieval and usage of configuration properties within your application.

**Describe the purpose and usage of the @DynamicPropertySource annotation in Spring Boot testing.**

The @DynamicPropertySource annotation in Spring Boot testing allows you to dynamically define and modify configuration properties during the test execution. You can use this annotation in conjunction with the TestPropertyValues class to set or override properties based on dynamic values or test conditions.

This provides flexibility in configuring the environment for testing and allows you to simulate different scenarios or configurations during testing.

**What is the purpose of the @TransactionalEventListener annotation in Spring Boot?**

The @TransactionalEventListener annotation in Spring Boot lets you listen to transactional events and perform actions based on those events. You can use this annotation on methods that should be invoked when a specific transactional event occurs such as before or after a transaction is committed or rolled back.

The @TransactionalEventListener annotation provides a convenient way to handle domain-specific logic or side effects based on transactional events in a Spring Boot application.

**What is the purpose of the @Scheduled annotation in Spring Boot?**

The @Scheduled annotation is used to configure scheduled tasks in a Spring Boot application. Applying this annotation to a method enables you to specify the schedule at which the method should be executed.

The schedule can be defined using various options such as fixed-rate, fixed-delay, or cron expressions. Spring Boot automatically detects and executes the scheduled methods based on the specified schedule.

**Describe the role of the @Profile annotation in Spring Boot.**

The @Profile annotation is used to activate or deactivate specific configuration components or beans based on the current environment or profile in a Spring Boot application. Annotating a class or method with @Profile and specifying the desired profile name lets you control when that component or bean should be active. This allows you to have different configurations for different deployment environments such as development, testing, or production.

**What is the purpose of Spring Boot's dynamic reloading and how does it work?**

Spring Boot's dynamic reloading feature allows you to make changes to the application code or resources without the need to restart the entire application. It improves development productivity by automatically reloading the modified classes or resources on the fly.

The dynamic reloading feature uses class reloading mechanisms provided by the underlying JVM, such as Java Instrumentation API or custom class loaders, to reload the changed classes while preserving the application's state.

**Explain the concept of externalized logging in Spring Boot using Logback or Log4j2**.

Externalized logging in Spring Boot allows you to configure and customize logging behavior without modifying the application code. Logback or Log4j2 can be used as the underlying logging framework.

The configuration is typically done in an external configuration file, such as logback.xml or log4j2.xml, which can be placed in the classpath or specified using the logging.config property. The externalized logging configuration file provides flexibility in defining log levels, appenders, formatters, and other logging-related properties.

**What is the purpose of the @ModelAttribute annotation in Spring Boot?**

The @ModelAttribute annotation is used in Spring Boot to bind request parameters or form data to method parameters or model attributes. It can be applied to method parameters or method return values.

When applied to method parameters, the @ModelAttribute annotation binds the incoming request parameters or form data to the corresponding method parameters. When applied to method return values, it binds the method's return value to a model attribute, making it available in the view for rendering.

**Describe the purpose and usage of the @Transactional (propagation = Propagation.NESTED) annotation.**

The @Transactional(propagation = Propagation.NESTED) annotation is used to define a nested transactional scope in a Spring Boot application. When a method is annotated with this annotation, a nested transaction is created within the current transaction.

The nested transaction behaves as an independent transaction and can be rolled back separately from the outer transaction. If the nested transaction fails, only the changes made within the nested transaction are rolled back, while the outer transaction remains unaffected.

**What is the purpose of the @DataJpaTest annotation in Spring Boot testing?**

The @DataJpaTest annotation is used to configure and customize the testing environment for JPA repositories in a Spring Boot application. When applied to a test class, it sets up an in-memory database, configures Spring Data JPA, and loads only the necessary components for testing JPA repositories.

@DataJpaTest provides a lightweight and isolated environment for testing JPA-related functionality without requiring a full application context or a real database connection.

**Describe the purpose and usage of the Spring Boot Admin Server for monitoring and managing applications.**

The Spring Boot Admin Server is a tool that provides a web-based interface for monitoring and managing multiple Spring Boot applications in a centralized manner. It collects and displays various metrics, health statuses, and other information about the registered Spring Boot applications.

The Admin Server allows you to view and manage application details, monitor JVM metrics, and receive alerts on specific conditions. It simplifies the monitoring and management of Spring Boot applications in a production environment.

**Differentiate between Spring MVC and Spring Boot.**

Spring MVC is a framework for building web applications using the Model-View-Controller (MVC) architectural pattern. It provides features for handling requests, managing controllers, rendering views, and managing data flow.

Spring Boot, on the other hand, is an opinionated framework built on top of Spring that aims to simplify the setup and configuration of Spring applications. It provides out-of-the-box defaults and auto-configuration, reducing the need for manual configuration and boilerplate code.

**Explain the concept of dependency injection in Spring Boot.**

Hide Answer

Dependency injection is a core concept in Spring Boot. It allows objects to be loosely coupled by providing their dependencies from external sources. Spring Boot uses inversion of control (IoC) and the dependency injection pattern to manage dependencies.

Spring Boot automatically resolves and injects the required dependencies at runtime by annotating classes with appropriate annotations such as @Autowired.

**How does Spring Boot support database operations?**

Spring Boot provides excellent support for database operations through its integration with Spring Data JPA. By defining entities and repositories, you can perform CRUD (Create, Read, Update, Delete) operations on databases with minimal boilerplate code.

Spring Boot automatically configures the database connection and transaction management, and provides powerful querying capabilities.

**Describe the role of the Spring Boot Actuator.**

Spring Boot Actuator is a feature that provides insight into the runtime of a Spring Boot application. It offers a set of production-ready endpoints that expose information about application health, metrics, environment, logging, and more.

The Actuator enables monitoring and management of the application, making it easier to understand and troubleshoot in production environments.

**What is the purpose of the @Scheduled annotation in Spring Boot?**

The @Scheduled annotation is used to schedule the execution of a method at fixed intervals or specific times. It allows you to define cron expressions or fixed delay/initial delay values. Spring Boot automatically triggers the annotated method based on the specified schedule, making it suitable for performing recurring tasks such as data synchronization or sending periodic notifications.

**How can you enable cross-origin resource sharing (CORS) in a Spring Boot application?**

To enable CORS in a Spring Boot application, you can use the @CrossOrigin annotation at the controller level or globally configure CORS using a WebMvcConfigurer bean. The annotation allows you to specify the allowed origins, HTTP methods, headers, and other CORS-related settings. Enabling CORS ensures that web browsers can make requests to your application from different domains.

**Describe the role of the @Transactional annotation in Spring Boot.**

The @Transactional annotation is used to mark a method or class for transaction management in Spring Boot. It ensures that the annotated method or all methods within the annotated class are executed within a transactional context.

The @Transactional annotation manages the transaction boundaries, rollback behavior, and other transactional aspects to ensure data consistency and integrity.

**How does Spring Boot handle internationalization (i18n) and localization (l10n)?**

Spring Boot automatically resolves the appropriate message based on the user's locale, making it convenient to build multi-language applications. It provides support for internationalization and localization through properties files and the use of the MessageSource interface.

By defining message bundles for different locales and configuring the message source, you can easily retrieve and display localized messages in your application.

**What is the purpose of the @RestControllerAdvice annotation?**

The @RestControllerAdvice annotation combines the functionalities of @ControllerAdvice and @ResponseBody annotations. It is used to define a global exception handler for RESTful controllers in a Spring Boot application.

Annotated classes can contain exception-handling methods annotated with @ExceptionHandler which handles exceptions thrown within any @RestController in the application. These methods can return custom error responses or perform other actions based on the exception type.

**Explain the concept of Spring Data REST and its advantages.**

Spring Data REST is a project built on top of Spring Data, it takes the features of Spring HATEOAS and Spring Data to build Spring MVC-based RESTful services with less code. With Spring Data REST, you can leverage your Spring Data repositories and convert them into full-featured RESTful services with ease.

Some of its advantages are:

Rapid Development: With Spring Data REST, a great deal of your HTTP resource implementation time can be saved. It's quick and easy to build a RESTful service with full CRUD functionality.

Data Access: It leverages Spring Data's repositories and provides seamless, RESTful access to your data model.

HAL Browser: Spring Data REST includes support for the HAL Browser, allowing users to navigate, create, update, and delete resources directly from their web browsers.

Search Support: It has built-in support for searches. Custom repository methods are automatically exposed as HTTP resources.

**How can you implement file upload and download functionality in a Spring Boot application?**

File upload and download functionality can be implemented in a Spring Boot application by configuring multipart file handling. By using the MultipartFile object as a method parameter, Spring Boot automatically binds uploaded files to it.

For file download, you can return the file as a response with appropriate headers. Additionally, you can leverage storage services like Amazon S3 or Azure Blob Storage for file storage and retrieval.

**Describe the purpose and usage of the @Async annotation in Spring Boot.**

The @Async annotation is used to indicate that a method should be executed asynchronously. When a method is annotated with @Async, Spring Boot runs it in a separate thread from a task executor, allowing the caller to continue execution without waiting for the asynchronous method to complete.

This annotation is useful for offloading time-consuming tasks, improving performance, and providing a more responsive user experience.

**What is the role of the embedded servlet container in Spring Boot?**

The embedded servlet container in Spring Boot allows you to run web applications without the need for a separate web server. It provides a lightweight servlet container, such as Tomcat, Jetty, or Undertow, that is embedded within the application.

Spring Boot automatically configures and starts the embedded servlet container, simplifying the deployment and execution of web applications.

**How can you implement request and response logging in a Spring Boot application?**

Request and response logging in a Spring Boot application can be implemented using filters or interceptors. Creating a custom filter or interceptor lets you intercept incoming requests and outgoing responses and log their details, such as headers, payloads, and other relevant information.

Spring Boot allows you to register these filters or interceptors in the application's configuration, enabling centralized logging across the application.

21.

Explain the concept of reactive data access in Spring Boot using Spring Data R2DBC.

Hide Answer

Reactive data access in Spring Boot allows you to build non-blocking and efficient applications that handle a large number of concurrent requests. Spring Data R2DBC provides reactive database access by integrating with R2DBC (Reactive Relational Database Connectivity).

It enables you to perform asynchronous database operations using reactive programming paradigms, such as Flux and Mono, providing better scalability and responsiveness compared to traditional blocking database access.

**Describe the purpose and usage of the @Conditional annotation in Spring Boot.**

The @Conditional annotation in Spring Boot allows you to conditionally activate or deactivate beans or configurations based on specific conditions. By annotating a bean or configuration class with @Conditional and providing a condition class implementing the Condition interface, you can control whether the bean or configuration should be created and registered based on runtime conditions. This enables flexible configuration based on environment, properties, or other factors.

**What is the purpose of the @SpringBootTest annotation in Spring Boot testing?**

The @SpringBootTest annotation is used to bootstrap a Spring Boot application context for testing purposes. It allows you to load the entire application context, including all configurations and beans, during integration tests.

@SpringBootTest provides features like auto-configuration, dependency injection, and easy access to application-specific components, enabling comprehensive testing of Spring Boot applications.

**How can you secure REST APIs in a Spring Boot application using JSON Web Tokens (JWT)?**

You can secure REST APIs in a Spring Boot application using JSON Web Tokens (JWT) by integrating Spring Security and JWT libraries. Spring Security provides mechanisms for authentication and authorization, while JWT facilitates token-based authentication.

By configuring Spring Security filters, implementing authentication and authorization providers, and validating JWT tokens, you can protect your REST APIs and control access based on user roles and permissions.

**Describe the purpose and usage of the @EntityScan annotation in Spring Boot.**

The @EntityScan annotation is used to specify the base packages to scan for entity classes in a Spring Boot application. When using JPA (Java Persistence API) with Spring Boot, @EntityScan helps the JPA provider locate and manage entity classes.

By default, Spring Boot scans the package of the application's main class and its sub-packages. However, if entity classes are located in different packages, you need to use @EntityScan to include those packages.

**What is the purpose of the @Retryable annotation in Spring Boot?**

The @Retryable annotation is used to specify that a method should be retried if it fails due to specified exceptions. Adding @Retryable and configuring the desired retry behavior enables Spring Boot to automatically retry the method when exceptions occur.

This can be useful for handling transient errors, such as network timeouts or temporary resource unavailability, and ensuring the successful execution of critical operations.

**Explain the concept of auto-reconfiguration in Spring Boot and its limitations.**

Auto-reconfiguration in Spring Boot is a feature that automatically configures certain components and dependencies based on the classpath and available resources. It simplifies the configuration process by detecting and configuring components like data sources, messaging brokers, and caches.

Auto-reconfiguration has limitations when it comes to complex or custom configurations. It may not always provide the desired configuration out of the box. In such cases, manual configuration may be required.

**How can you create a Spring Boot application using Gradle?**

To create a Spring Boot application using Gradle, follow these steps:

• Set up a new Gradle project or add Spring Boot dependencies to an existing Gradle project.

• Make sure you have the required plugins configured in the build.gradle file, such as the org.springframework.boot and io.spring.dependency-management plugins.

• Define the necessary dependencies in the dependencies section of the build.gradle file. Specify the desired Spring Boot starter dependencies.

• Create the main application class and annotate it with @SpringBootApplication.

• Implement the application logic within the main application class or other components.

• Use the Gradle command line or an IDE plugin to build and run the application.

**How can you customize the default error pages in a Spring Boot application?**

To customize the default error pages in a Spring Boot application, you can create an error page template or controller method that handles the error. Defining an error template with the appropriate name and placing it in the src/main/resources/templates/error directory enables Spring Boot to automatically render that template for the corresponding error status code.

Alternatively, you can create a controller method with @ExceptionHandler annotation to handle specific exceptions and return a custom error response.

**How can you create a Spring Boot application using Maven?**

To create a Spring Boot application using Maven, follow these steps:

• Set up a new Maven project or add Spring Boot dependencies to an existing Maven project.

• Ensure that the project's dependencies include the spring-boot-starter-parent as the parent project.

• Define the necessary dependencies in the project's pom.xml file, such as spring-boot-starter-web for web applications.

• Create the main application class and annotate it with @SpringBootApplication.

• Implement the application logic within the main application class or other components.

• Use the Maven command line or an IDE plugin to build and run the application.

**How can you implement security in a Spring Boot application?**

Security can be implemented in a Spring Boot application by adding the appropriate dependencies, such as spring-boot-starter-security, and configuring the security settings. This can be done by creating a security configuration class that extends WebSecurityConfigurerAdapter and overriding its methods to define authentication and authorization rules.

Additionally, you can customize the login page, handle logout, and secure specific endpoints using annotations like @EnableWebSecurity and @EnableGlobalMethodSecurity.

**What are the different deployment options for a Spring Boot application?**

Spring Boot applications can be deployed in various ways including:

Standalone JAR: Packaging the application as a self-contained executable JAR file with an embedded servlet container like Tomcat or Jetty.

WAR deployment: Packaging the application as a traditional WAR file and deploying it to a servlet container.

Docker: Containerizing the application using Docker and running it on Docker containers.

Cloud platforms: Deploy the application to cloud platforms like AWS, Azure, or Google Cloud using platform-specific deployment options such as AWS Elastic Beanstalk or Azure App Service.

**Describe the process of creating a RESTful API using Spring Boot.**

To create a RESTful API using Spring Boot, you can follow these steps:

• Define your domain model and business logic.

• Create a Spring MVC controller class and define handler methods annotated with @RequestMapping or other mapping annotations.

• Implement the required CRUD operations within the handler methods using appropriate annotations like @GetMapping, @PostMapping, etc.

• Customize the request and response handling with annotations such as @RequestBody to map request payloads and @ResponseBody to define the response body.

• Configure additional features like exception handling, input validation, and security, if required.

• Run the Spring Boot application and the API endpoints will be accessible based on the mappings defined in the controller.

**How can you handle large file uploads in a Spring Boot application?**

To handle large file uploads in a Spring Boot application, you can configure the maximum file size limit in the application's properties file by setting the spring.servlet.multipart.max-file-size and spring.servlet.multipart.max-request-size properties to appropriate values. Additionally, you can use the MultipartFile parameter in the controller method to receive the uploaded file and process it as needed.

**How can you implement microservices architecture using Spring Boot?**

To implement a microservices architecture using Spring Boot, you can follow these steps:

• Identify the different business capabilities and boundaries of your application.

• Design and develop each microservice as a separate Spring Boot application, encapsulating a specific business capability.

• Use lightweight communication mechanisms like REST or messaging for inter-service communication.

• Implement service discovery and registration using tools like Netflix Eureka or HashiCorp Consul.

• Apply fault tolerance and resilience patterns like circuit breakers (Hystrix) and distributed tracing (Sleuth) for better reliability.

• Deploy and manage microservices using containerization platforms like Docker and orchestration tools like Kubernetes.

How does Spring Boot integrate with Apache Kafka for event-driven architectures?

Hide Answer

Spring Boot provides integration with Apache Kafka through the Spring Kafka project. You can include the spring-boot-starter-kafka dependency to get started.

Spring Kafka provides abstractions to produce and consume messages from Kafka topics using the KafkaTemplate and @KafkaListener annotations, respectively. Additionally, Spring Kafka integrates with Spring Boot's auto-configuration to simplify the configuration of Kafka-related properties.

**How can you implement distributed session management in a Spring Boot application using Spring Session?**

To implement distributed session management in a Spring Boot application using Spring Session, follow these steps:

• Include the necessary dependencies for Spring Session and a session store implementation like Redis or Hazelcast.

• Configure the session store details, such as the connection properties, in the application's configuration file (e.g., application.properties or application.yml).

• Enable Spring Session support by annotating your configuration class with @EnableRedisHttpSession (for Redis) or @EnableHazelcastHttpSession (for Hazelcast).

• Spring Session will automatically handle session creation, serialization, and synchronization with the session store, allowing session data to be shared across multiple instances of your application.

**How can you implement serverless functions using Spring Boot and AWS Lambda?**

You can use the spring-cloud-function-adapter-aws dependency to implement serverless functions using Spring Boot and AWS Lambda. You can deploy Spring Boot applications as serverless functions on AWS Lambda by creating a function bean and configuring the AWS Lambda handler.

The adapter takes care of the integration between Spring Cloud Function and AWS Lambda, allowing you to develop serverless functions using the familiar Spring Boot programming model.

**How can you implement method-level security in a Spring Boot application?**

To implement method-level security in a Spring Boot application, you can use the @PreAuthorize or @PostAuthorize annotations provided by Spring Security. Here's an example:

• Configure Spring Security in your application by including the necessary dependencies and configuration.

• Annotate the methods that require security checks with @PreAuthorize or @PostAuthorize.

• Specify the desired security expressions in the annotations to define the required conditions for method invocation.

• Spring Security will evaluate the expressions and allow or deny access to the methods based on the configured security rules.

**How can you implement server-sent events (SSE) in a Spring Boot application?**

To implement server-sent events (SSE) in a Spring Boot application, you can use the SseEmitter class provided by Spring Framework. Here's an example of how to implement SSE:

• Create a controller method that returns an SseEmitter object.

• In this method, use the SseEmitter to send events to the client.

• Use the send() method of SseEmitter to send events periodically or based on specific triggers.

• Set appropriate headers, such as Content-Type and Cache-Control, for SSE support.

• Register the SseEmitter as a handler method in your controller.

**How can you enable HTTPS in a Spring Boot application?**

To enable HTTPS in a Spring Boot application, you need to configure the appropriate SSL certificate and modify the application's configuration. Here are the general steps:

• Obtain an SSL certificate and private key.

• Configure the certificate and private key in the application's configuration, such as application.properties or application.yml, using properties like server.ssl.key-store and server.ssl.key-password.

• Set the server.ssl.enabled property to true to enable HTTPS.

• Optionally, configure other SSL-related properties like the SSL protocol and cipher suites.

**HIBERNATE**

### 1. What is an ORM tool?

An Object Relational Mapping (ORM) tool helps to simplify data creation, manipulation, and access by internally using Java API to interact with the databases. It’s a technique that maps objects stored in a database.

### 2. What does “lightweight” mean?

In the context of computers, “lightweight” describes an app, computer program, or device that doesn’t use many system resources due to its small memory footprint (RAM) and low CPU usage.

### 3. What are the advantages of Hibernate?

Here’s a list of Hibernate’s many advantages:

* It’s fast
* It’s lightweight and open source
* It reduces code length, removing boilerplate code, freeing up developers for other tasks
* It strengthens the object-level relationship
* It facilitates the generation of independent database queries
* It provides resources for creating tables automatically
* It’s easy to integrate with other Java Enterprise Edition (EE) frameworks.

### 4. Why is Hibernate better than Java Database Connectivity (JDBC)?

Hibernate outclasses JDBC because:

* Hibernate code is cleaner and more readable thanks to the elimination of boiler-plate code, something found in JDBC
* Unlike JDBC API, Hibernate supports associations, collections, and inheritances
* HQL (Hibernate Query Language) is closer to Java and is more object-oriented
* Developers don’t need to write code to store and load data into the database
* Hibernate enables faster application development

### 5. What is “persistence”?

In the context of Java, persistence describes data and objects that last beyond the process used to create them.

### 6. Name some databases that Hibernate supports.

Hibernate supports databases like:

* DB2/NT
* FrontBase
* HSQL Database Engine
* Informix Dynamic Server
* Microsoft SQL Server Database
* MySQL
* Oracle
* PostgreSQL
* SQL Server
* Sybase

### 7. What is HQL?

HQL stands for Hibernate Query Language, a powerful object-oriented language independent of the database. It’s like [SQL](https://www.simplilearn.com/tutorials/sql-tutorial/what-is-sql), except that it uses objects instead of table names. HQL is a very simple, efficient, and flexible query language used to do various operations on a relational database without the need for complex database queries.

### 8. Name the four ORM levels in Hibernate.

Hibernate’s four ORM levels are:

* Full Object Mapping
* Light Object Mapping
* Medium Object Mapping
* Pure Relational

## **Become a Software Development Professional**

#### Full Stack Java Developer

* + Kickstart Full Stack Java Developer career with industry-aligned curriculum by experts
  + Hands-on practice through 20+ projects, assessments, and tests

6 months

[**View Program**](javascript:void(0))

#### Full Stack Web Developer - MEAN Stack

* + Comprehensive Blended Learning program
  + 8X higher interaction in live online classes conducted by industry experts

11 months

[**View Program**](javascript:void(0))

Not sure what you’re looking for?[View all Related Programs](https://www.simplilearn.com/mobile-and-software-development?source=InpageBannerCategory)

### 9. What is a Session in Hibernate?

A Session in Hibernate is a lightweight, non-thread-safe object representing a single unit of work with the database. It is used to perform CRUD operations on persistent objects.

### 10. What is a SessionFactory?

A SessionFactory in Hibernate is a heavyweight and thread-safe object used to create and manage multiple sessions across an application. It is also configuring and managing the underlying connection pooling and caching mechanisms.

### 11. What do you think about the statement - "session being a thread-safe object"?

The statement is incorrect. A Session in Hibernate is a lightweight and non-thread-safe object and should not be shared across multiple threads.

### 12. What is the difference between first-level cache and second-level cache?

The first level cache is associated with a Session and is used to store the currently loaded objects in memory. The second level cache is associated with a SessionFactory and is used to hold objects across multiple sessions, thereby reducing the number of database queries needed.

### 13. What can you tell about the Hibernate Configuration File?

The Hibernate Configuration File (hibernate.cfg.xml) is an [XML file](https://www.simplilearn.com/tutorials/programming-tutorial/what-is-xml) used to configure the basic settings of Hibernate, such as the database URL, username, password, and dialect. It also contains the mapping information of the persistent classes and the resources required for connection pooling.

### 14. How do you create an immutable class in hibernate?

To create an immutable class in Hibernate, you should mark all class properties as 'final' and provide only getter methods for them without any setters. Additionally, you can use the 'mutable' attribute of the 'property' or 'component' element in the Hibernate mapping file to make a property or component immutable.

### 15. Can you explain the concept behind Hibernate Inheritance Mapping?

Hibernate Inheritance Mapping is used to represent the inheritance relationships between classes in a relational database. It allows the developer to map a single table to multiple classes using techniques like table per class hierarchy, table per subclass, and table per concrete class.

### 16. Is hibernate prone to SQL injection attacks?

Hibernate, by itself, is not prone to [SQL injection attacks](https://www.simplilearn.com/tutorials/cyber-security-tutorial/what-is-sql-injection). However, the application may be vulnerable to SQL injection attacks if user input is concatenated with the HQL or Criteria queries.

## **Intermediate Level Hibernate Interview Questions**

Let’s push the difficulty level up a few notches with this set of eight moderately challenging Hibernate interview questions and answers.

### 17. Name Hibernate’s five collection types used in one-to-many relationship mappings.

The five collection types are:

* Array
* Bag
* List
* Map
* Set

### 18. What is “dirty checking”?

The dirty checking feature helps developers and users avoid time-consuming write actions, thereby reducing database write times. Dirty checking changes or updates only the fields that require action, while keeping the rest of the fields untouched and unchanged.

### 19. What is Hibernate’s default cache service?

Hibernate’s default cache service is EHCache, though the framework additionally supports OSCache, SWARMCache, and TreeCache.

### 20. What is Light Object Mapping?

Light Object Mapping is one of the more valuable levels of ORM quality. This approach uses specific design patterns to hide the syntax from business logic. All entities are represented as classes and mapped manually. The Light Object Mapping approach works well with applications that have fewer entities and applications that use metadata-driven data models.

### 21. List and describe the Hibernate framework’s essential interfaces.

Hibernate’s important interfaces are:

* SessionFactory (org.hibernate.SessionFactory). SessionFactory is an immutable thread-safe cache of compiled mappings meant for a single database. After users initialize SessionFactory once, they can cache and reuse it. SessionFactory is designed to return the session objects for database operations.
* Session (org.hibernate.Session). A session is a single-threaded, short-lived object that represents a dialogue between the persistent store and the application. It is the interface that exists between the Hibernate framework and the Java application code, providing methods for CRUD operations. A session should be opened only when required, then closed as soon as the user is finished.
* Transaction (org.hibernate.transaction). The transaction is a single-threaded, short-lived object that the application uses to specify atomic units of work.

### 22. What is lazy loading?

Lazy loading is a technique where objects are loaded as needed, instead of an entire page, for example. This technique became default since Hibernate version 3.

### 23. What are the concurrency strategies?

Concurrency strategies are mediators responsible for storing and retrieving cached items. When enabling a second-level cache, the developer must decide which cache concurrency to implement for each persistent class and collection.

The concurrency strategies are:

* Non-strict-Read-Write: This strategy works with data that can be altered and can tolerate a small chance of stale data. This strategy offers no guarantee of consistency between the database and the cache.
* Read-Only: This strategy works best with data that can’t be changed, and consequently, is only used to reference data.
* Transactional: This strategy is used primarily for read-mostly data in cases where it’s essential to prevent stale data in concurrent transactions, in those rare instances of an update.
* Read-Write: This strategy is like the transactional strategy.

### 24. Define Hibernate’s validator framework.

Data validation is an integral part of any application and is used in the presentation layer when using JavaScript and server-side code before processing. Validation is a cross-cutting task that occurs before making it persistent so that it adheres to the correct format.

### 25. Explain hibernate mapping file.

A hibernate mapping file is an XML file that defines the relationship between a Java class and a database table. It specifies the mapping of the class properties to the table columns and describes any additional connections between the class and other classes or tables.

### 26. What are the most commonly used annotations available to support hibernate mapping?

The most commonly used annotations for hibernate mapping are @Entity, @Table, @Column, @Id, @GeneratedValue, @ManyToOne, and @OneToMany.

### 27. Explain Hibernate architecture

Hibernate architecture consists of several components, such as the Session Factory, Session, Transaction, and Query. For example, the Session Factory is responsible for creating and managing sessions, interacting with the database, and executing queries.

### 28. Can you tell the difference between the getCurrentSession and openSession methods?

The getCurrentSession method creates or retrieves the current Session from the current session context. In contrast, the open Session process begins a new session that is not bound to the current context.

## **Become a Software Development Professional**

#### Full Stack Java Developer

* + Kickstart Full Stack Java Developer career with industry-aligned curriculum by experts
  + Hands-on practice through 20+ projects, assessments, and tests

6 months

[**View Program**](javascript:void(0))

#### Full Stack Web Developer - MEAN Stack

* + Comprehensive Blended Learning program
  + 8X higher interaction in live online classes conducted by industry experts

11 months

[**View Program**](javascript:void(0))

Not sure what you’re looking for?[View all Related Programs](https://www.simplilearn.com/mobile-and-software-development?source=InpageBannerCategory)

### 29. Differentiate between save() and saveOrUpdate() methods in hibernate Session.

The save() method is used to persist a new object in the database. In contrast, the saveOrUpdate() method is used to continue a new object or update an existing object in the database.

### 30. Differentiate between get() and load() in Hibernate session

The get() method retrieves an object from the database by its [primary key](https://www.simplilearn.com/tutorials/sql-tutorial/primary-key-in-sql) and throws an exception if the object is not found. The load() method is also used to retrieve an object from the database by its primary key, but it returns a proxy object if it is not found.

### 31. What are the criteria for API in hibernate?

The criteria API in hibernate is a programmatic way of creating and executing queries. It allows developers to build complex queries using a fluent interface rather than writing raw SQL.

### 32. Can you tell me something about one too many associations and how we can use them in Hibernate?

A one-to-many association occurs when one entity is associated with multiple other entities. In Hibernate, one-to-many associations can be implemented using the @OneToMany annotation, and the @JoinColumn annotation is used to specify the column that will be used to join the two entities.

### 33. What are Many to Many associations?

A many-to-many association occurs when multiple entities are associated with various other entities. In Hibernate, many-to-many associations can be implemented using the @ManyToMany annotation, and a join table is used to store the relationship between the two entities.

### 34. What does Session.lock() method in hibernate do?

The SessionSession.lock() method is used to acquire a lock on an object in the current Session. This is useful for preventing concurrent updates to the same thing.

### 35. What is hibernate caching?

Hibernate caching refers to storing data in memory to retrieve it quickly without hitting the database again. This improves performance and reduces the load on the database.

### 36. Types of Hibernate Caching

There are two types of hibernate caching: first-level caching and second-level caching. First-level caching is enabled by default and is associated with the Session object. Second-level caching is optional and is associated with the SessionFactory object.

### 37. When is the merge() method of the hibernate Session useful?

The merge() method of the hibernate Session is useful when you want to update an existing object in the database without reattaching it to the Session.

### 38. Collection mapping can be done using One-to-One and Many-to-One Associations. What do you think?

Yes, collection mapping can be done using both One-to-One and Many-to-One associations. One-to-One association is used when one object is associated with one other object. At the same time, the Many-to-One association is used when one object is associated with multiple other objects.

### 39. Can you tell the difference between setMaxResults() and setFetchSize() of Query?

The setMaxResults() method limits the number of results returned by a query, while the setFetchSize() method controls the number of rows retrieved from the database at a time. setMaxResults() is used to limit the total number of results returned, while setFetchSize() controls the number of rows retrieved at a time to avoid memory issues.

### 40. Does Hibernate support Native SQL Queries?

Yes, Hibernate supports Native SQL Queries, which allow you to use SQL statements directly to interact with the database. This can be useful when you want to perform complex queries that are impossible with HQL or Criteria API.

Now that we have learned some of the intermediate level Hibernate interview questions, let us next increase the difficulty level and look into some of the advanced level Hibernate interview questions and answers.

## **Advanced Level Hibernate Interview Questions**

We round out the Hibernate interview questions with eight expert questions.

### 41. What design patterns does the Hibernate framework use?

Some design patterns include:

* Data Mapper, which moves data between objects and a database, keeping them independent of each other and the mapper
* Domain Model Pattern, which is a domain object model that incorporates both behavior and data
* Proxy Pattern, for lazy loading
* Factory pattern in SessionFactory

### 42. What is Hibernate tuning?

The process of Hibernate tuning is designed to optimize Hibernate applications’ performance. The three strategies are:

* SQL Optimization
* Session Management
* Data Caching

### 43. Name the states that a persistent entity exists in.

Persistent entities exist in only three states:

* Transient
* Persistent
* Detached

### 44. How can you view the Hibernate-generated SQL on a console?

To enable viewing SQL on a console for debugging purposes, you must add the following in the Hibernate configuration file:

1 <property name="show\_sql">true</property>

### 45. What’s the difference between Session and SessionFactory?

A Session provides the first-level cache and is a single-threaded, short-lived object. A SessionFactory provides the second-level cache and is immutable and shared by all Sessions. It lives until Hibernate is running.

### 46. How many ways can an object be fetched from Hibernate’s database?

There are four ways to fetch objects from Hibernate’s database:

* Criteria API
* HQL
* The identifier
* Standard SQL

### 47. How many ways can you disable Hibernate’s second-level cache?

There are three ways to disable the cache:

* By setting hibernate. cache. use\_second\_level\_cache property to false
* By using CACHEMODE.IGNORE
* Using a cache provider such as org.hibernate.cache.NoCacheProvider

### 48. Describe the differences between Hibernate’s transient, persistent, and detached states.

Here is how the states differ:

* Transient. This state describes new objects that are created in Java but not associated with a Hibernate session.
* Persistent. This state describes objects associated with a Hibernate session.
* Detached. This state describes an object that was formerly Persistent and associated with a Hibernate session. Developers can reattach the object to a Hibernate session by using either update() or saveOrUpdate().

### 49. What happens when the no-args constructor is absent in the Entity bean?

If the no-args constructor is absent in the Entity bean, Hibernate cannot instantiate the object and throw an exception. Therefore, it is always recommended to have a no-args constructor in the Entity bean for Hibernate to work properly.

### 50. Can we declare the Entity class final?

No, the Entity class cannot be declared final, as Hibernate uses runtime proxies to enhance the class for persistence. Therefore, a final class cannot be subclassed and thus cannot be proxied.

### 51. Explain Query Cache

The Query Cache is a second-level cache that stores the results of a query so that they can be reused later. It improves performance by avoiding multiple times hitting the database for the same Query.

### 52. Can you say something about the N+1 SELECT problem in Hibernate?

The N+1 SELECT problem in Hibernate refers to a single SELECT statement being executed to retrieve the parent object. Then N SELECT statements are executed to retrieve the child objects, resulting in poor performance.

### 53. How to solve the N+1 SELECT problem in Hibernate?

The N+1 SELECT problem can be solved using fetching strategies such as eager loading, lazy loading, or batch fetching. Another solution is to use the JOIN FETCH clause in your HQL or JPQL Query.

### 54. What is a Single Table Strategy?

The Single Table Strategy maps multiple classes that inherit from a single root class to a single table in the database. Each row in the table represents an instance of one of the classes in the hierarchy.

## **Become a Software Development Professional**

#### Full Stack Java Developer

* + Kickstart Full Stack Java Developer career with industry-aligned curriculum by experts
  + Hands-on practice through 20+ projects, assessments, and tests

6 months

[**View Program**](javascript:void(0))

#### Full Stack Web Developer - MEAN Stack

* + Comprehensive Blended Learning program
  + 8X higher interaction in live online classes conducted by industry experts

11 months

[**View Program**](javascript:void(0))

Not sure what you’re looking for?[View all Related Programs](https://www.simplilearn.com/mobile-and-software-development?source=InpageBannerCategory)

### 55. Can you say something about the Table Per Class Strategy?

The Table Per Class Strategy maps classes in an inheritance hierarchy to separate tables in the database. Each class in the hierarchy has its table, and the data for that class is stored in the corresponding table.

### 56. Can you say something about Named SQL Query?

Named SQL Query is a feature in Hibernate that allows you to define a named query and reuse it throughout your application. It is defined in the mapping file or by using the @NamedQuery annotation.

### 57. What are the benefits of NamedQuery?

The benefits of NamedQuery include the following:

* Improved code maintainability and readability.
* Reduced code duplication.
* Ability to easily change the Query without affecting its code.

It also improves performance when the same Query is used multiple times by caching the query results.

**SPRING MVC**

1) What is MVC?

The MVC (Model-View-Controller) is a software architectural design pattern. It separates the functionality of an application into three interconnected parts - Model, View, and Controller. This approach facilitates the reusability of the code and parallel development.

2) What is Spring MVC?

A Spring MVC is a Java Framework which is used to develop dynamic web applications. It implements all the basic features of a core spring framework like Inversion of Control and Dependency Injection. It follows the Model-View-Controller design pattern.



Here,

* **Model** - A model contains the data of the application. Data can be a single object or a collection of objects.
* **Controller** - A controller contains the business logic of an application. Here, the @Controller annotation is used to mark the class as the controller.
* **View** - A view represents the provided information in a particular format. So, we can create a view page by using view technologies like JSP+JSTL, Apache Velocity, Thymeleaf, and FreeMarker.

3) What is the front controller of Spring MVC?

The front controller is a **DispatcherServlet** class present in **org.springframework.web.servlet** package. It dispatches the request to the appropriate controller and manages the flow of the application. It is required to specify the **DispatcherServlet** class in the web.xml file.

4) Explain the flow of Spring MVC?



* Once the request has been generated, it intercepted by the DispatcherServlet that works as the front controller.
* The DispatcherServlet gets an entry of handler mapping from the XML file and forwards the request to the controller.
* The controller returns an object of ModelAndView.
* The DispatcherServlet checks the entry of view resolver in the XML file and invokes the specified view component.

5) What are the advantages of Spring MVC Framework?

The following are the advantages of Spring MVC Framework : -

* **Separate roles** - The Spring MVC separates the application into three interconnected layers where each layer has its role.
* **Light-weight** - It uses light-weight servlet container to develop and deploy your application.
* **Powerful Configuration** - It provides a robust configuration for both framework and application classes that includes easy referencing across contexts, such as from web controllers to business objects and validators.
* **Rapid development** - The Spring MVC facilitates fast and parallel development.
* **Reusable business code** - Instead of creating new objects, it allows us to use the existing business objects.
* **Flexible Mapping** - It provides the specific annotations that easily redirect the page.

6) What does an additional configuration file contain in Spring MVC application?

The Spring MVC application contains an additional configuration file that contains the properties information. This file can be created either in the form of **an xml** file or **properties** file. In this file, we generally define the base-package and view resolver where **DispatcherServlet** searches for the controller classes and view components path. However, it can also contain various other configuration properties.

7) What is an InternalResourceViewResolver in Spring MVC?

The **InternalResourceViewResolver** is a class which is used to resolve internal view in Spring MVC. Here, you can define the properties like prefix and suffix where prefix contains the location of view page and suffix contains the extension of view page. For example:-

1. **<bean** id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver"**>**
2. **<property** name="prefix" value="/WEB-INF/jsp/"**></property>**
3. **<property** name="suffix" value=".jsp"**></property>**
4. **</bean>**

8) How to declare a class as a controller class in Spring MVC?

The @Controller annotation is used to declare a class as a controller class. It is required to specify this annotation on the class name. For example:-

1. @Controller
2. class Demo
3. {
5. }

9) How to map controller class and its methods with URL?

The **@RequestMapping** annotation is used to map the controller class and its methods. You can specify this annotation on the class name as well as method name with a particular URL that represents the path of the requested page. For example:-

1. @Controller
2. @RequestMapping("/ form")
3. class Demo
4. {
5. @RequestMapping("/show")
6. public String display()
7. {
9. }
11. }

10) Name the annotations used to handle different types of incoming HTTP request methods?

The following annotations are used to handle different types of incoming HTTP request methods: -

* @GetMapping
* @PostMapping
* @PutMapping
* @PatchMapping
* @DeleteMapping

11) What is the purpose of @PathVariable annotation in Spring MVC?

The @PathVariable annotation is used to extract the value of the URI template. It is passed within the parameters of the handler method.

For example :-

1. @RequestMapping("/show/{id}")
2. public String handler(@PathVariable("id") String s, Model map)
3. {
4. }

12) What is the role of @ResponseBody annotation in Spring MVC?

The @ResponseBody annotation is used to serialize the returned object automatically in JSON and bind it with the Http response body. Here, it not required to invoke the model.

For example :-

1. @RequestMapping("/show")
2. @ResponseBody
3. public ResponseHandler display(
4. @RequestBody ShowForm form) {
5. return new ResponseHandler("display form");
6. }
7. }

13) What is the role of the Model interface in Spring MVC?

The **Model** interface works as a container that contains the data of the application. Here, data can be in any form such as objects, strings, information from the database, etc.

[click here for more details](https://www.javatpoint.com/spring-mvc-model-interface)

14) What do you mean by ModelAndView in Spring MVC?

The **ModelAndView** is a class that holds both Model and View where the model represents the data, and view represents the representation of that data. This class returns the model and view in the single return value.

15) What is ModelMap in Spring MVC?

The **ModelMap** is a class that provides the implementation of Map. It extends the LinkedHashMap class. It facilitates to pass a collection of values as if they were within a Map.

16) What are the ways of reading data from the form in Spring MVC?

The following ways to read the data from the form are: -

ADVERTISEMENT

* **HttpServletRequest interface** - The **HttpServletRequest** is a java interface present in javax.servlet.http package. Like Servlets, you can use HttpServletRequest in Spring to read the HTML form data provided by the user.
* **@RequestParam annotation** - The **@RequestParam** annotation reads the form data and binds it automatically to the parameter present in the provided method.
* **@ModelAttribute annotation** - The **@ModelAttribute** annotation binds a method parameter or its return value to a named model attribute.

17) What is Spring MVC form tag library?

The Spring MVC form tags can be seen as data binding-aware tags that can automatically set data to Java object/bean and also retrieve from it. These tags are the configurable and reusable building blocks for a web page. It provides view technologies, an easy way to develop, read, and maintain the data.

[click here for more details](https://www.javatpoint.com/spring-mvc-form-tag-library)

18) What do you understand by validations in Spring MVC?

The validation is one of the most important features of Spring MVC, that is used to restrict the input provided by the user. To validate the user's input, it is required to use the Spring 4 or higher version and Bean Validation API. Spring validations can validate both server-side as well as client-side applications.

19) What is Bean Validation API?

The **Bean Validation API** is a Java specification which is used to apply constraints on object model via annotations. Here, we can validate a length, number, regular expression, etc. Apart from that, we can also provide custom validations.

As Bean Validation API is just a specification, it requires an implementation. So, for that, it uses Hibernate Validator. The Hibernate Validator is a fully compliant JSR-303/309 implementation that allows to express and validate application constraints.

20) What is the use of @Valid annotation in Spring MVC?

The **@Valid** annotation is used to apply validation rules on the provided object.

21) What is the purpose of BindingResult in Spring MVC validations?

The **BindingResult** is an interface that contains the information of validations. For example :-

1. @RequestMapping("/helloagain")
2. public String submitForm( @Valid @ModelAttribute("emp") Employee e, BindingResult br)
3. {
4. if(br.hasErrors())
5. {
6. return "viewpage";
7. }
8. else
9. {
10. return "final";
11. }
12. }

22) How to validate user's input within a number range in Spring MVC?

In Spring MVC Validation, we can validate the user's input within a number range by using the following annotations: -

* **@Min annotation** - It is required to pass an integer value with @Min annotation. The user input must be equal to or greater than this value.
* **@Max annotation** - It is required to pass an integer value with @Max annotation. The user input must be equal to or smaller than this value.

[click here for more details](https://www.javatpoint.com/spring-mvc-number-validation)

23) How to validate the user input in a particular sequence in Spring MVC?

The Spring MVC Validation allows us to validate the user input in a particular sequence by using @Pattern annotation. Here, we can provide the required regular expression to **regexp** attribute and pass it with the annotation.

[click here for more details](https://www.javatpoint.com/spring-mvc-regular-expression-validation)

24) What is the purpose of custom validations in Spring MVC?

The Spring MVC framework allows us to perform custom validations. In such a case, we declare our own annotations. We can perform validation based on own business logic.

[click here for more details](https://www.javatpoint.com/spring-mvc-custom-validation)

25) What do you understand by Spring MVC Tiles?

The Spring provides integration support with apache tiles framework. So we can manage the layout of the Spring MVC application with the help of spring tiles support. The following are the advantages of Tiles support in Spring MVC: -

* **Reusability:** We can reuse a single component in multiple pages like header and footer components.
* **Centralized control:** We can control the layout of the page by a single template page only.
* **Easy to change the layout:** By the help of a single template page, we can change the layout of the page anytime. So, your website can easily adopt new technologies such as bootstrap and jQuery.

**SPRING**

### 1) What is Spring?

It is a lightweight, loosely coupled, and integrated framework for developing enterprise applications in java.

### 2) What are the advantages of spring framework?

1. Predefined Templates
2. Loose Coupling
3. Easy to test
4. Lightweight
5. Fast Development
6. Powerful Abstraction
7. Declarative support

[More details...](https://www.javatpoint.com/spring-tutorial)

### 3) What are the modules of spring framework?

1. Test
2. Spring Core Container
3. AOP, Aspects and Instrumentation
4. Data Access/Integration
5. Web

[More details...](https://www.javatpoint.com/spring-modules)

### 4) What is IOC and DI?

IOC (Inversion of Control) and DI (Dependency Injection) is a design pattern to provide loose coupling. It removes the dependency from the program.

Let's write a code without following IOC and DI.

1. **public** **class** Employee{
2. Address address;
3. Employee(){
4. address=**new** Address();//creating instance
5. }
6. }

Now, there is dependency between Employee and Address because Employee is forced to use the same address instance.

Let's write the IOC or DI code.

1. **public** **class** Employee{
2. Address address;
3. Employee(Address address){
4. **this**.address=address;//not creating instance
5. }
6. }

Now, there is no dependency between Employee and Address because Employee is not forced to use the same address instance. It can use any address instance.

### 5) What is the role of IOC container in spring?

IOC container is responsible to:

ADVERTISEMENT

ADVERTISEMENT

* create the instance
* configure the instance, and
* assemble the dependencies

[More details...](https://www.javatpoint.com/ioc-container)

### 6) What are the types of IOC container in spring?

There are two types of IOC containers in spring framework.

1. BeanFactory
2. ApplicationContext

[More details...](https://www.javatpoint.com/ioc-container)

### 7) What is the difference between BeanFactory and ApplicationContext?

BeanFactory is the **basic container** whereas ApplicationContext is the **advanced container**. ApplicationContext extends the BeanFactory interface. ApplicationContext provides more facilities than BeanFactory such as integration with spring AOP, message resource handling for i18n etc.

ADVERTISEMENT

ADVERTISEMENT

### 8) What is the difference between constructor injection and setter injection?

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

[More details...](https://www.javatpoint.com/difference-between-constructor-and-setter-injection)

### 9) 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 |

The "autodetect" mode is deprecated since spring 3.

### 10) What are the different bean scopes in spring?

There are 5 bean scopes in spring framework.

|  |  |  |
| --- | --- | --- |
| **No.** | **Scope** | **Description** |
| 1) | singleton | The bean instance will be only once and same instance will be returned by the IOC container. It is the default scope. |
| 2) | prototype | The bean instance will be created each time when requested. |
| 3) | request | The bean instance will be created per HTTP request. |
| 4) | session | The bean instance will be created per HTTP session. |
| 5) | globalsession | The bean instance will be created per HTTP global session. It can be used in portlet context only. |

### 11) In which scenario, you will use singleton and prototype scope?

Singleton scope should be used with EJB **stateless session bean** and prototype scope with EJB **stateful session bean**.

### 12) What are the transaction management supports provided by spring?

Spring framework provides two type of transaction management supports:

ADVERTISEMENT

ADVERTISEMENT

1. **Programmatic Transaction Management**: should be used for few transaction operations.
2. **Declarative Transaction Management**: should be used for many transaction operations.

## **» Spring JDBC Interview Questions**

### 13) What are the advantages of JdbcTemplate in spring?

**Less code**: By using the JdbcTemplate class, you don't need to create connection,statement,start transaction,commit transaction and close connection to execute different queries. You can execute the query directly.

[More details...](https://www.javatpoint.com/spring-JdbcTemplate-tutorial)

### 14) What are classes for spring JDBC API?

1. JdbcTemplate
2. SimpleJdbcTemplate
3. NamedParameterJdbcTemplate
4. SimpleJdbcInsert
5. SimpleJdbcCall

[More details...](https://www.javatpoint.com/spring-JdbcTemplate-tutorial)

### 15) How can you fetch records by spring JdbcTemplate?

ADVERTISEMENT

ADVERTISEMENT

You can fetch records from the database by the **query method of JdbcTemplate**. There are two interfaces to do this:

1. [ResultSetExtractor](https://www.javatpoint.com/ResultSetExtractor-example)
2. [RowMapper](https://www.javatpoint.com/RowMapper-example)

### 16) What is the advantage of NamedParameterJdbcTemplate?

NamedParameterJdbcTemplate class is used to pass value to the named parameter. A named parameter is better than ? (question mark of PreparedStatement).

ADVERTISEMENT

ADVERTISEMENT

It is **better to remember**.

[More details...](https://www.javatpoint.com/spring-NamedParameterJdbcTemplate-example)

### 17) What is the advantage of SimpleJdbcTemplate?

The **SimpleJdbcTemplate** supports the feature of var-args and autoboxing.

[More details...](https://www.javatpoint.com/spring-SimpleJdbcTemplate-example)

## **» Spring AOP Interview Questions**

### 18) What is AOP?

ADVERTISEMENT

ADVERTISEMENT

AOP is an acronym for Aspect Oriented Programming. It is a methodology that divides the program logic into pieces or parts or concerns.

It increases the modularity and the key unit is Aspect.

[More details...](https://www.javatpoint.com/spring-aop-tutorial)

### 19) What are the advantages of spring AOP?

AOP enables you to dynamically add or remove concern before or after the business logic. It is **pluggable** and **easy to maintain**.

[More details...](https://www.javatpoint.com/spring-aop-tutorial)

### 20) What are the AOP terminology?

AOP terminologies or concepts are as follows:

* JoinPoint
* Advice
* Pointcut
* Aspect
* Introduction
* Target Object
* Interceptor
* AOP Proxy
* Weaving

[More details...](https://www.javatpoint.com/spring-aop-tutorial)

### 21) What is JoinPoint?

JoinPoint is any point in your program such as field access, method execution, exception handling etc.

### 22) Does spring framework support all JoinPoints?

No, spring framework supports method execution joinpoint only.

### 23) What is Advice?

ADVERTISEMENT

ADVERTISEMENT

Advice represents action taken by aspect.

ADVERTISEMENT

ADVERTISEMENT

### 24) What are the types of advice in AOP?

There are 5 types of advices in spring AOP.

1. Before Advice
2. After Advice
3. After Returning Advice
4. Throws Advice
5. Around Advice

### 25) What is Pointcut?

Pointcut is expression language of Spring AOP.

### 26) What is Aspect?

Aspect is a class in spring AOP that contains advices and joinpoints.

### 27) What is Introduction?

Introduction represents introduction of new fields and methods for a type.

### 28) What is target object?

Target Object is a proxy object that is advised by one or more aspects.

### 29) What is interceptor?

Interceptor is a class like aspect that contains one advice only.

ADVERTISEMENT

ADVERTISEMENT

### 30) What is weaving?

Weaving is a process of linking aspect with other application.

### 31) Does spring perform weaving at compile time?

No, spring framework performs weaving at runtime.

### 32) What are the AOP implementation?

There are 3 AOP implementation.

1. Spring AOP
2. Apache AspectJ
3. JBoss AOP

## **» Spring MVC Interview Questions**

### 33) What is the front controller class of Spring MVC?

The **DispatcherServlet** class works as the front controller in Spring MVC.

[More details...](https://www.javatpoint.com/spring-3-mvc-tutorial)

### 34) What does @Controller annotation?

The **@Controller** annotation marks the class as controller class. It is applied on the class.

### 35) What does @RequestMapping annotation?

The **@RequestMapping** annotation maps the request with the method. It is applied on the method.

ADVERTISEMENT

ADVERTISEMENT

### 36) What does the ViewResolver class?

The **View Resolver** class resolves the view component to be invoked for the request. It defines prefix and suffix properties to resolve the view component.

### 37) Which ViewResolver class is widely used?

The **org.springframework.web.servlet.view.InternalResourceViewResolver** class is widely used.

### 38) Does spring MVC provide validation support?

Yes.

**SPRING SECURITY**

## **Explain Spring Security Architecture using Spring Boot?**

Let us understand how Spring Security Works.  
A diagram of a security system

Description automatically generated

## **How is Security mechanism implemented using Spring:**

The default implementation in Spring Security is called Provider Manager and rather than handling the authentication request itself, it delegates to a list of configured Authentication Provider s, each of which is queried in turn to see if it can perform the authentication.

## **What is OAuth2 Authorization code grant type?**

OAuth (Open Authorization) is a simple way to publish and interact with protected data.  
It is an open standard for token-based authentication and authorization on the Internet. It allows an end user's account information to be used by third-party services.

The OAuth specification describes five grants for acquiring an access token:

* Authorization code grant
* Implicit grant
* Resource owner credentials grant
* Client credentials grant
* Refresh token grant.

## **Using Spring Boot Security how to refresh expired JSON Web Token?**

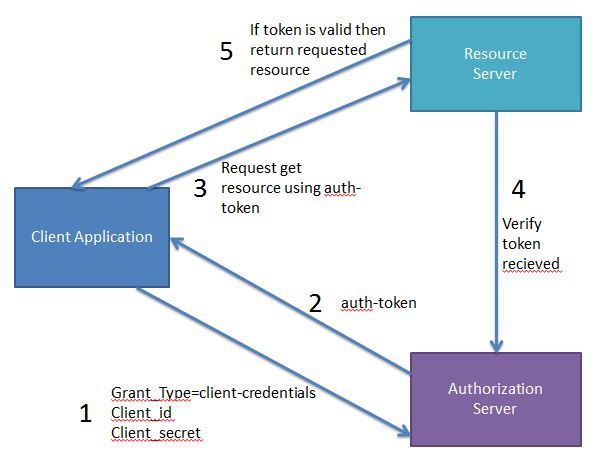
We will be working on a solution where if the user receives JWT expired exception, then he can call another API with the expired token. A new token will then provide the user which he can use for future interactions. A diagram of a server

Description automatically generated

## **What is JWT? How to implement it using Spring Boot Security?**

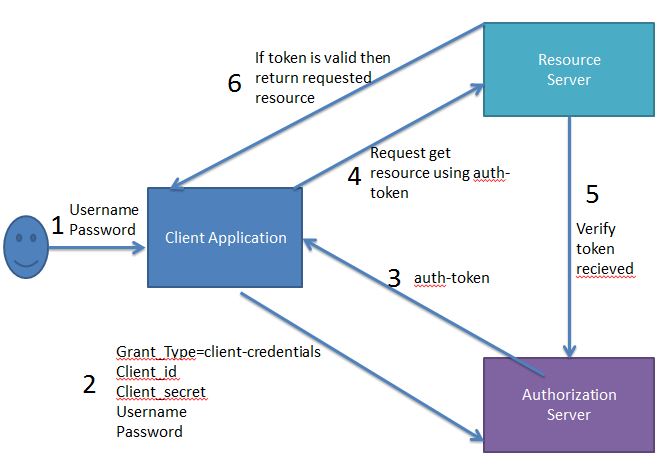
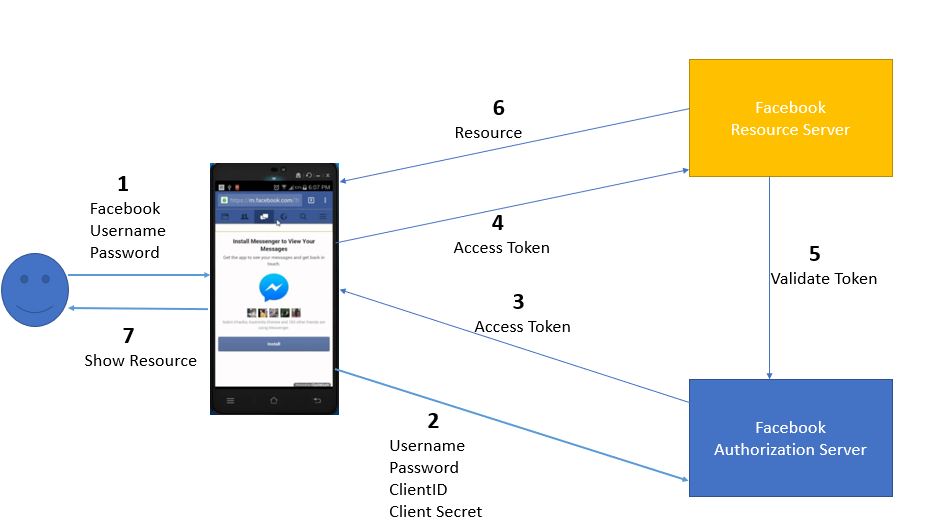
* It stands for JSON Web Tokens (JWT)
* JWTs allow you to transmit information securely between parties as a compact, self-contained, and digitally signed JSON object.

## **What is OAuth2 Client Credentials Grant? How to implement it using Spring Boot Security?**

The Client Credentials Grant involves machine to machine authentication. In case of Client credentials grant type, the user has no role to play.

This type of Authentication does not involve any end-user. Unlike Authorization Grant where the end user had to authenticate himself using Authorization Server like Gmail, here the machine itself authenticates itself to access a protected resource.

## **What is OAuth2 Password Grant? How to implement it using Spring Boot Security**

In case of Password grant type the user triggers the client to get some resource. While doing so it passes the username and password to the client. The client then communicates with the authorization server using the provided username, password and also its own clientId and clientSecret to get the access token. Using this access token it then gets the required resource from the resource server.  
  
The real life example of Password grant will be you doing a login to you facebook account using its mobile application. Here the user will have to specify the facebook credentials to the app. also the app will be having its own client id and client secret.  


## **How to use Form Login Authentication using Spring Boot**

We make use of Spring Boot Security to get default login page and authentication users.

@Override

protected void configure(HttpSecurity http) throws Exception {

http.authorizeRequests().antMatchers("/").permitAll().antMatchers("/welcome")

.hasAnyRole("USER", "ADMIN").antMatchers("/getEmployees").hasAnyRole("USER", "ADMIN")

.antMatchers("/addNewEmployee").hasAnyRole("ADMIN").anyRequest().authenticated().and().formLogin()

.permitAll().and().logout().permitAll();

http.csrf().disable();

}

## **How to create Custom Login Page using Spring Boot Security?**

We can create our own custom login page and use it for authentication.

@Override

protected void configure(HttpSecurity http) throws Exception {

http.authorizeRequests().antMatchers("/").permitAll().antMatchers("/welcome").hasAnyRole("USER", "ADMIN")

.antMatchers("/getEmployees").hasAnyRole("USER", "ADMIN").antMatchers("/addNewEmployee")

.hasAnyRole("ADMIN").anyRequest().authenticated()

.and().formLogin().**loginPage("/login")**.permitAll()

.and().logout().permitAll();

http.csrf().disable();

}

## **How to do authentication against database tables using Spring Boot Security?**

Spring Authentication using username, password and authorization using roles can be done using either

* In Memory Configuration -
* @Autowired
* public void configureGlobal(AuthenticationManagerBuilder authenticationMgr) throws Exception {
* authenticationMgr.inMemoryAuthentication().withUser("employee").password("employee")
* .authorities("ROLE\_USER").and().withUser("javainuse").password("javainuse")
* .authorities("ROLE\_USER", "ROLE\_ADMIN");
* }

* Database Authentication-
* @Autowired
* public void configAuthentication(AuthenticationManagerBuilder auth) throws Exception {
* auth.jdbcAuthentication().dataSource(dataSource);
* }

## **What is the use of Spring Boot Security AuthenticationHandler class?**

In some scenarios we might want to redirect different users to different pages depending on the roles assigned to the users.  
For example we might want users with role USER to be redirected to the welcome page, while users with role ADMIN to be redirected to the add employee page.  
We will be making use of the AuthenticationSuccessHandler.

@Override

protected void configure(HttpSecurity http) throws Exception {

http.authorizeRequests().antMatchers("/").permitAll().antMatchers("/welcome").hasAnyRole("USER", "ADMIN")

.antMatchers("/getEmployees").hasAnyRole("USER", "ADMIN").antMatchers("/addNewEmployee")

.hasAnyRole("ADMIN").anyRequest().authenticated()

.and().formLogin().**successHandler(successHandler)**

.loginPage("/login").permitAll().and().logout().permitAll();

http.csrf().disable();

}

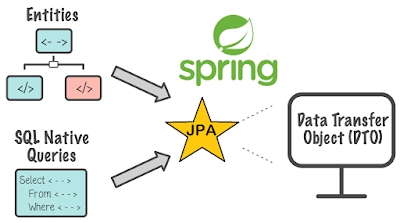
## **What is the difference between ROLE\_USER and ROLE\_ANONYMOUS in a Spring intercept url configuration?**

* **ROLE\_ANONYMOUS** is the default role assigned to an unauthenticated (anonymous) user when a configuration uses Spring Security's "anonymous authentication" filter . This is enabled by default. However, it is probably clearer if you use the expression isAnonymous() instead, which has the same meaning.
* **ROLE\_USER** has no meaning unless you assign this role to your users when they are authenticated (you are in charge of loading the roles (authorities) for an authenticated user). It isn't a name that is built in to Spring Security's infrastructure. In the given example, presumably that role is assigned to an authenticated user.

**SPRING JPA**

**1. What is JPA?**  
Answer: JPA stands for Java Persistence API. It is a Java specification used to persist data between the relational database and Java objects. It acts as a bridge between object-oriented domain models and relational databases.  Since interaction with database from Java application is very common, JPA was created to standardize this interaction.

There are many popular JPA implementations available in the Java world like Hibernate. You can further see these [**Spring Data JPA using Hibernate course**](https://medium.com/javarevisited/5-best-spring-data-jpa-courses-for-java-developers-45e6438be3c9)to learn more about how to use Hibernate with Spring Data JPA in Java application.

[](https://javarevisited.blogspot.com/2020/06/10-advanced-spring-framework-courses.html)

**2. What are some advantages of using JPA?**

Answer: Here are some advantages of Java Persistence API or JPA:

* JPA reduces the burden of interacting with databases.
* Annotation in JPA reduces the cost of creating a definition file.
* It is user-friendly.
* JPA providers help merge applications.

**3. What is the Spring data repository? (**[**answer**](https://javarevisited.blogspot.com/2021/10/what-is-spring-data-repository.html)**)**  
Answer: Spring data repository is a very important feature of JPA. It helps in reducing a lot of boilerplate code. Moreover, it decreases the chance of errors significantly. This is also the key abstraction that is provided using the Repository interface. It takes the domain class to manage as well as the id type of the domain class as Type Arguments.   
  
  
**4. What is the naming convention for finder methods in the Spring data repository interface?**  
Answer:  This is another key feature of Spring Data JPA API which makes writing query method really easy. The finder method should use a special keyword, i.e. "find", followed by the name of the variable. For example, findByLastName().  
  
  
**5. Why is an interface not a class?**  
Answer: Interface is not a class because it does not contain concrete methods. It can contain only abstract methods.  
  
  
**6. Can we perform actual tasks like access, persist, and manage data with JPA?**  
Answer: No, we can't because JPA is only a Java specification.

**7. How can we create a custom repository in Spring data JPA?**  
Answer: To create a custom repository, we have to extend it to any of the following interfaces:  
a) Repository  
b) PagingAndSortingRepository  
c) CrudRepository  
d) JpaRepository  
e) QueryByExampleRepository  
  
  
**8. What is PagingAndSortingRepository? (**[**answer**](https://www.java67.com/2023/08/difference-between-jparepository.html)**)**  
Answer: The PagingAndSortingRepository provides methods that are used to retrieve entities using pagination and sorting. It extends the CrudRepository interface.  
  
  
**9. What is @Query used for? (**[**example**](https://javarevisited.blogspot.com/2021/09/spring-data-jpa-query-example-tutorial.html)**)**  
Answer:  Spring Data API provides many ways to define SQL query which can be executed and Query annotations one of them. The @Query is an annotation that is used to execute both JPQL and native SQL queries.  
  
**10. Give an example of using @Query annotation with JPQL.**  
Answer: Here is an example of @Query annotation from Spring Data Application which returns all active orders from the database:

@Query("**SELECT** **order** **FROM** Orders o **WHERE** o.Disabled= 0")

Collection<User> findAllActiveOrders();

and, here is another example, which returns matching employees from the database

@Query("select e from Employee e where se.name = ?1")

List<Employee> getEmployees(String name);

You can further see [**Spring Framework: Spring Data JPA** course](https://javarevisited.blogspot.com/2020/05/top-5-cloud-courses-for-java-and-spring-boot-developers.html#axzz6UV11QHE1) on Pluralsight to learn more about @Query annotation of Spring Data JPA.

[](https://medium.com/javarevisited/12-advanced-spring-framework-courses-for-java-programmers-a273f6e4448c)

**11. Can you name the different types of entity mapping.**  
Answer: one-to-one mapping, one-to-many mapping, many-to-one mapping, and many-to-many mapping.  
  
**12. Define entity and name the different properties of an entity.**  
Answer: An entity is a group of states bundled (or associated) together in a single unit. It behaves like an object. It also becomes a major constituent of the object-oriented paradigm.  
  
  
**13. What is PlatformTransactionMangaer?**  
Answer: PlatformTransactionMangaer is an interface that extends TransactionManager. It is the central interface in Spring's transaction infrastructure.  
  
  
**14. How can we enable Spring Data JPA features?**  
Answer: To enable Spring data JPA features, first we have to define a configuration class and then, we can use @EnableJpaRepositoties annotation with it. This annotation will enable the features.

**15. Differentiate between findById() and getOne().**  
Answer: The findById() is available in CrudRepository while getOne() is available in JpaRepository. The findById() returns null if record does not exist while the getOne() will throw an exception called EntityNotFoundException.

1. Tell me the difference between Method Overloading and Method Overriding in Java.

This is one of the core java interview questions for experienced professionals. Method Overloading is used to increase the readability of the program.

Method Overloading has the same method name but different parameters. Method Overriding is used for providing the specific implementation of the method. Method Overriding has the same method and same parameters.

2. What do you mean by the class loader in Java?

In the Java Virtual Machine, the class loader is a type of subsystem. It is used for loading class files. When you run the program, it will be first loaded by the class loader. The built-in class loaders are of three types in Java-

The Bootstrap Classloader is the first-class loader in JVM. It is the superclass of the Extension class loader. It loads the rt.jar file that constitutes all the class files of Java Standard Edition. The files being java.lang package classes, java.net package classes, java.util packages classes, java.io packages classes, java.sql packages classes, etc.

The Extension Classloader is the class loader of Bootstrap and the parent class loader of the system classloader. It generally loads the jar files located inside the $JAVA\_HOME\_/jre/lib/ext directory.

The System Application Classloader is the type child class loader of the Extension class loader. From the cross path, the System Application Classloader loads the class files. It is by default; the classpath is set to the current directory. Therefore, it is also called the Application class loader.

3. What do you mean by inheritance in Java?

To this Java interview question, you can answer it by saying; In Java, inheritance is a mechanism through which an object acquires all the properties and behavior of another class. It is usually used for Method Overriding and Code Reusability.

The concept of inheritance in Java is based on the fact that it creates new classes that are built upon the existing classes. Therefore, these methods and fields of the parent class can be reused if we inherit from an existing class. And also, we can add new methods and fields to our current class.

In Java, the inheritance is of five types-

Hybrid Inheritance

Hierarchical Inheritance

Single-level Inheritance

Multi-level Inheritance

Multiple Inheritance

4. Why can we not override the static method in Java?

This is one of the core Java programming interview questions for experienced professionals in the Java interview.

The reasons why we cannot override the static method in Java are :

(a) The static method does not belong to the object level. Instead, it belongs to the class level. The object decides which method can be called in the method overriding.

(b) In the static method, which is the class level method, the type reference decides on which method is to be called without referring to the object. It concludes that the method that is called is determined at the compile time.

If any child class defines the static method with the same signature as the parent class, then the method in the child class hides the method in the parent class.

5. Can you tell us what the Dynamic Method Dispatch is in Java?

The Dynamic method dispatch is a process through which a call towards an overridden method is solved at a run time. It is the object that is being referred to, not the type reference variable. It decides which version of the overridden method needs to be executed.

6. What is a Java ClassPath?

A Java ClassPath is a type of environment variable which the Java Virtual Machine (JVM) uses to collect all classes by the program.

7. What can be stated as a volatile keyword in Java?

If a variable will be marked as volatile, then the variable can be read from the main memory instead of the cache memory.

8. Where does the final block not execute in Java?

To this Java interview question, you can answer it by saying; There is only one case where the final block does not execute in Java. The final block does not execute when you run System.exit(0) in the try or catch block in the Java programs.

9. What are the major points of distinction between StringBuffer and StringBuilder in Java?

A StringBuffer is thread-safe. Therefore, simultaneously two threads cannot call the methods of StringBuffer. On the other hand, in comparison to StringBuffer, a StringBuilder is not known to be thread-safe. Hence means that two threads can call the methods of StringBuilder at the same time.

On the basis of performance StringBuffer’s performance is less efficient as it is thread-safe. Whereas StringBuilder’s performance is more efficient as it is not thread-safe.

10. Can you tell the difference between the Vector and ArrayList in Java?

You can answer this Java interview question by listing the differences between the vector and the ArrayList.

(a) The common difference between the vector and the ArrayList in Java is that the vector is synchronized and thread-safe while the ArrayList is not.

(b) Since vector is synchronized, it is slow, but since the ArrayList is not synchronized, it is comparatively faster

11. Why is String immutable in Java?

To this Java interview question, you can answer it by saying there are several reasons why String is immutable or unchangeable in Java.

String pool- if you assign a value to String using the double quotes (” “), it gets stored in the string literal pool area, and a single String can be referenced by several reference variables, which will make it affect all reference variables if the String becomes mutable.

Classloading- String is used for the mechanism of class loading. If String becomes mutable, then it will become a security threat because anyone can hack it.

Cache hash value-when you use String as a key in HashMap or any other collection, you can cache its hash value. You do not need to calculate each time as it will always be constant because the string is immutable.

12. Can you tell the difference between the HashMap and HashSet in Java?

This is one of the advanced Java Interview questions for 2 to 3 years of experience candidates. The interviewer will expect you to know the answer to this Java interview question.

To answer this Java interview question, HashMap implements the Map interface, which maps keys to value. It is not synchronized, and it is not thread safe. Null keys and values are allowed, whereas duplicate keys are not allowed.

HashMap<Integer,String> studentHashMap=new HashMap<Integer,String>();

studentHashMap.put(1, “Anushka”);

studentHashMap.put(2. “Akansha”);

In the case of HashSet, it implements a Set interface that does not allow duplicate values. As a result, it is neither synchronized nor thread-safe.

HashSet studentSet=new HashSet();

studentSet.add(“Anushka”);

studentSet.add(“Akansha”);

studentSet.add(“Anjana”);

13. What is the Producer-Consumer Problem? What are the advantages of the Producer-Consumer Pattern?

This is one of the advanced Java interview questions that is asked in a Java interview. The Producer-Consumer Pattern can be implemented so that the Producer should wait if the bucket is full. The Customer needs to wait if the bucket is empty. The classical way of solving the Producer-Consumer Problem can be implemented by using the wait and notify method. It is to communicate between Consumer and Producer threads and also blocks each of them on individual conditions such as empty queue and full queue.

The Producer-Consumer Pattern is usually while writing the concurrent code and the multithreaded code. The advantages of the Producer-Consumer Pattern are-

The Producer does not need to know about the consumer or the number of consumers there are. The same case goes with the Producer as well.

The Producer and the Consumer can work at separate, different speeds. Usually, if you monitor the speed of the consumer, then more consumers can be introduced for better utilization.

If you functionally separate the Producer and Consumer, it can result in clean, readable, and manageable code.

14. How do you think we can create an immutable class in Java?

An immutable class has only one state, and it is carefully instantiated by the constructor.

To create an immutable class in Java, you can follow these steps-

(a) You can make your class final so that no class will be able to extend that. Therefore, no one will be able to override methods in this class.

(b) You can make your instance variables private, then any class will not be able to access instance variables. You can make the final so that you cannot change it.

(c) You should not create a setter method for instance variables so that no explicit way will be there to change the state of the instance variables.

(d) Initialize the variables in the constructor and take proper care while working with the mutable object. You must do the deep copy in the case of mutable objects.

(e) You should return a clone of the object from the getter method so that it won’t return the original object. After that, your original object will be intact.

15. Can you tell us the difference between the fail-safe and fail-fast iterator in java?

This is an advanced Java question for experienced professionals. To answer this question, the difference between the Fail-fast iterator and fail-safe iterator in Java are-

fail-fast iterator- It fails as soon as they realize that the structure of the collection is changed since the beginning of the iteration. The structural changes here are removing, adding, or updating any element from the collection when one thread is iterating over that other collection. The fail-fast can be implemented by maintaining a modification count. When the iteration thread will realize the change in the count, it throws ConcurrentModificationException.

fail-safe iterator in Java- It does not throw an Exception when the collection is modified in a structural manner while one iteration is working over it. It works on the clone collection instead of the original collection.