


Knowledge Base ▾Resources ▾Deals ▾Join Us ▾About ▾Login ▾Register



Java Code Geeks
JAVA 2 JAVA DEVELOPERS RESOURCE CENTER

ANDROID ▾

JAVA ▾

JVM LANGUAGES ▾

SOFTWARE DEVELOPMENT

AGILE

CAREER


COMMUNICATIONS

DEVOPS

META JCG ▾

Home » Java » Core Java » 150 Java Interview Questions and Answers – The ULTIMATE List (PDF Download)

ABOUT SOTIRIOS-EFSTATHIOS MANEAS



Sotirios-Efstathios (Stathis) Maneas is a postgraduate student at the Department of Informatics and Telecommunications of The National and Kapodistrian University of Athens. His main interests include distributed systems, web crawling, model checking, operating systems, programming languages and web applications.

150 Java Interview Questions and Answers – The ULTIMATE List (PDF Download)

Posted by: Sotirios-Efstathios Maneas

In Core Java

April 7th, 2014

152 Comments

32531 Views

Our Java Interview Questions and Answers collection is all about different types of questions that can be used in a Java interview, in order for the employer to test your skills in Java and object-oriented programming in general.

In the following sections we will discuss Java Interview Questions about object-oriented programming and its characteristics, general questions regarding Java and its functionality, collections in Java, garbage collectors, exception handling, Java applets, Swing, JDBC, Remote Method Invocation (RMI), Servlets and JSP.

Let's go...!




Table Of Contents

A.Object Oriented Programming (OOP)

B.General Questions about Java

C.Java Threads

D.Java Collections

E.Garbage Collectors

F.Exception Handling

G.Java Applets

H.Swing

I.JDBC

J.Remote Method Invocation (RMI)

K.Servlets

L.JSP

A. Object Oriented Programming (OOP)

1. What is Java?

Java is a computer programming language that is concurrent, class-based and object-oriented. The advantages of object-oriented software development are shown below:

- Modular development of code, which leads to easy maintenance and modification.
- Reusability of code.
- Improved reliability and flexibility of code.
- Increased understanding of code.

2. What are the concepts of OOP?

Object Oriented Programming (OOP) includes:

- Abstraction
- Encapsulation
- Polymorphism
- Inheritance

1 of 31

8/31/19, 12:56 AM

- Predefined types must be objects
- User defined types must be objects
- Operations must be performed by sending messages to objects

3. Mention some features of Java

Some of the features which play important role in the popularity of java are as follows:

- Object-Oriented
- Platform independent
- High Performance
- Multithreaded
- Portable
- Secure

Sample code for Helloworld in java is shown below:

Hello World

```
1 | public class Helloworld{
2 |
3 |     public static void main(String args[])
4 |     {
5 |         System.out.println("Hello World");
6 |     }
7 |
8 | }
```

4. Is Java 100% Object-oriented?

Not 100%. Java does not satisfy all the OOP conditions (predefined types must be objects) because it uses eight primitive data types(Boolean, byte, char, int, float, double, long, short) which are not objects.

5. What is Abstraction?

Abstraction is the process of separating ideas from specific instances and thus, develop classes in terms of their own functionality, instead of their implementation details. Java supports the creation and existence of abstract classes that expose interfaces, without including the actual implementation of all methods. The abstraction technique aims to separate the implementation details of a class from its behavior.

Abstract class Person is presented below. It has an abstract method getName.

Abstract Class Person

```
1 | public abstract class Person
2 | {
3 |     public abstract String getName();
4 | }
```

Employee class extends the Abstract class Person. The method getName returns the name attribute of the employee.

Employee Class

```
01 | public class Employee extends Person
02 | {
03 |     private String name;
04 |     public Employee(String name)
05 |     {
06 |         this.name = name;
07 |     }
08 |     public String getName()
09 |     {
10 |         return this.name;
11 |     }
12 |     public static void main (String args[])
13 |     {
14 |         Employee employee = new Employee("John Wilson");
15 |         System.out.println("Employee's Name "+ employee.getName());
16 |         Person person = new Employee("Thomas Smith");
17 |         System.out.println("Employee-Person's Name "+ person.getName());
18 |     }
19 | }
20 |
21 |
22 |
23 |
24 |
25 | }
```

6. What is Encapsulation?

Encapsulation provides objects with the ability to hide their internal characteristics and behavior. Each object provides a number of methods, which can be accessed by other objects and change its internal data. In Java, there are three access modifiers: public, private and protected. Each modifier imposes different access rights to other classes, either in the same or in external packages. Some of the advantages of using encapsulation are listed below:

- The internal state of every object is protected by hiding its attributes.
- It increases usability and maintenance of code, because the behavior of an object can be independently changed or extended.
- It improves modularity by preventing objects to interact with each other, in an undesired way.

You can refer to our tutorial [here](#) for more details and examples on encapsulation.

A sample class Student which has attributes Id and Name is shown as an example for encapsulation.



Student Class

```

01 | public class Student{
02 |     private int id;
03 |     private String name;
04 |
05 |     public void setId(int id)
06 |     {
07 |         this.id = id;
08 |     }
09 |
10 |     public void setName(String name)
11 |     {
12 |         this.name = name;
13 |     }
14 |
15 |     public int getId()
16 |     {
17 |         return this.id;
18 |     }
19 |
20 |     public String getName()
21 |     {
22 |         return this.name;
23 |     }
24 |
25 |     public static void main(String args[])
26 |     {
27 |         Student student=new Student();
28 |         student.setId(1034);
29 |         student.setName("David Smith");
30 |
31 |         System.out.println("Student id "+ student.getId());
32 |         System.out.println("Student name "+ student.getName());
33 |
34 |     }
35 |
36 | }

```

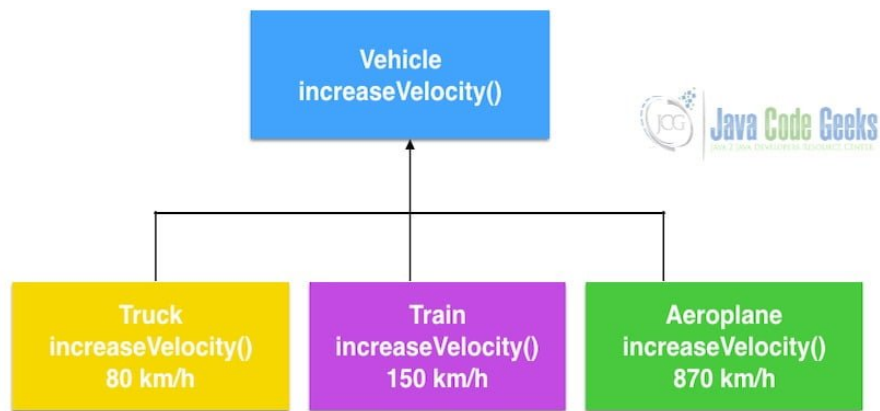
7. What are the differences between Abstraction and Encapsulation?

Abstraction and encapsulation are complementary concepts. On the one hand, abstraction focuses on the behavior of an object. On the other hand, encapsulation focuses on the implementation of an object's behavior. Encapsulation is usually achieved by hiding information about the internal state of an object and thus, can be seen as a strategy used in order to provide abstraction.

8. What is Polymorphism?

Polymorphism is the ability of programming languages to present the same interface for differing underlying data types. A polymorphic type is a type whose operations can also be applied to values of some other type.

You can see the example below where Vehicle interface has the method increaseVelocity. Truck, Train and Aeroplane implement the Vehicle Interface and the method increases the velocity to the appropriate velocity related to the vehicle type.



Polymorphism

9. What are the types of Polymorphism?

There are two types of Polymorphism in Java:

- Compile-time polymorphism (Static binding) – Method overloading
- Runtime polymorphism (Dynamic binding) – Method overriding

We can perform polymorphism by Method Overloading and Method Overriding.

Compile Time	Runtime
--------------	---------

Methods of a class have the same name. Each method has a different number of parameters . It can have parameters with different types and order.	the subclass has method with the name as of a superclass method. It has the number of paramers, type of parameters and the return type as of a superclass method.
Method Overloading is to add to the method behavior. It can be extending to the method's behavior.	Method Overriding is to modify the method's behavior .
Overloaded methods will not have same signature.	Overridden methods will have exactly the same signature.
Inheritance is not need in this case.	Inheritance is required.

Sample code for overloading method subtract of Calculator Class is shown below:

Calculator Class

```

01 | public class Calculator {
02 |
03 |     public int subtract(int a, int b)
04 |     {
05 |         return a-b;
06 |     }
07 |     public double subtract( double a, double b)
08 |     {
09 |         return a-b;
10 |     }
11 |
12 |     public static void main(String args[])
13 |     {
14 |         Calculator calculator = new Calculator();
15 |         System.out.println("Difference of 150 and 12 is " +calculator.subtract(150,12));
16 |         System.out.println("Difference of 15.5 and 15.4 is " +calculator.subtract(15.50,15.40));
17 |     }

```

Method overriding is shown below in Shape class. Shape has a method getArea.

Shape Class

```

1 | public class Shape
2 | {
3 |     public void getArea(){System.out.println("Shape Area");}
4 | }

```

Rectangle class overrides getArea method and the implementation of the method is specific to Rectangle. Override annotation is used to indicate to the compiler that the method is overridden. Readability of the code is improved using the annotation.

Rectangle Class

```

01 | public class Rectangle extends Shape{
02 |
03 |     @Override
04 |     public void getArea()
05 |     {
06 |         System.out.println("Rectangle Area");
07 |     }
08 | }
09 |
10 |
11 |
12 |     public static void main(String args[])
13 |     {
14 |         Shape shape = new Shape();
15 |         shape.getArea();
16 |
17 |         Rectangle rectangle = new Rectangle();
18 |         rectangle.getArea();
19 |     }
20 | }
21 |
22 | }

```

10. What is Inheritance?

Inheritance provides an object with the ability to acquire the fields and methods of another class, called base class. Inheritance provides reusability of code and can be used to add additional features to an existing class, without modifying it.

Sample class Mammal is shown below which has a constructor.

Mammal Class

```

1 | public class Mammal{
2 |
3 |     public Mammal()
4 |     {
5 |         System.out.println("Mammal created");
6 |     }
7 | }
8 | }

```

Man class extends Mammal which has a default constructor. The sample code is shown below.

Man class

```

1 | public class Man extends Mammal{
2 |
3 |     public Man()
4 |     {
5 |         System.out.println("Man is created");
6 |     }
7 | }

```

Inheritance is tested by creating an instance of Man using default constructor. The sample code is shown to demonstrate the inheritance.

TestInheritance Class

```

1 | public class TestInheritance{
2 |
3 |     public static void main(String args[])

```

```
4 | {  
5 |     Man man = new Man();  
6 | }  
7 | }
```

11. What is composition?

Composition is exactly like Aggregation except that the lifetime of the 'part' is controlled by the 'whole'. This control may be direct or transitive. That is, the 'whole' may take direct responsibility for creating or destroying the 'part', or it may accept an already created part, and later pass it on to some other whole that assumes responsibility for it.

Sample class Car is shown below to demonstrate Composition of tires, doors, windows and steering.

Car class

```
01 | public class Car  
02 | {  
03 |     private Tire[] tires;  
04 |  
05 |     private Door[] doors;  
06 |  
07 |     private Steering steering;  
08 |  
09 |     private Window[] windows;  
10 | }  
11 |  
12 | class Tire  
13 | {  
14 | }  
15 |  
16 | class Door  
17 | {  
18 | }  
19 |  
20 |  
21 | class Steering  
22 | {  
23 | }  
24 |  
25 |  
26 | class Window  
27 | {  
28 | }  
29 |  
30 | }
```

12. What is an association?

Association represents the ability of one instance to send a message to another instance. This is typically implemented with a pointer or reference instance variable, although it might also be implemented as a method argument or the creation of a local variable.

13. What is aggregation?

Aggregation is the typical whole/part relationship. This is exactly the same as an association with the exception that instances cannot have cyclic aggregation relationships.

Sample class Person is shown below to demonstrate Aggregation relationship with Address.

Person class

```
01 | public class Person  
02 | {  
03 |     private Address address;  
04 | }  
05 |  
06 | class Address  
07 | {  
08 |     private String city;  
09 |     private String state;  
10 |     private String country;  
11 |     private String line1;  
12 |     private String line2;  
13 | }  
14 |  
15 |  
16 |  
17 |  
18 |  
19 | }
```

B.General Questions about Java

14. What is JVM?

A Java virtual machine (JVM) is a process virtual machine that can execute Java bytecode. Each Java source file is compiled into a bytecode file, which is executed by the JVM.

15. Why is Java called the Platform Independent Programming Language?

Java was designed to allow application programs to be built that could be run on any platform, without having to be rewritten or recompiled by the programmer for each separate platform. A Java virtual machine makes this possible because it is aware of the specific instruction lengths and other particularities of the underlying hardware platform.

16. What is the Difference between JDK and JRE?



The Java Runtime Environment (JRE) is basically the Java Virtual Machine (JVM) where your Java programs are being executed. It also includes browser plugins for applet execution. The Java Development Kit (JDK) is the full-featured Software Development Kit for Java, including the JRE, the compilers and tools (like JavaDoc, and Java Debugger), in order for a user to develop, compile and execute Java applications.

JDK	JRE
JDK stands for the term : Java Development Kit.	JRE stands for the term: Java Runtime Environment.
JDK is the tool for compiling, documenting and packaging Java software.	JRE is a runtime environment. JavaByte code gets executed in the environment.
JDK has JRE and development tools.	JRE is a JVM implementation

17. What does the static keyword mean?

The static keyword denotes that a member variable or method can be accessed, without requiring an instantiation of the class to which it belongs.

Sample static method which is shown below:

Static method

```
1 | static void printGreeting()  
2 |     {  
3 |  
4 |  
5 |     }
```

18. Can you override private or static method in Java?

A user cannot override static methods in Java, because method overriding is based upon dynamic binding at runtime and static methods are statically bound at compile time. A static method is not associated with any instance of a class so the concept is not applicable.

19. Can you access the non-static variable in static context?

A static variable in Java belongs to its class and its value remains the same for all its instances. A static variable is initialized when the class is loaded by the JVM. If your code tries to access a non-static variable, without any instance, the compiler will complain, because those variables are not created yet and they are not associated with any instance.

20. What are the Data Types supported by Java?

The eight primitive data types supported by the Java programming language are:

- byte
- short
- int
- long
- float
- double
- boolean
- char

21. What is Autoboxing and Unboxing?

Autoboxing is the automatic conversion made by the Java compiler between the primitive types and their corresponding object wrapper classes. For example, the compiler converts an int to an Integer, a double to a Double, and so on. If the conversion goes the other way, this operation is called unboxing.

22. What is Function Overriding and Overloading in Java?

Method overloading in Java occurs when two or more methods in the same class have the exact same name, but different parameters. On the other hand, method overriding is defined as the case when a child class redefines the same method as a parent class. Overridden methods must have the same name, argument list, and return type. The overriding method may not limit the access of the method it overrides.

23. What is a Constructor?

A constructor gets invoked when a new object is created. Every class has a constructor. In case the programmer does not provide a constructor for a class, the Java compiler (Javac) creates a default constructor for that class.

A default constructor in java is shown in the example below:

Default Constructor

```
1 | public Man()  
2 | {  
3 |     System.out.println("Man is created");  
4 | }
```

Constructor which takes a parameter is shown in the sample below:

Constructor

```
1 | private String name;  
2 |  
3 | public Employee(String name)
```



```
4 | {  
5 |     this.name = name;  
6 | }
```

24. What is Constructor Overloading?

The constructor overloading is similar to method overloading in Java. Different constructors can be created for a single class. Each constructor must have its own unique parameter list.

25. What is Copy-Constructor?

Finally, Java does support copy constructors like C++, but the difference lies in the fact that Java does not create a default copy constructor if you do not write your own.

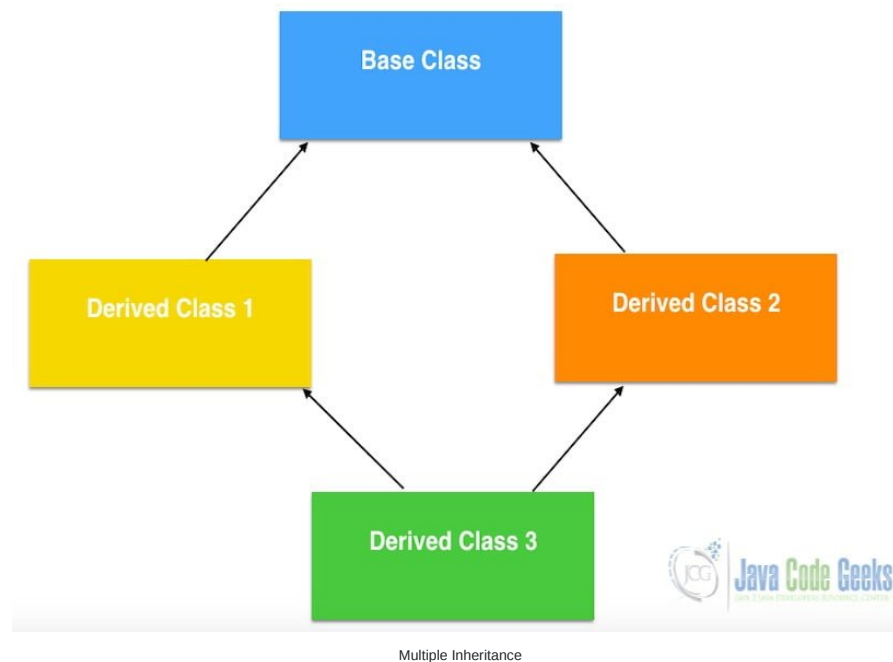
Copy constructor for Employee class is shown below:

Copy Constructor

```
01 | public class Employee extends Person  
02 | {  
03 |     private String name;  
04 |  
05 |     public Employee(String name)  
06 |     {  
07 |         this.name = name;  
08 |     }  
09 |  
10 |     public Employee(Employee emp)  
11 |     {  
12 |         this.name = emp.name;  
13 |     }  
14 | }  
15 | }
```

26. Does Java support multiple inheritance?

No, Java does not support multiple inheritance. Each class is able to extend only on one class but is able to implement more than one interfaces.



27. What is the difference between an Interface and an Abstract class?

Java provides and supports the creation of both the abstract classes and interfaces. Both implementations share some common characteristics, but they differ in the following features:

- All methods in an interface are implicitly abstract. On the other hand, an abstract class may contain both abstract and non-abstract methods.
- A class may implement a number of Interfaces but can extend only one abstract class.
- In order for a class to implement an interface, it must implement all its declared methods. However, a class may not implement all declared methods of an abstract class. Though, in this case, the sub-class must also be declared as abstract.
- Abstract classes can implement interfaces without even providing the implementation of interface methods.
- Variables declared in a Java interface is by default final. An abstract class may contain non-final variables.
- Members of a Java interface are public by default. A member of an abstract class can either be private, protected or public.

- An interface is absolutely abstract and cannot be instantiated. An abstract class also cannot be instantiated but can be invoked if it contains the main method.

Also, check out the Abstract class and Interface differences for JDK 8.

Interface	Abstract Class
An interface has the method signatures. It does not have any implementation.	Abstract class has the abstract methods and details to be overridden.
A Class can implement multiple interfaces	In this case, a class can extend just one abstract class
Interface has all abstract methods.	Non abstract methods can be there in an abstract class.
Instance properties cannot be there in an interface.	Instance properties can be there in an abstract class.
An Interface is publicly visible or not visible.	An abstract class can be public, private and protected visible.
Any change in the interface will impact the classes implementing the interface.	Adding a method to an abstract class and implementing it does not require change in the code for derived classes.
An Interface cannot have constructors	An abstract class can have constructors
Interfaces are slow in terms of performance	Abstract classes are fast in execution of the methods in the derived classes.

28. What are pass by reference and pass by value?

When an object is passed by value, this means that a copy of the object is passed. Thus, even if changes are made to that object, it does not affect the original value. When an object is passed by reference, this means that the actual object is not passed, rather a reference of the object is passed. Thus, any changes made by the external method, are also reflected in all places.

Sample code is presented below which shows pass by value.

Pass by Value

```

01 | public class ComputingEngine
02 | {
03 |     public static void main(String[] args)
04 |     {
05 |         int x = 15;
06 |         ComputingEngine engine = new ComputingEngine();
07 |         engine.modify(x);
08 |         System.out.println("The value of x after passing by value "+x);
09 |     }
10 |     public void modify(int x)
11 |     {
12 |         x = 12;
13 |     }
14 | }

```

Below example shows pass by reference in the code.

Pass by Reference

```

01 | public class ComputingEngine
02 | {
03 |     public static void main(String[] args)
04 |     {
05 |
06 |         ComputingEngine engine = new ComputingEngine();
07 |
08 |         Computation computation = new Computation(65);
09 |         engine.changeComputedValue(computation);
10 |
11 |         System.out.println("The value of x after passing by reference "+ computation.x);
12 |
13 |     }
14 |
15 |
16 |
17 |     public void changeComputedValue(Computation computation)
18 |     {
19 |         computation = new Computation();
20 |         computation.x = 40;
21 |     }
22 | }
23 |
24 |
25 | class Computation
26 | {
27 |     int x;
28 |     Computation(int i) { x = i; }
29 |     Computation() { x = 1; }
30 | }
31 |

```

29. What is the purpose of a Volatile Variable?

Volatile variable values can be modified by different threads. They will never have the chance to block and hold a lock. Synchronization will happen whenever the variables are accessed. Using volatile may be faster than a lock, but it will not work in some situations. The range of situations in which volatile is effective was expanded in Java 5; in particular, double-checked locking now works correctly.

Sample code for volatile variable is shown below:

Volatile Variable

```

1 | public class DistributedObject {
2 |
3 |     public volatile int count = 0;
4 |
5 | }

```



30. What is the purpose of a Transient variable?

A transient variable would not be serialized even if the class to which it belongs is serialized.

Sample class which has transient variable is shown below:

Transient variable

```
1 public class Paper implements Serializable
2 {
3     private int id;
4     private String title;
5     private String author;
6     private transient int version = 1;
7 }
8 }
```

31. What is Local Variable and Instance Variable?

Local variable	Instance variable
Local variable is declared inside a method or constructor. It can be declared within a block	Instance variable is declared inside a class.
Local variable need to be initialized before use. The code will not compile.	Instance variable initialization is not necessary. If not initialized, default value is used.

32. What are the different access modifiers available in Java?

There are four types of Access modifiers:

- Public – accessible from everywhere in the application
- Protected – accessible within the package and the subclasses in any package
- Package Private (Default) – accessible strictly within the package
- Private – accessible only within the same class where it is declared

33. Difference between static binding and dynamic binding

Static Binding	Dynamic Binding
Definition of a procedure is related to static binding	An example for dynamic binding is activation of a procedure
Declaration of a name for a variable is done to bind statically the variable.	Binding of a name can be dynamic bound.
The Scope of the declaration is statically bound.	Lifetime of a binding is dynamically bound.

Sample code for static binding is shown below:

Static Binding

```
01 public class Shape
02 {
03     public void getArea()
04     {
05         System.out.println("Shape Area");
06     }
07 }
08
09 public static void main(String args[])
10 {
11     Shape shape = new Shape();
12     shape.getArea();
13 }
14 }
15 }
```

Sample code for Dynamic binding is shown below:

Dynamic Binding

```
01 public class Rectangle extends Shape{
02
03     public void getArea()
04     {
05         System.out.println("Rectangle Area");
06     }
07 }
08
09
10
11
12 public static void main(String args[])
13 {
14     Shape shape = new Rectangle();
15     shape.getArea();
16 }
17 }
18 }
19 }
```

34. What are wrapper classes?

A wrapper class converts java primitives into objects. So a primitive wrapper class is a wrapper class that encapsulates, hides or wraps data types from the eight primitive data types so that these can be used to create instantiated objects with methods in another class or in other classes. The primitive wrapper classes are found in the Java API.



35. What is singleton class and how can we make a class singleton?

In a singleton class we:

- ensure that only one instance of the singleton class ever exists
- provide global access to that instance

To create a singleton class we:

- declare all constructors of the class as private
- provide a static method that returns a reference to the instance

Sample code below shows Double checked Singleton class implementation.

Singleton Class

```
01 public class DoubleCheckedSingleton {
02     private static volatile DoubleCheckedSingleton instance;
03     public static DoubleCheckedSingleton getInstance() {
04         if (instance == null) {
05             synchronized (DoubleCheckedSingleton .class) {
06                 if (instance == null) {
07                     instance = new DoubleCheckedSingleton();
08                 }
09             }
10         }
11         return instance;
12     }
13 }
14 }
```

C.Java Threads

36. What is the difference between processes and threads?

A process is an execution of a program, while a Thread is a single execution sequence within a process. A process can contain multiple threads. A Thread is sometimes called a lightweight process.

Processes	Threads
Process is related to execution of a program .	Process consists of multiple threads.
Processes communicate with each other using inter-process communication .	Threads of a process can communicate with each other.
Processes have control over the child processes.	Threads of a process can have control over other threads.
Any modification in the parent process does not alter child processes	Any modification in the main thread may impact the behavior of the other threads of the process.
Processes get executed in separate memory spaces.	Threads are executed in shared memory spaces.
Operating system controls the process .	Developer of the software has control over the usage of the threads.
Processes are independent of each other.	Threads are dependent on each other.

37. Explain different ways of creating a thread. Which one would you prefer and why?

There are three ways that can be used in order for a Thread to be created:

- A class may extend the Thread class.
- A class may implement the Runnable interface.
- An application can use the Executor framework, in order to create a thread pool.

The Runnable interface is preferred, as it does not require an object to inherit the Thread class. In case your application design requires multiple inheritance, only interfaces can help you. Also, the thread pool is very efficient and can be implemented and used very easily.

38. Explain the available thread states in a high-level.

During its execution, a thread can reside in one of the following states:

- NEW: The thread becomes ready to run, but does not necessarily start running immediately.
- RUNNABLE: The Java Virtual Machine (JVM) is actively executing the thread's code.
- BLOCKED: The thread is in a blocked state while waiting for a monitor lock.
- WAITING: The thread waits for another thread to perform a particular action.
- TIMED_WAITING: The thread waits for another thread to perform a particular action up to a specified waiting time.
- TERMINATED: The thread has finished its execution.

39. What is the difference between a method and block that are synchronized?

In Java programming, each object has a lock. A thread can acquire the lock for an object by using the synchronized keyword. The synchronized keyword can be applied in a method level (coarse-grained lock) or block level of code (fine-grained lock).



40. How does thread synchronization occurs inside a monitor?

The JVM uses locks in conjunction with monitors. A monitor is basically a guardian that watches over a sequence of synchronized code and ensuring that only one thread at a time executes a synchronized piece of code. Each monitor is associated with an object reference. The thread is not allowed to execute the code until it obtains the lock.

41. What is a deadlock?

A condition that occurs when two processes are waiting for each other to complete, before proceeding. The result is that both processes wait endlessly.

42. How do you ensure that N threads can access N resources without deadlock?

A very simple way to avoid deadlock while using N threads is to impose an ordering on the locks and force each thread to follow that ordering. Thus, if all threads lock and unlock the mutexes in the same order, no deadlocks can arise.

43. What are the differences between wait and sleep method in Java?

	Wait	Sleep
Call on	current thread synchronizes on the lock object when there is a call on the object.	Call on a Thread happens on the currently executing thread.
Synchronized	Synchronized is used to access the same Object from multiple threads..	Synchronized is used to sleep over the Sleeping thread from multiple threads.
Hold lock	release the lock for other objects to have the chance to execute	keep lock for at least t times if timeout specified or somebody interrupt.
Wake up condition	until call notify(), notifyAll() from object	until at least time expire or call interrupt().
Usage	for time-synchronization	for multi-thread-synchronization

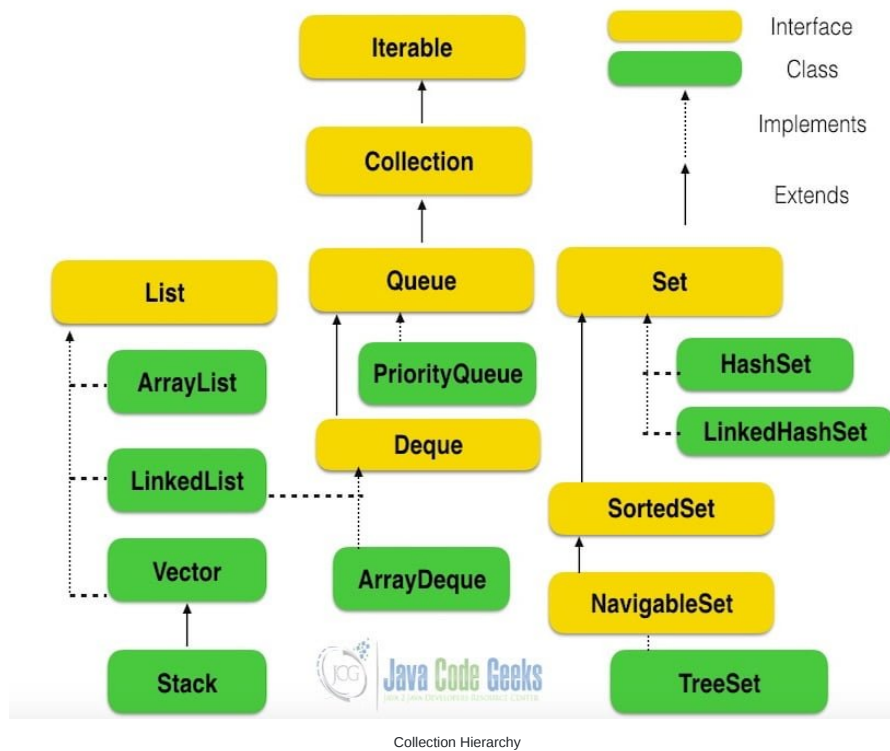
D.Java Collections

44. What are the basic interfaces of Java Collections Framework?

Java Collections Framework provides a well-designed set of interfaces and classes that support operations on a collections of objects. The most basic interfaces that reside in the Java Collections Framework are:

- Collection, which represents a group of objects known as its elements.
- Set, which is a collection that cannot contain duplicate elements.
- List, which is an ordered collection and can contain duplicate elements.
- Map, which is an object that maps keys to values and cannot contain duplicate keys.





45. Why Collection does not extend Cloneable and Serializable interfaces?

The Collection interface specifies groups of objects known as elements. Each concrete implementation of a Collection can choose its own way of how to maintain and order its elements. Some collections allow duplicate keys, while some other collections do not. The semantics and the implications of either cloning or serialization come into play when dealing with actual implementations. Thus, the concrete implementations of collections should decide how they can be cloned or serialized.

46. What is an Iterator?

The Iterator interface provides a number of methods that are able to iterate over any Collection. Each Java Collection contains the iterator method that returns an Iterator instance. Iterators are capable of removing elements from the underlying collection during the iteration.

47. What differences exist between Iterator and ListIterator?

The differences of these elements are listed below:

- An Iterator can be used to traverse the Set and List collections, while the ListIterator can be used to iterate only over Lists.
- The Iterator can traverse a collection only in the forward direction, while the ListIterator can traverse a List in both directions.
- The ListIterator implements the Iterator interface and contains extra functionality, such as adding an element, replacing an element, getting the index position for previous and next elements, etc.

48. What is the difference between fail-fast and fail-safe?

The Iterator's fail-safe property works with the clone of the underlying collection and thus, it is not affected by any modification in the collection. All the collection classes in java.util package are fail-fast, while the collection classes in java.util.concurrent are fail-safe. Fail-fast iterators throw a ConcurrentModificationException, while fail-safe iterator never throws such an exception.

49. How HashMap works in Java?

A HashMap in Java stores key-value pairs. The HashMap requires a hash function and uses hashCode and equals methods, in order to put and retrieve elements to and from the collection respectively. When the put method is invoked, the HashMap calculates the hash value of the key and stores the pair in the appropriate index inside the collection. If the key exists, its value is updated with the new value. Some important characteristics of a HashMap are its capacity, its load factor and the threshold resizing.

50. What is the importance of hashCode() and equals() methods?

In Java, a HashMap uses the hashCode and equals methods to determine the index of the key-value pair and to detect duplicates. More specifically, the hashCode method is used in order to determine where the specified key will be stored. Since different keys may produce the same hash value, the equals method is used, in order to determine whether the specified key actually exists in the collection or not. Therefore, the implementation of both methods is crucial to the accuracy and efficiency of the HashMap.



51. What is the difference between HashMap and Hashtable?

Both the HashMap and Hashtable classes implement the Map interface and thus, have very similar characteristics. However, they differ in the following features:

- A HashMap allows the existence of null keys and values, while a Hashtable does not allow neither null keys, nor null values.
- A Hashtable is synchronized, while a HashMap is not. Thus, HashMap is preferred in single-threaded environments, while a Hashtable is suitable for multi-threaded environments.
- A HashMap provides its set of keys and a Java application can iterate over them. Thus, a HashMap is fail-fast. On the other hand, a Hashtable provides an Enumeration of its keys.
- The Hashtable class is considered to be a legacy class.

52. What is the difference between Array and ArrayList? When will you use Array over ArrayList?

The Array and ArrayList classes differ on the following features:

- Arrays can contain primitive or objects, while an ArrayList can contain only objects.
- Arrays have fixed size, while an ArrayList is dynamic.
- An ArrayList provides more methods and features, such as addAll, removeAll, iterator, etc.
- For a list of primitive data types, the collections use autoboxing to reduce the coding effort. However, this approach makes them slower when working on fixed size primitive data types.

Array	ArrayList
Array should not have values of different data types	Array List can have values of different data types.
Size of the array is defined at the time of declaration	Size of the ArrayList can be dynamically changed
You have to specify the index in order to add data in an array	You do not need to specify the index in an ArrayList
Arrays are not type parameterized	ArrayLists can be type parameterized.
Arrays can have primitive data types as well as objects	ArrayLists can have only objects, no primitive data types are allowed

53. What is difference between ArrayList and LinkedList?

Both the ArrayList and LinkedList classes implement the List interface, but they differ on the following features:

- An ArrayList is an index based data structure backed by an Array. It provides random access to its elements with a performance equal to $O(1)$. On the other hand, a LinkedList stores its data as list of elements and every element is linked to its previous and next element. In this case, the search operation for an element has execution time equal to $O(n)$.
- The Insertion, addition and removal operations of an element are faster in a LinkedList compared to an ArrayList, because there is no need of resizing an array or updating the index when an element is added in some arbitrary position inside the collection.
- A LinkedList consumes more memory than an ArrayList, because every node in a LinkedList stores two references, one for its previous element and one for its next element.

Check also our article [ArrayList vs. LinkedList](#).

54. What is the difference between Comparable and Comparator?

- Java provides the Comparable interface, which contains only one method, called compareTo. This method compares two objects, in order to impose an order between them. Specifically, it returns a negative integer, zero, or a positive integer to indicate that the input object is less than, equal or greater than the existing object.
- Java provides the Comparator interface, which contains two methods, called compare and equals. The first method compares its two input arguments and imposes an order between them. It returns a negative integer, zero, or a positive integer to indicate that the first argument is less than, equal to, or greater than the second. The second method requires an object as a parameter and aims to decide whether the input object is equal to the comparator. The method returns true, only if the specified object is also a comparator and it imposes the same ordering as the comparator.

55. What is Java Priority Queue?

The PriorityQueue is an unbounded queue, based on a priority heap and its elements are ordered in their natural order. At the time of its creation, we can provide a Comparator that is responsible for ordering the elements of the PriorityQueue. A PriorityQueue does not allow null values, those objects that do not provide natural ordering or those objects that do not have any comparator associated with them. Finally, the Java PriorityQueue is not thread-safe and it requires $O(\log(n))$ time for its enqueueing and dequeueing operations.

56. What do you know about the big-O notation and can you give some examples with respect to different data structures?

The Big-O notation simply describes how well an algorithm scales or performs in the worst case scenario as the number of elements in a data structure increases. The Big-O notation can also be used to describe other behavior such as memory consumption. Since the collection classes are actually data structures, we usually use the Big-O notation to choose the best implementation to use, based on time, memory and performance. Big-O notation can give a good indication about performance for large amounts of data.

57. What is the trade-off between using an unordered array versus an ordered array?

The major advantage of an ordered array is that the search times have time complexity of $O(\log n)$, compared to that of an unordered array,



which is $O(n)$. The disadvantage of an ordered array is that the insertion operation has a time complexity of $O(n)$, because the elements with higher values must be moved to make room for the new element. Instead, the insertion operation for an unordered array takes constant time of $O(1)$.

58. What are some of the best practices related to the Java Collection framework?

- Choosing the right type of the collection to use, based on the application's needs, is very crucial for its performance. For example if the size of the elements is fixed and known a priori, we shall use an Array, instead of an ArrayList.
- Some collection classes allow us to specify their initial capacity. Thus, if we have an estimation on the number of elements that will be stored, we can use it to avoid rehashing or resizing.
- Always use Generics for type-safety, readability, and robustness. Also, by using Generics you avoid the ClassCastException during runtime.
- Use immutable classes provided by the Java Development Kit (JDK) as a key in a Map, in order to avoid the implementation of the hashCode and equals methods for our custom class.
- Program in terms of interface not implementation.
- Return zero-length collections or arrays as opposed to returning a null in case the underlying collection is actually empty.

59. What is the difference between Enumeration and Iterator interfaces?

Enumeration is twice as fast as compared to an Iterator and uses very less memory. However, the Iterator is much safer compared to Enumeration, because other threads are not able to modify the collection object that is currently traversed by the iterator. Also, Iterators allow the caller to remove elements from the underlying collection, something which is not possible with Enumerations.

60. What is the difference between HashSet and TreeSet?

The HashSet is implemented using a hash table and thus, its elements are not ordered. The add, remove, and contains methods of a HashSet have constant time complexity $O(1)$. On the other hand, a TreeSet is implemented using a tree structure. The elements in a TreeSet are sorted, and thus, the add, remove, and contains methods have time complexity of $O(\log n)$.

E. Garbage Collectors

61. What is the purpose of garbage collection in Java, and when is it used?

The purpose of garbage collection is to identify and discard those objects that are no longer needed by the application, in order for the resources to be reclaimed and reused.

62. What do System.gc() and Runtime.gc() methods do?

These methods can be used as a hint to the JVM, in order to start a garbage collection. However, this is up to the Java Virtual Machine (JVM) to start the garbage collection immediately or later in time.

Sample class ReferenceObject is shown below to demonstrate the usage of System.gc and Runtime.gc methods.

```
01 | public class ReferenceObject
02 | {
03 |     public void finalize()
04 |     {
05 |         System.out.println("object is garbage collected");
06 |     }
07 | }
08 |
09 | public static void main(String args[]){
10 |     ReferenceObject refObj1=new ReferenceObject();
11 |     ReferenceObject refObj2=new ReferenceObject();
12 |     refObj1=null;
13 |     refObj2=null;
14 |     System.gc();
15 |
16 |     Runtime.gc();
17 | }
18 | }
```

63. When is the finalize() called? What is the purpose of finalization?

The finalize method is called by the garbage collector, just before releasing the object's memory. It is normally advised to release resources held by the object inside the finalize method.

Finalize method in ReferenceObject class is shown below as an example.

Finalize Method

```
1 | public class ReferenceObject
2 | {
3 |     public void finalize()
4 |     {
5 |         System.out.println("object is garbage collected");
6 |     }
7 | }
8 | }
```



64. If an object reference is set to null, will the Garbage Collector immediately free the memory held by that object?

No, the object will be available for garbage collection in the next cycle of the garbage collector.

65. What is structure of Java Heap?

The JVM has a heap that is the runtime data area from which memory for all class instances and arrays is allocated. It is created at the JVM start-up. Heap memory for objects is reclaimed by an automatic memory management system which is known as a garbage collector. Heap memory consists of live and dead objects. Live objects are accessible by the application and will not be a subject of garbage collection. Dead objects are those which will never be accessible by the application, but have not been collected by the garbage collector yet. Such objects occupy the heap memory space until they are eventually collected by the garbage collector.

66. What is the difference between Serial and Throughput Garbage collector?

The throughput garbage collector uses a parallel version of the young generation collector and is meant to be used with applications that have medium to large data sets. On the other hand, the serial collector is usually adequate for most small applications (those requiring heaps of up to approximately 100MB on modern processors).

67. When does an Object becomes eligible for Garbage collection in Java?

A Java object is subject to garbage collection when it becomes unreachable to the program in which it is currently used.

68. Does Garbage collection occur in permanent generation space in JVM?

Garbage Collection does occur in PermGen space and if PermGen space is full or cross a threshold, it can trigger a full garbage collection. If you look carefully at the output of the garbage collector, you will find that PermGen space is also garbage collected. This is the reason why correct sizing of PermGen space is important to avoid frequent full garbage collections. Also check our article [Java 8: PermGen to Metaspace](#).

F.Exception Handling

69. What are the differences between Checked Exception and Unchecked Exception?

Checked Exception	Unchecked Exception
known as compile time exceptions	known as Runtime exceptions
propagated using throws keyword	automatically propagated
can create custom exception by extending java.lang.Exception class	can create custom exception by extending Runtime exception

70. What is the difference between Exception and Error in java?

Exception and Error classes are both subclasses of the Throwable class. The Exception class is used for exceptional conditions that a user's program should catch. The Error class defines exceptions that are not expected to be caught by the user program.

71. What is the difference between throw and throws?

The throw keyword is used to explicitly raise a exception within the program. On the contrary, the throws clause is used to indicate those exceptions that are not handled by a method. Each method must explicitly specify which exceptions does not handle, so the callers of that method can guard against possible exceptions. Finally, multiple exceptions are separated by a comma.

Throw	Throws
Throw is used for throwing an exception explicitly.	To declar an exception,throws is used.
Using throw only, Checked exceptions can not be propagated.	Using throws, Checked exception can be propagated.
Throw is always used with an instance.	Throws is always used with a class.
Throw is used inside the method.	Throws is used always with the method signature.
You should not throw multiple exception	You can declare multiple exceptions.

72. What is the importance of finally block in exception handling?

A finally block will always be executed, whether or not an exception is actually thrown. Even in the case where the catch statement is missing and an exception is thrown, the finally block will still be executed. Last thing to mention is that the finally block is used to release resources like I/O buffers, database connections, etc.

Sample code below shows the finally block when exception is thrown.

Finally Block

```
01 public class DivideByZeroException
02 {
03     public static void main(String []args){
04         try{
```



```
05         int a = 1;
06         System.out.println(a/0);
07     }
08     catch(Exception exception)
09     {
10         System.out.println("exception is thrown");
11     }
12     finally
13     {
14         System.out.println("after the exception is handled");
15     }
16 }
17 }
```

73. What will happen to the Exception object after exception handling?

The Exception object will be garbage collected in the next garbage collection.

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

- **Final** keyword is used to apply restrictions on class(immutable), method(cannot override) and variable(constant).
- **Finally** is a block that always executes when the try block exits even if an unexpected exception occurs.
- **Finalize** is a method called to clean or release the resources by the Garbage Collector before destroying the object.

G.Java Applets

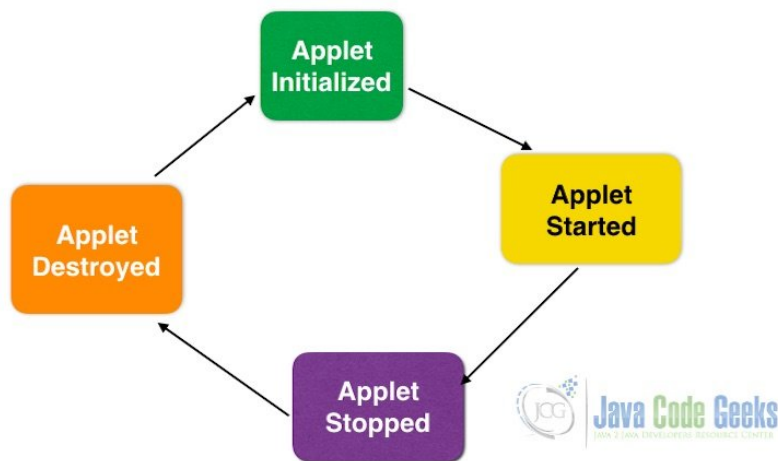
75. What is an Applet?

A java applet is program that can be included in a HTML page and be executed in a java enabled client browser. Applets are used for creating dynamic and interactive web applications.

76. Explain the life cycle of an Applet.

An applet may undergo the following states:

- **Init** : An applet is initialized each time is loaded.
- **Start** : Begin the execution of an applet.
- **Stop** : Stop the execution of an applet.
- **Destroy** : Perform a final cleanup, before unloading the applet.



Applet Lifecycle

77. What happens when an applet is loaded?

First of all, an instance of the applet's controlling class is created. Then, the applet initializes itself and finally, it starts running.

78. What is the difference between an Applet and a Java Application?

Applets are executed within a java enabled browser, but a Java application is a standalone Java program that can be executed outside of a browser. However, they both require the existence of a Java Virtual Machine (JVM). Furthermore, a Java application requires a main method with a specific signature, in order to start its execution. Java applets do not need such a method to start their execution. Finally, Java applets

typically use a restrictive security policy, while Java applications usually use more relaxed security policies.

79. What are the restrictions imposed on Java applets?

Mostly due to security reasons, the following restrictions are imposed on Java applets:

- An applet cannot load libraries or define native methods.
- An applet cannot ordinarily read or write files on the execution host.
- An applet cannot read certain system properties.
- An applet cannot make network connections except to the host that it came from.
- An applet cannot start any program on the host that is executing it.

80. What are untrusted applets?

Untrusted applets are those Java applets that cannot access or execute local system files. By default, all downloaded applets are considered as untrusted.

81. What is the difference between applets loaded over the internet and applets loaded via the file system?

Regarding the case where an applet is loaded over the internet, the applet is loaded by the applet classloader and is subject to the restrictions enforced by the applet security manager. Regarding the case where an applet is loaded from the client's local disk, the applet is loaded by the file system loader. Applets loaded via the file system are allowed to read files, write files and to load libraries on the client. Also, applets loaded via the file system are allowed to execute processes and finally, applets loaded via the file system are not passed through the byte code verifier.

82. What is the applet class loader, and what does it provide?

When an applet is loaded over the internet, the applet is loaded by the applet classloader. The class loader enforces the Java namespace hierarchy. Also, the class loader guarantees that a unique namespace exists for classes that come from the local file system, and that a unique namespace exists for each network source. When a browser loads an applet over the net, that applet's classes are placed in a private namespace associated with the applet's origin. Then, those classes loaded by the class loader are passed through the verifier. The verifier checks that the class file conforms to the Java language specification. Among other things, the verifier ensures that there are no stack overflows or underflows and that the parameters to all bytecode instructions are correct.

83. What is the applet security manager, and what does it provide?

The applet security manager is a mechanism to impose restrictions on Java applets. A browser may only have one security manager. The security manager is established at startup, and it cannot thereafter be replaced, overloaded, overridden, or extended.

H.Swing

84. What is the difference between a Choice and a List?

A Choice is displayed in a compact form that must be pulled down, in order for a user to be able to see the list of all available choices. Only one item may be selected from a Choice. A List may be displayed in such a way that several List items are visible. A List supports the selection of one or more List items.

85. What is a layout manager?

A layout manager is used to organize the components in a container.

86. What is the difference between a Scrollbar and a JScrollPane?

A Scrollbar is a Component, but not a Container. A ScrollPane is a Container. A ScrollPane handles its own events and performs its own scrolling.

87. Which Swing methods are thread-safe?

There are only three thread-safe methods: repaint, revalidate, and invalidate.

88. Name three Component subclasses that support painting.

The Canvas, Frame, Panel, and Applet classes support painting.

89. What is clipping?

Clipping is defined as the process of confining paint operations to a limited area or shape.

90. What is the difference between a MenuItem and a CheckboxMenuItem?



The `CheckboxMenuItem` class extends the `MenuItem` class and supports a menu item that may be either checked or unchecked.

91. How are the elements of a `BorderLayout` organized?

The elements of a `BorderLayout` are organized at the borders (North, South, East, and West) and the center of a container.

92. How are the elements of a `GridBagLayout` organized?

The elements of a `GridBagLayout` are organized according to a grid. The elements are of different sizes and may occupy more than one row or column of the grid. Thus, the rows and columns may have different sizes.

93. What is the difference between a `Window` and a `Frame`?

The `Frame` class extends the `Window` class and defines a main application window that can have a menu bar.

94. What is the relationship between clipping and repainting?

When a window is repainted by the AWT painting thread, it sets the clipping regions to the area of the window that requires repainting.

95. What is the relationship between an event-listener interface and an event-adaptor class?

An event-listener interface defines the methods that must be implemented by an event handler for a particular event. An event adapter provides a default implementation of an event-listener interface.

96. How can a GUI component handle its own events?

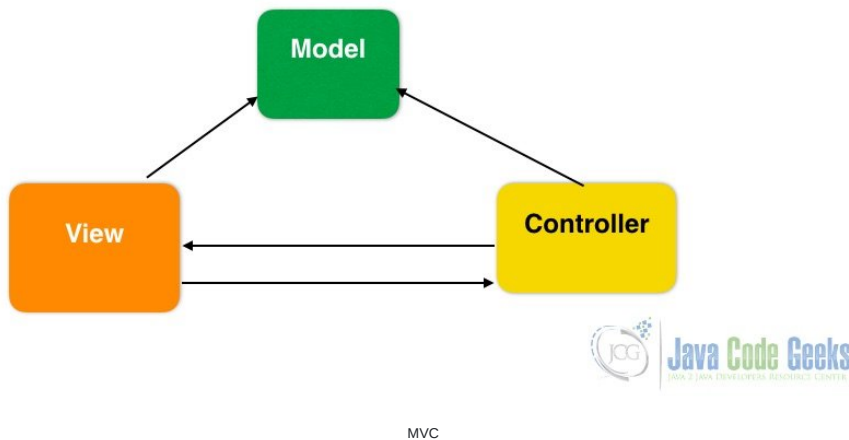
A GUI component can handle its own events, by implementing the corresponding event-listener interface and adding itself as its own event listener.

97. What advantage do Java's layout managers provide over traditional windowing systems?

Java uses layout managers to lay out components in a consistent manner, across all windowing platforms. Since layout managers are not tied to absolute sizing and positioning, they are able to accommodate platform-specific differences among windowing systems.

98. What is the design pattern that Java uses for all Swing components?

The design pattern used by Java for all Swing components is the Model View Controller (MVC) pattern.



I.JDBC

99. What is JDBC?

JDBC is an abstraction layer that allows users to choose between databases. JDBC enables developers to write database applications in Java, without having to concern themselves with the underlying details of a particular database.

100. What are the JDBC API components?



The java.sql package contains:

Interfaces:

- Driver
- Connection
- Statement
- PreparedStatement
- CallableStatement
- ResultSet

Classes:

- DriverManager
- SQLException

101. Explain the role of Driver in JDBC.

The JDBC Driver provides vendor-specific implementations of the abstract classes provided by the JDBC API. Each driver must provide implementations for the following interfaces of the java.sql package: Connection, Statement, PreparedStatement, CallableStatement, ResultSet and Driver.

102. What is JDBC Connection interface?

Connection interface maintains a session with the database. SQL statements are executed and results are returned within the context of a connection. A Connection object's database is able to provide information describing its tables, its supported SQL grammar, its stored procedures, the capabilities of this connection, and so on. This information is obtained with the `getMetaData` method.

103. What does Connection pooling mean?

The interaction with a database can be costly, regarding the opening and closing of database connections. Especially, when the number of database clients increases, this cost is very high and a large number of resources is consumed. A pool of database connections is obtained at start up by the application server and is maintained in a pool. A request for a connection is served by a connection residing in the pool. In the end of the connection, the request is returned to the pool and can be used to satisfy future requests.

104. What is the role of JDBC DriverManager class?

The DriverManager provides the user with a basic service for managing a set of JDBC drivers. It maintains contact with the available drivers and establishes a database connection with an appropriate one.

105. What is the purpose Class.forName method?

This method is used to load the driver that will establish a connection to the database.

Sample Class ClassLoader is shown below to demonstrate the usage of Class.forName() method.

Class.forName

```
01 | public class ClassLoader
02 | {
03 |
04 |     public static void main(String[] args) {
05 |
06 |         try
07 |         {
08 |             Class cls = Class.forName("BasicClass");
09 |             ....
10 |
11 |             System.out.println("Class = " + cls.getName());
12 |
13 |         }
14 |
15 |         catch(ClassNotFoundException exception)
16 |         {
17 |             System.out.println(exception.toString());
18 |         }
19 |
20 |
21 | }
```

106. What is the advantage of PreparedStatement over Statement?

PreparedStatement is precompiled and thus, performance is much better. Also, PreparedStatement objects can be reused with different input values to their queries.

107. What is the use of CallableStatement?

A CallableStatement is used to execute stored procedures. Stored procedures are stored and offered by a database. Stored procedures may take input values from the user and may return a result. The usage of stored procedures is highly encouraged, because it offers security and modularity. The method that prepares a CallableStatement is `CallableStatement.prepareCall()`.

108. What do you mean by batch processing in JDBC?

Batch processing groups related SQL statements and execute multiple queries when the batch size reaches a desired threshold. This makes



the performance faster.

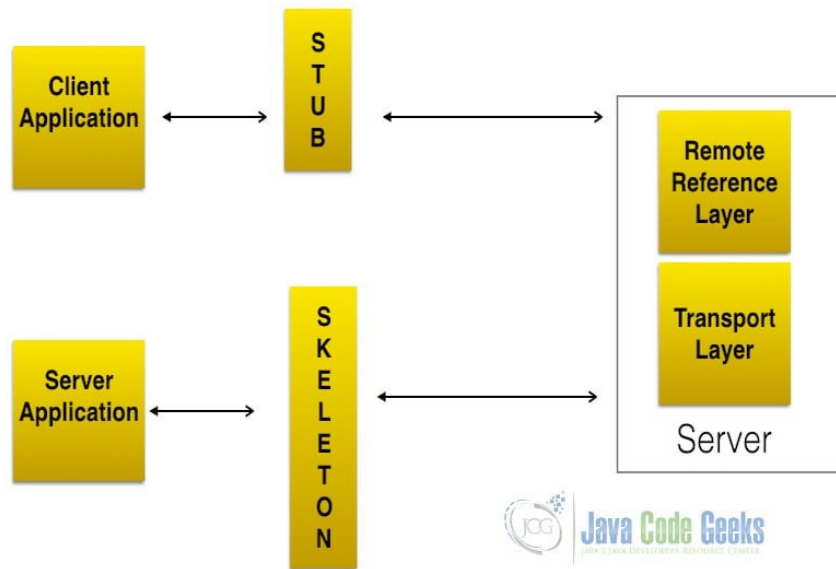
J.Remote Method Invocation (RMI)

109. What is RMI?

The Java Remote Method Invocation (Java RMI) is a Java API that performs the object-oriented equivalent of remote procedure calls (RPC), with support for direct transfer of serialized Java classes and distributed garbage collection. Remote Method Invocation (RMI) can also be seen as the process of activating a method on a remotely running object. RMI offers location transparency because a user feels that a method is executed on a locally running object. Check some RMI Tips here.

110. What is the basic principle of RMI architecture?

The RMI architecture is based on a very important principle which states that the definition of the behavior and the implementation of that behavior, are separate concepts. RMI allows the code that defines the behavior and the code that implements the behavior to remain separate and to run on separate JVMs.



RMI Architecture

111. What are the layers of RMI Architecture?

The RMI architecture consists of the following layers:

- Stub and Skeleton layer : This layer lies just beneath the view of the developer. This layer is responsible for intercepting method calls made by the client to the interface and redirect these calls to a remote RMI Service.
- Remote Reference Layer : The second layer of the RMI architecture deals with the interpretation of references made from the client to the server's remote objects. This layer interprets and manages references made from clients to the remote service objects. The connection is a one-to-one (unicast) link.
- Transport layer : This layer is responsible for connecting the two JVM participating in the service. This layer is based on TCP/IP connections between machines in a network. It provides basic connectivity, as well as some firewall penetration strategies.

112. What is the role of Remote Interface in RMI?

The Remote interface serves to identify interfaces whose methods may be invoked from a non-local virtual machine. Any object that is a remote object must directly or indirectly implement this interface. A class that implements a remote interface should declare the remote interfaces being implemented, define the constructor for each remote object and provide an implementation for each remote method in all remote interfaces.

113. What is the role of the java.rmi.Naming Class?

The java.rmi.Naming class provides methods for storing and obtaining references to remote objects in the remote object registry. Each method of the Naming class takes as one of its arguments a name that is a String in URL format.

114. What is meant by binding in RMI?

Binding is the process of associating or registering a name for a remote object, which can be used at a later time, in order to look up that remote object. A remote object can be associated with a name using the bind or rebind methods of the Naming class.



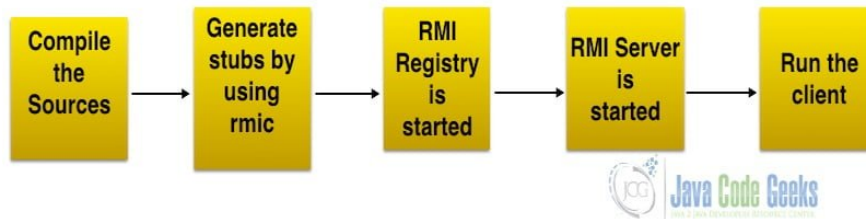
115. What is the difference between using bind() and rebind() methods of Naming Class?

The bind method bind is responsible for binding the specified name to a remote object, while the rebind method is responsible for rebinding the specified name to a new remote object. In case a binding exists for that name, the binding is replaced.

116. What are the steps involved to make work a RMI program?

The following steps must be involved in order for a RMI program to work properly:

- Compilation of all source files.
- Generation of the stubs using rmic.
- Start the rmiregistry.
- Start the RMI Server.
- Run the client program.



RMI Flow

117. What is the role of stub in RMI?

A stub for a remote object acts as a client's local representative or proxy for the remote object. The caller invokes a method on the local stub, which is responsible for executing the method on the remote object. When a stub's method is invoked, it undergoes the following steps:

- It initiates a connection to the remote JVM containing the remote object.
- It marshals the parameters to the remote JVM.
- It waits for the result of the method invocation and execution.
- It unmarshals the return value or an exception if the method has not been successfully executed.
- It returns the value to the caller.

118. What is DGC and how does it work?

DGC stands for Distributed Garbage Collection. Remote Method Invocation (RMI) uses DGC for automatic garbage collection. Since RMI involves remote object references across JVMs, garbage collection can be quite difficult. DGC uses a reference counting algorithm to provide automatic memory management for remote objects.

119. What is the purpose of using RMISecurityManager in RMI?

RMISecurityManager provides a security manager that can be used by RMI applications, which use downloaded code. The class loader of RMI will not download any classes from remote locations, if the security manager has not been set.

120. Explain Marshalling and demarshalling.

When an application wants to pass its memory objects across a network to another host or persist it to storage, the in-memory representation must be converted to a suitable format. This process is called marshalling and the revert operation is called demarshalling.

121. Explain Serialization and Deserialization.

Java provides a mechanism, called object serialization where an object can be represented as a sequence of bytes and includes the object's data, as well as information about the object's type, and the types of data stored in the object. Thus, serialization can be seen as a way of flattening objects, in order to be stored on disk, and later, read back and reconstituted. Deserialisation is the reverse process of converting an object from its flattened state to a live object.

K.Servlets

122. What is a Servlet?

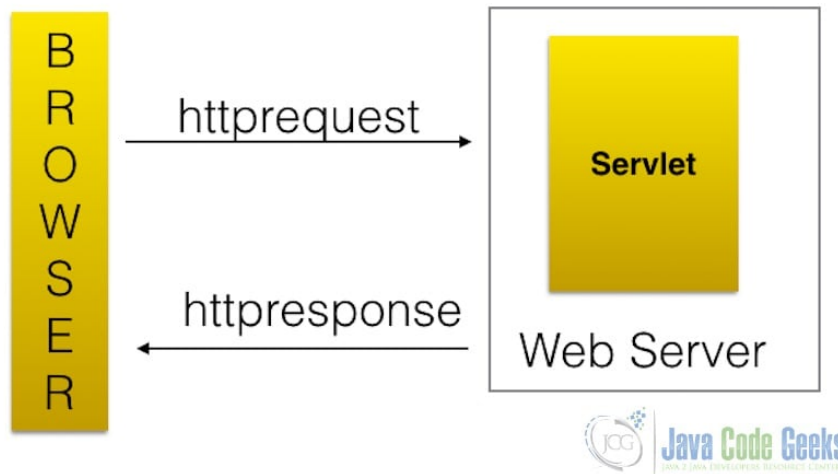
The servlet is a Java programming language class used to process client requests and generate dynamic web content. Servlets are mostly used to process or store data submitted by an HTML form, provide dynamic content and manage state information that does not exist in the stateless HTTP protocol.

123. Explain the architecture of a Servlet.

The core abstraction that must be implemented by all servlets is the `javax.servlet.Servlet` interface. Each servlet must implement it either



directly or indirectly, either by extending `javax.servlet.GenericServlet` or `javax.servlet.http.HttpServlet`. Finally, each servlet is able to serve multiple requests in parallel using multithreading.



Servlet Architecture

124. What is the difference between an Applet and a Servlet?

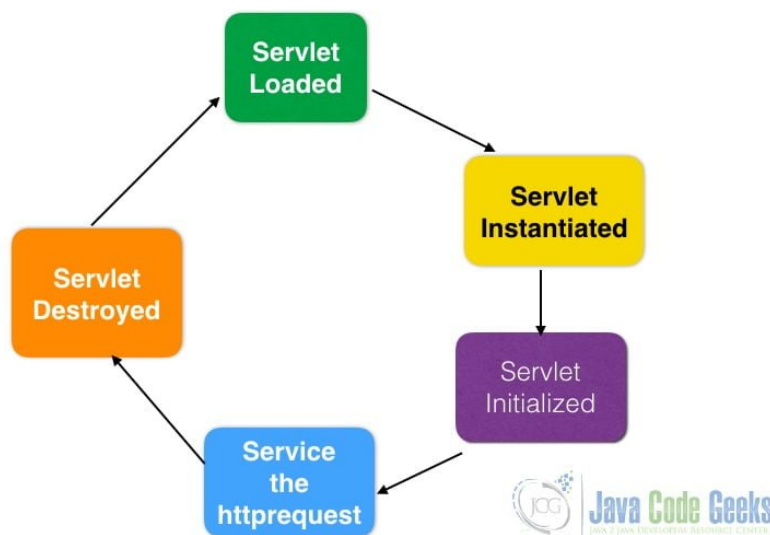
An Applet is a client side java program that runs within a Web browser on the client machine. On the other hand, a servlet is a server side component that runs on the web server. An applet can use the user interface classes, while a servlet does not have a user interface. Instead, a servlet waits for client's HTTP requests and generates a response in every request.

125. What is the difference between GenericServlet and HttpServlet?

`GenericServlet` is a generalized and protocol-independent servlet that implements the `Servlet` and `ServletConfig` interfaces. Those servlets extending the `GenericServlet` class shall override the `service` method. Finally, in order to develop an HTTP servlet for use on the Web that serves requests using the HTTP protocol, your servlet must extend the `HttpServlet` instead. Check Servlet examples here.

126. Explain the life cycle of a Servlet.

On every client's request, the Servlet Engine loads the servlets and invokes its `init` methods, in order for the servlet to be initialized. Then, the Servlet object handles all subsequent requests coming from that client, by invoking the `service` method for each request separately. Finally, the servlet is removed by calling the server's `destroy` method.



Servlet Lifecycle

127. What is the difference between `doGet()` and `doPost()`?

`doGET`: The GET method appends the name-value pairs on the request's URL. Thus, there is a limit on the number of characters and

subsequently on the number of values that can be used in a client's request. Furthermore, the values of the request are made visible and thus, sensitive information must not be passed in that way. doPOST: The POST method overcomes the limit imposed by the GET request, by sending the values of the request inside its body. Also, there is no limitations on the number of values to be sent across. Finally, the sensitive information passed through a POST request is not visible to an external client.

The code below shows the BasicServlet class which has doGet and doPost methods to be implemented.

Get and Post methods

```
01 public class BasicServlet extends HttpServlet
02 {
03     public void doGet(HttpServletRequest request, HttpServletResponse response)
04         throws ServletException, IOException
05     {
06     }
07
08     public void doPost(HttpServletRequest request, HttpServletResponse response)
09         throws ServletException, IOException
10     {
11     }
12 }
13
14
15
16
17
```

128. What is meant by a Web Application?

A Web application is a dynamic extension of a Web or application server. There are two types of web applications: presentation-oriented and service-oriented. A presentation-oriented Web application generates interactive web pages, which contain various types of markup language and dynamic content in response to requests. On the other hand, a service-oriented web application implements the endpoint of a web service. In general, a Web application can be seen as a collection of servlets installed under a specific subset of the server's URL namespace.

129. What is a Server Side Include (SSI)?

Server Side Includes (SSI) is a simple interpreted server-side scripting language, used almost exclusively for the Web, and is embedded with a servlet tag. The most frequent use of SSI is to include the contents of one or more files into a Web page on a Web server. When a Web page is accessed by a browser, the Web server replaces the servlet tag in that Web page with the hyper text generated by the corresponding servlet.

130. What is Servlet Chaining?

Servlet Chaining is the method where the output of one servlet is sent to a second servlet. The output of the second servlet can be sent to a third servlet, and so on. The last servlet in the chain is responsible for sending the response to the client.

131. How do you find out what client machine is making a request to your servlet?

The ServletRequest class has functions for finding out the IP address or host name of the client machine. getRemoteAddr() gets the IP address of the client machine and getRemoteHost() gets the host name of the client machine. See example here.

132. What is the structure of the HTTP response?

The HTTP response consists of three parts:

- **Status Code:** describes the status of the response. It can be used to check if the request has been successfully completed. In case the request failed, the status code can be used to find out the reason behind the failure. If your servlet does not return a status code, the success status code, HttpServletResponse.SC_OK, is returned by default.
- **HTTP Headers:** they contain more information about the response. For example, the headers may specify the date/time after which the response is considered stale, or the form of encoding used to safely transfer the entity to the user. See how to retrieve headers in Servlet here.
- **Body:** it contains the content of the response. The body may contain HTML code, an image, etc. The body consists of the data bytes transmitted in an HTTP transaction message immediately following the headers.

133. What is a cookie?

A cookie is a bit of information that the Web server sends to the browser. The browser stores the cookies for each Web server in a local file. In a future request, the browser, along with the request, sends all stored cookies for that specific Web server.

134. What is the difference between session and cookie?

The differences between session and a cookie are the following:

- The session should work, regardless of the settings on the client browser. The client may have chosen to disable cookies. However, the sessions still work, as the client has no ability to disable them in the server side.
- The session and cookies also differ in the amount of information they can store. The HTTP session is capable of storing any Java object, while a cookie can only store String objects.

135. Which protocol will be used by browser and servlet to communicate?

The browser communicates with a servlet by using the HTTP protocol.



136. What is HTTP Tunneling?

HTTP Tunneling is a technique by which, communications performed using various network protocols are encapsulated using the HTTP or HTTPS protocols. The HTTP protocol therefore acts as a wrapper for a channel that the network protocol being tunneled uses to communicate. The masking of other protocol requests as HTTP requests is HTTP Tunneling.

137. What's the difference between sendRedirect and forward methods?

The sendRedirect method creates a new request, while the forward method just forwards a request to a new target. The previous request scope objects are not available after a redirect, because it results in a new request. On the other hand, the previous request scope objects are available after forwarding. Finally, in general, the sendRedirect method is considered to be slower compare to the forward method.

SendRedirect	Forward
This method sends a new request always. Th is because it uses the URL bar of the browser for redirecting.	This method sends the request to another resource by forwarding it.
This method is used at client side.	This method is usead at server side.
This method is used inside and outside the web server.	This method is used inside the web server only.

138. What is URL Encoding and URL Decoding?

The URL encoding procedure is responsible for replacing all the spaces and every other extra special character of a URL, into their corresponding Hex representation. In correspondence, URL decoding is the exact opposite procedure.

139. What is Request Dispatcher?

Servlet Request Dispatcher is an interface whose implementation defines that an object can dispatch requests to any resource (such as HTML, Image, JSP, Servlet etc.) on the server. Another advantage of this interface is that it is used in two cases:

- To include the response of one Servlet into another (i.e. the client gets the response of both Servlets)
- To forward the client request to another Servlet to honor the request (i.e. the client calls a Servlet but the response to client is given by another Servlet)

L.JSP

140. What is a JSP Page?

A Java Server Page (JSP) is a text document that contains two types of text: static data and JSP elements. Static data can be expressed in any text-based format, such as HTML or XML. JSP is a technology that mixes static content with dynamically-generated content. See JSP example here.

141. How are the JSP requests handled?

On the arrival of a JSP request, the browser first requests a page with a .jsp extension. Then, the Web server reads the request and using the JSP compiler, the Web server converts the JSP page into a servlet class. Notice that the JSP file is compiled only on the first request of the page, or if the JSP file has changed. The generated servlet class is invoked, in order to handle the browser's request. Once the execution of the request is over, the servlet sends a response back to the client. See how to get Request parameters in a JSP.

142. What are the advantages of JSP?

The advantages of using the JSP technology are shown below:

- JSP pages are dynamically compiled into servlets and thus, the developers can easily make updates to presentation code.
- JSP pages can be pre-compiled.
- JSP pages can be easily combined to static templates, including HTML or XML fragments, with code that generates dynamic content.
- Developers can offer customized JSP tag libraries that page authors access using an XML-like syntax.
- Developers can make logic changes at the component level, without editing the individual pages that use the application's logic.

143. What are Directives?

Directives are instructions that are processed by the JSP engine, when the page is compiled to a servlet. Directives are used to set page-level instructions, insert data from external files, and specify custom tag libraries.

144. What are the different types of Directives available in JSP?

Directives are defined between < %@ and % >. The different types of directives are shown below:

- Include directive: it is used to include a file and merges the content of the file with the current page.
- Page directive: it is used to define specific attributes in the JSP page, like error page and buffer.
- Taglib: it is used to declare a custom tag library which is used in the page.

145. What are JSP actions?

JSP actions use constructs in XML syntax to control the behavior of the servlet engine. They are executed when a JSP page is requested. They can be dynamically inserted into a file, re-use JavaBeans components, forward the user to another page, or generate HTML for the Java



plugin. Some of the available actions are listed below:

- `jsp:include` includes a file, when the JSP page is requested.
- `jsp:useBean` finds or instantiates a `JavaBean`.
- `jsp:setProperty` sets the property of a `JavaBean`.
- `jsp:getProperty` gets the property of a `JavaBean`.
- `jsp:forward` forwards the requester to a new page.
- `jsp:plugin` generates browser-specific code.

146. What are Scriptlets?

In Java Server Pages (JSP) technology, a scriptlet is a piece of Java-code embedded in a JSP page. The scriptlet is everything inside the tags. Between these tags, a user can add any valid scriptlet.

147. What are Declarations?

Declarations are similar to variable declarations in Java. Declarations are used to declare variables for subsequent use in expressions or scriptlets. To add a declaration, you must use the sequences to enclose your declarations.

Sample code is added below to show the JSP declarations.

Declarations

```
1 | <%! int j = 0; %>
2 | <%! int d, e, f; %>
3 | <%! Shape a = new Shape(); %>
```

148. What are Expressions?

A JSP expression is used to insert the value of a scripting language expression, converted into a string, into the data stream returned to the client, by the web server. Expressions are defined between `<% =` and `%>` tags.

JSP Expression

```
1 | <html>
2 |   <head><title>My Blog</title></head>
3 |
4 |   <body>
5 |     <p>Today's Date is: <%= (new java.util.Date()).toLocaleString()%></p>
6 |   </body>
7 | </html>
```

149. What is meant by implicit objects and what are they?

JSP implicit objects are those Java objects that the JSP Container makes available to developers in each page. A developer can call them directly, without being explicitly declared. JSP Implicit Objects are also called pre-defined variables. The following objects are considered implicit in a JSP page:

- `application`
- `page`
- `request`
- `response`
- `session`
- `exception`
- `out`
- `config`
- `pageContext`

JSP sample tags for disabling session is shown below:

Disabling Session

```
1 | <%@ page session="false" %>
```

150. What are the different tags provided in JSTL?

There are 5 type of JSTL tags:

Core:

- Variable support
- Flow control
- URL management
- Miscellaneous

XML:

- Core
- Flow control
- Transformation

Internationalization:



- Locale
- Message formatting
- Number and date formatting

Database:

- SQL

Functions:

- Collection length
- String manipulation

Want more Java interview questions?

Are you still with us? Wow, that was a huge article about different types Java Interview Questions. If you enjoyed this, then **subscribe to our newsletter** to enjoy weekly updates and complimentary whitepapers! Also, check out our **courses** for more advanced training!

So, what other Java Interview Questions are there? Let us know in the comments and we will include them in the article! Happy coding!

Java Interview Questions was last updated on Aug. 07th, 2019

Tagged with: [INTERVIEW](#) [INTERVIEW QUESTIONS](#) [JAVA APPLETS](#) [JDBC](#) [JSP](#) [RMI](#) [SERVLETS](#) [SWING](#) [ULTIMATE](#)



(+4 rating, 6 votes)

You need to be a registered member to rate this. 🗨️ 152 Comments 👁️ 32531 Views 🐦 Tweet it!

Do you want to know how to develop your skillset to become a **Java Rockstar**?



Subscribe to our newsletter to start Rocking **right now!**

To get you started we give you our best selling eBooks for **FREE!**

1. JPA Mini Book
2. JVM Troubleshooting Guide
3. JUnit Tutorial for Unit Testing
4. Java Annotations Tutorial
5. Java Interview Questions
6. Spring Interview Questions
7. Android UI Design

and many more

☐ I agree to the Terms and Privacy Policy

[Sign up](#)

LIKE THIS ARTICLE? READ MORE FROM JAVA CODE GEEKS



152 Leave a Reply



Join the discussion...

128 24 1 ⚡ 🔥



138

This site uses Akismet to reduce spam. [Learn how your comment data is processed.](#)

Subscribe ▾

▲ newest ▲ oldest ▲ most voted



Guest

aerobless



That list seems really good. I study computer science and it'll definitely come in handy once I start looking for a job.

+ -1 - Reply

🕒 5 years ago ^



Member

tomi crow



Hope this will help you for java interview more...

<http://net-informations.com/java/cjava/default.htm>

Crow

+ 0 - Reply

🕒 2 years ago



Guest

user



good luck for you

+ 0 - Reply

🕒 10 days ago



Guest

leon



One small error on hashCode and equals: if 2 instances are equal, they must provide the same hashCode, BUT two different instances may return the same hashCode.

Having hashCode always return 1 or some other fixed number satisfies the contract, although it would kill the performance of HashMap and other hash related functionality.

The hashCode just determines in which hashmap bucket the instance is put, and within that bucket the correct instance is then looked up with the much more expensive equals.

+ 0 - Reply

🕒 5 years ago





teoman perfect post, thanks for sharing.



+ 0 -

[Reply](#)

🕒 5 years ago

Guest



Madiraju Krishna Chaitanya



Nice Post.Thanks for sharing this with us.

+ 0 -

[Reply](#)

🕒 5 years ago

Guest



Sumit Bisht



This is correct, but many topics are outdated here.

+ 2 -

[Reply](#)

🕒 5 years ago ^

Guest



Jeffrey Burch



Yeah, there should be some questions about Java 8 features. It will tell if the candidate puts any effort into self improvement.

+ 2 -

[Reply](#)

🕒 5 years ago

Guest



Nikitha



Really perfect post, thanks for sharing with us.

+ 0 -

[Reply](#)

🕒 5 years ago

Guest



Robert Martin



This is a quality peace of work. Thank you for the effort.

+ 0 -

[Reply](#)

🕒 5 years ago

Guest



Vlad



Thanks to the author
I appreciate your efforts to write everything in one place

+ 0 -

[Reply](#)

🕒 5 years ago

Guest



raghu kumar challa



good very useful to all fresh engineering graduates and experianced to get into job easily.

+ -1 -

[Reply](#)

🕒 5 years ago

Guest



M.Hagras



Thanks a lot, it is very useful

+ 0 -

[Reply](#)

🕒 4 years ago

Guest



Tk



Very good work. It would be really great if you could add more questions especially on the new versions of Java. Also, few implementation examples can be posted for different concept. You have done a great job.

+ 0 -

[Reply](#)

🕒 4 years ago

Guest



RAVIKUMAR.M



It is very useful for job seekers to get a great job. thanks for sharing with us.

Guest



+ -1 - [Reply](#)

🕒 4 years ago



Ruth



Thanks so much for this compilation. It is very useful.

Guest

+ 0 - [Reply](#)

🕒 4 years ago



Nazim



Thanks, but found some inaccuracies:
Q33 Other threads are able to modify Collection of some Iterator, but next called method on Iterator will throw ConcurrentModificationException
Q37 It is recommended not to override finalize() method in order to release resource as JVM does not guarantee this method to be invoked.

Guest

+ 0 - [Reply](#)

🕒 4 years ago



satish shirale



Thanks, I refreshed my java knowledge once again.

Guest

+ 0 - [Reply](#)

🕒 4 years ago



Guarav



Thanks ... very useful

Guest

+ 0 - [Reply](#)

🕒 4 years ago



Madhu



Thanks to the author ! Keep updating !

Guest

+ 1 - [Reply](#)

🕒 4 years ago



Jun



nice job. Thank You. This tutorial is very useful for people who seek job in programming. I just love coding.

Guest

+ 0 - [Reply](#)

🕒 4 years ago



Akansha



nice information

Guest

+ 0 - [Reply](#)

🕒 4 years ago



Ahmed Mansour



nice information

Guest

+ 0 - [Reply](#)

🕒 4 years ago

Load More Comments



NEWSLETTER

Insiders are already enjoying weekly updates and complimentary whitepapers!

Join them now to gain exclusive

access to the latest news in the Java world, as well as insights about Android, Scala, Groovy and other related technologies.

☐ I agree to the [Terms and Privacy Policy](#)

JOIN US



With **1,240,600** monthly unique visitors and over **500** authors we are placed among the top Java related sites around. Constantly being on the lookout for partners; we encourage you to join us. So if you have a blog with unique and interesting content then you should check out our **JCG** partners program. You can also be a **guest writer** for Java Code Geeks and hone your writing skills!



KNOWLEDGE BASE[Courses](#)[Examples](#)[Minibooks](#)[Resources](#)[Tutorials](#)**PARTNERS**[Mkyong](#)**THE CODE GEEKS NETWORK**[.NET Code Geeks](#)[Java Code Geeks](#)[System Code Geeks](#)[Web Code Geeks](#)**HALL OF FAME**["Android Full Application Tutorial" series](#)[11 Online Learning websites that you should check out](#)[Advantages and Disadvantages of Cloud Computing – Cloud computing pros and cons](#)[Android Google Maps Tutorial](#)[Android JSON Parsing with Gson Tutorial](#)[Android Location Based Services Application – GPS location](#)[Android Quick Preferences Tutorial](#)[Difference between Comparator and Comparable in Java](#)[GWT 2 Spring 3 JPA 2 Hibernate 3.5 Tutorial](#)[Java Best Practices – Vector vs ArrayList vs HashSet](#)**ABOUT JAVA CODE GEEKS**

JCGs (Java Code Geeks) is an independent online community focused on creating the ultimate Java to Java developers resource center; targeted at the technical architect, technical team lead (senior developer), project manager and junior developers alike. JCGs serve the Java, SOA, Agile and Telecom communities with daily news written by domain experts, articles, tutorials, reviews, announcements, code snippets and open source projects.

DISCLAIMER

All trademarks and registered trademarks appearing on Java Code Geeks are the property of their respective owners. Java is a trademark or registered trademark of Oracle Corporation in the United States and other countries. Examples Java Code Geeks is not connected to Oracle Corporation and is not sponsored by Oracle Corporation.