Java Interview Question

Designed by TechoLamror

Java Full Detail | Java Program | Java Topic Wise Interview Question | How to do work faster

1.What are the principle concepts of OOPS?

There are four principle concepts upon which object oriented design and programming rest. They are:

- Abstraction
- Polymorphism
- Inheritance
- Encapsulation

(i.e. easily remembered as A-PIE).

2.What is Abstraction?

Abstraction refers to the act of representing essential features without including the background details or explanations.

3.What is Encapsulation?

Encapsulation is a technique used for hiding the properties and behaviors of an object and allowing outside access only as appropriate. It prevents other objects from directly altering or accessing the properties or methods of the encapsulated object.

4.What is the difference between abstraction and encapsulation?

- **Abstraction** focuses on the outside view of an object (i.e. the interface) **Encapsulation** (information hiding) prevents clients from seeing it's inside view, where the behavior of the abstraction is implemented.
- **Abstraction** solves the problem in the design side while **Encapsulation** is the Implementation.
- **Encapsulation** is the deliverables of Abstraction. Encapsulation barely talks about grouping up your abstraction to suit the developer needs.

5.What is Inheritance?

- Inheritance is the process by which objects of one class acquire the properties of objects of another class.
- A class that is inherited is called a superclass.
- The class that does the inheriting is called a subclass.
- Inheritance is done by using the keyword extends.
- The two most common reasons to use inheritance are:
    - To promote code reuse
    - To use polymorphism

6.What is Polymorphism?

Polymorphism is briefly described as "one interface, many implementations." Polymorphism is a characteristic of being able to assign a different meaning or usage to something in different contexts - specifically, to allow an entity such as a variable, a function, or an object to have more than one form.

7.How does Java implement polymorphism?

(Inheritance, Overloading and Overriding are used to achieve Polymorphism in java).
Polymorphism manifests itself in Java in the form of multiple methods having the same name.

- In some cases, multiple methods have the same name, but different formal argument lists (overloaded methods).
- In other cases, multiple methods have the same name, same return type, and same formal argument list (overridden methods).

8.Explain the different forms of Polymorphism.

There are two types of polymorphism one is **Compile time polymorphism** and the other is run time polymorphism. Compile time polymorphism is method overloading. **Runtime time polymorphism** is done using inheritance and interface.
Note: *From a practical programming viewpoint, polymorphism manifests itself in three distinct forms in Java:*

- *Method overloading*
- *Method overriding through inheritance*
- *Method overriding through the Java interface*

9.What is runtime polymorphism or dynamic method dispatch?

In Java, runtime polymorphism or dynamic method dispatch is a process in which a call to an overridden method is resolved at runtime rather than at compile-time. In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.

10.What is Dynamic Binding?

Binding refers to the linking of a procedure call to the code to be executed in response to the call. Dynamic binding (also known as late binding) means that the code associated with a given procedure call is not known until the time of the call at run-time. It is associated with polymorphism and inheritance.

11.What is method overloading?

Method Overloading means to have two or more methods with same name in the same class with different arguments. The benefit of method overloading is that it allows you to implement methods that support the same semantic operation but differ by argument number or type.
Note:

- *Overloaded methods MUST change the argument list*
- *Overloaded methods CAN change the return type*
- *Overloaded methods CAN change the access modifier*

- *Overloaded methods CAN declare new or broader checked exceptions*
- *A method can be overloaded in the same class or in a subclass*

12. What is method overriding?

Method overriding occurs when sub class declares a method that has the same type arguments as a method declared by one of its superclass. The key benefit of overriding is the ability to define behavior that's specific to a particular subclass type.

Note:

- *The overriding method cannot have a more restrictive access modifier than the method being overridden (Ex: You can't override a method marked public and make it protected).*
- *You cannot override a method marked final*
- *You cannot override a method marked static*

13. What are the differences between method overloading and method overriding?

	Overloaded Method	Overridden Method
Arguments	Must change	Must not change
Return type	Can change	Can't change except for covariant returns
Exceptions	Can change	Can reduce or eliminate. Must not throw new or broader checked exceptions
Access	Can change	Must not make more restrictive (can be less restrictive)
Invocation	Reference type determines which overloaded version is selected. Happens at compile time.	Object type determines which method is selected. Happens at runtime.

14. Can overloaded methods be override too?

Yes, derived classes still can override the overloaded methods. Polymorphism can still happen. Compiler will not binding the method calls since it is overloaded, because it might be overridden now or in the future.

15. Is it possible to override the main method?

NO, because main is a static method. A static method can't be overridden in Java.

16. How to invoke a superclass version of an Overridden method?

To invoke a superclass method that has been overridden in a subclass, you must either call the method directly through a superclass instance, or use the super prefix in the subclass itself. From the point of the view of the subclass, the super prefix provides an explicit reference to the superclass' implementation of the method.

```
        // From subclass
              super.overriddenMethod();
```

17. What is super?

`super` is a keyword which is used to access the method or member variables from the superclass. If a method hides one of the member variables in its superclass, the method can refer to the hidden variable through the use of the super keyword. In the same way, if a method overrides one of the methods in its superclass, the method can invoke the overridden method through the use of the super keyword.
Note:

- *You can only go back one level.*
- *In the constructor, if you use super(), it must be the very first code, and you cannot access any* `this.xxx` *variables or methods to compute its parameters.*

18. How do you prevent a method from being overridden?

To prevent a specific method from being overridden in a subclass, use the final modifier on the method declaration, which means "this is the final implementation of this method", the end of its inheritance hierarchy.

```
    public final void exampleMethod() {
//  Method statements
}
```

19. What is an Interface?

An interface is a description of a set of methods that conforming implementing classes must have.
Note:

- *You can't mark an interface as final.*
- *Interface variables must be static.*
- *An Interface cannot extend anything but another interfaces.*

20. Can we instantiate an interface?

You can't instantiate an interface directly, but you can instantiate a class that implements an interface.

21. Can we create an object for an interface?

Yes, it is always necessary to create an object implementation for an interface. Interfaces cannot be instantiated in their own right, so you must write a class that implements the interface and fulfill all the methods defined in it.

22. Do interfaces have member variables?

Interfaces may have member variables, but these are implicitly `public`, `static`, and `final`- in other words, interfaces can declare only constants, not instance variables that are available to all implementations and may be used as key references for method arguments for example.

23. What modifiers are allowed for methods in an Interface?

Only `public` and `abstract` modifiers are allowed for methods in interfaces.

24. What is a marker interface?

Marker interfaces are those which do not declare any required methods, but signify their compatibility with certain operations. The `java.io.Serializable`interface and `Cloneable` are typical marker interfaces. These do not contain any methods, but classes must implement this interface in order to be serialized and de-serialized.

25.What is an abstract class?

Abstract classes are classes that contain one or more abstract methods. An abstract method is a method that is declared, but contains no implementation.
Note:

- *If even a single method is abstract, the whole class must be declared abstract.*
- *Abstract classes may not be instantiated, and require subclasses to provide implementations for the abstract methods.*
- *You can't mark a class as both abstract and final.*

26.Can we instantiate an abstract class?

An abstract class can never be instantiated. Its sole purpose is to be extended (subclassed).

27.What are the differences between Interface and Abstract class?

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

28.When should I use abstract classes and when should I use interfaces?

Use Interfaces when…

- You see that something in your design will change frequently.
- If various implementations only share method signatures then it is better to use Interfaces.

- you need some classes to use some methods which you don't want to be included in the class, then you go for the interface, which makes it easy to just implement and make use of the methods defined in the interface.

Use Abstract Class when…

- If various implementations are of the same kind and use common behavior or status then abstract class is better to use.
- When you want to provide a generalized form of abstraction and leave the implementation task with the inheriting subclass.
- Abstract classes are an excellent way to create planned inheritance hierarchies. They're also a good choice for nonleaf classes in class hierarchies.

29. When you declare a method as abstract, can other nonabstract methods access it?

Yes, other nonabstract methods can access a method that you declare as abstract.

30. Can there be an abstract class with no abstract methods in it?

Yes, there can be an abstract class without abstract methods.

31. What is Constructor?

- A constructor is a special method whose task is to initialize the object of its class.
- It is special because its name is the **same as the class name**.
- They do not have return types, not even **void** and therefore they cannot return values.
- They **cannot be inherited**, though a derived class can call the base class constructor.
- Constructor is invoked whenever an object of its associated class is created.

32. How does the Java default constructor be provided?

If a class defined by the code does **not** have any constructor, compiler will automatically provide one no-parameter-constructor (default-constructor) for the class in the byte code. The access modifier (public/private/etc.) of the default constructor is the same as the class itself.

33. Can constructor be inherited?

No, constructor cannot be inherited, though a derived class can call the base class constructor.

34. What are the differences between Contructors and Methods?

	Constructors	Methods
Purpose	Create an instance of a class	Group Java statements
Modifiers	Cannot be *abstract*, *final*, *native*, *static*, or *synchronized*	Can be *abstract*, *final*, *native*, *static*, or *synchronized*
Return Type	No return type, not even void	void or a valid return type
Name	Same name as the class (first letter is capitalized by convention) -- usually a noun	Any name except the class. Method names begin with a lowercase letter by convention -- usually the name of an action

this	Refers to another constructor in the same class. If used, it must be the first line of the constructor	Refers to an instance of the owning class. Cannot be used by static methods.
super	Calls the constructor of the parent class. If used, must be the first line of the constructor	Calls an overridden method in the parent class
Inheritance	Constructors are not inherited	Methods are inherited

35.How are this() and super() used with constructors?

- Constructors use *this* to refer to another constructor in the same class with a different parameter list.
- Constructors use *super* to invoke the superclass's constructor. If a constructor uses *super*, it must use it in the first line; otherwise, the compiler will complain.

36.What are the differences between Class Methods and Instance Methods?

Class Methods	Instance Methods
Class methods are methods which are declared as static. The method can be called without creating an instance of the class	Instance methods on the other hand require an instance of the class to exist before they can be called, so an instance of a class needs to be created by using the new keyword. Instance methods operate on specific instances of classes.
Class methods can only operate on class members and not on instance members as class methods are unaware of instance members.	Instance methods of the class can also not be called from within a class method unless they are being called on an instance of that class.
Class methods are methods which are declared as static. The method can be called without creating an instance of the class.	Instance methods are not declared as static.

37.How are this() and super() used with constructors?

- Constructors use *this* to refer to another constructor in the same class with a different parameter list.
- Constructors use *super* to invoke the superclass's constructor. If a constructor uses super, it must use it in the first line; otherwise, the compiler will complain.

38.What are Access Specifiers?

One of the techniques in object-oriented programming is *encapsulation*. It concerns the hiding of data in a class and making this class available only through methods. Java allows you to control access to classes, methods, and fields via so-called *access specifiers*..

39.What are Access Specifiers available in Java?

Java offers four access specifiers, listed below in decreasing accessibility:

- **Public**- *public* classes, methods, and fields can be accessed from everywhere.
- **Protected**- *protected* methods and fields can only be accessed within the same class to which the methods and fields belong, within its subclasses, and within classes of the same package.
- **Default(no specifier)-** If you do not set access to specific level, then such a class, method, or field will be accessible from inside the same package to which the class, method, or field belongs, but not from outside this package.
- **Private**- *private* methods and fields can only be accessed within the same class to which the methods and fields belong. *private* methods and fields are not visible within subclasses and are not inherited by subclasses.

Situation	`public`	`protected`	**default**	`private`
Accessible to class from same package?	yes	yes	yes	no
Accessible to class from different package?	yes	no, *unless it is a subclass*	no	no

40.What is final modifier?

The final modifier keyword makes that the programmer cannot change the value anymore. The actual meaning depends on whether it is applied to a class, a variable, or a method.

- ***final* Classes**- A final class cannot have subclasses.
- ***final* Variables**- A final variable cannot be changed once it is initialized.
- ***final* Methods**- A final method cannot be overridden by subclasses.

41.What are the uses of final method?

There are two reasons for marking a method as final:

- Disallowing subclasses to change the meaning of the method.
- Increasing efficiency by allowing the compiler to turn calls to the method into inline Java code.

42.What is static block?

Static block which exactly executed exactly once when the class is first loaded into JVM. Before going to the main method the static block will execute.

43.What are static variables?

Variables that have only one copy per class are known as static variables. They are not attached to a particular instance of a class but rather belong to a class as a whole. They are declared by using the static keyword as a modifier.

```
static type  varIdentifier;
```

where, the name of the variable is varIdentifier and its data type is specified by type.
Note: Static variables that are not explicitly initialized in the code are automatically initialized with a default value. The default value depends on the data type of the variables.

44. What is the difference between static and non-static variables?

A static variable is associated with the class as a whole rather than with specific instances of a class. Non-static variables take on unique values with each object instance.

45. What are static methods?

Methods declared with the keyword static as modifier are called static methods or class methods. They are so called because they affect a class as a whole, not a particular instance of the class. Static methods are always invoked without reference to a particular instance of a class.
Note:The use of a static method suffers from the following restrictions:

- *A static method can only call other static methods.*
- *A static method must only access static data.*
- *A static method **cannot** reference to the current object using keywords super or this.*

46. What is an Iterator ?

- The Iterator interface is used to step through the elements of a Collection.
- Iterators let you process each element of a Collection.
- Iterators are a generic way to go through all the elements of a Collection no matter how it is organized.
- Iterator is an Interface implemented a different way for every Collection.

47. How do you traverse through a collection using its Iterator?

To use an iterator to traverse through the contents of a collection, follow these steps:

- Obtain an iterator to the start of the collection by calling the collectionâ€™s ***iterator()*** method.
- Set up a loop that makes a call to ***hasNext()***. Have the loop iterate as long as ***hasNext()*** returns **true**.
- Within the loop, obtain each element by calling **next()**.

48. How do you remove elements during Iteration?

Iterator also has a method ***remove()*** when remove is called, the current element in the iteration is deleted.

49. What is the difference between Enumeration and Iterator?

Enumeration	Iterator
Enumeration doesn't have a remove() method	Iterator has a remove() method
Enumeration acts as Read-only interface, because it has the methods only to traverse and fetch the objects	Can be *abstract*, *final*, *native*, *static*, or *synchronized*

Note: So Enumeration is used whenever we want to make Collection objects as Read-only.

50. How is ListIterator?

ListIterator is just like Iterator, except it allows us to access the collection in either the forward or backward direction and lets us modify an element

51.What is the List interface?

- The List interface provides support for ordered collections of objects.
- Lists may contain duplicate elements.

52.What are the main implementations of the List interface ?

The main implementations of the List interface are as follows :

- **ArrayList** : Resizable-array implementation of the List interface. The best all-around implementation of the List interface.
- **Vector** : Synchronized resizable-array implementation of the List interface with additional "legacy methods."
- **LinkedList** : Doubly-linked list implementation of the List interface. May provide better performance than the ArrayList implementation if elements are frequently inserted or deleted within the list. Useful for queues and double-ended queues (deques).

53.What are the advantages of ArrayList over arrays ?

Some of the advantages ArrayList has over arrays are:

- It can grow dynamically
- It provides more powerful insertion and search mechanisms than arrays.

54.Difference between ArrayList and Vector ?

ArrayList	Vector
ArrayList is **NOT** synchronized by default.	Vector List is synchronized by default.
ArrayList can use only Iterator to access the elements.	Vector list can use Iterator and Enumeration Interface to access the elements.
The ArrayList increases its array size by 50 percent if it runs out of room.	A Vector defaults to doubling the size of its array if it runs out of room
ArrayList has no default size.	While vector has a default size of 10.

55.How to obtain Array from an ArrayList ?

Array can be obtained from an ArrayList using ***toArray()*** method on ArrayList.

```
List arrayList = new ArrayList();
arrayList.add(â€¦
```

```
ObjectÂ  a[] = arrayList.toArray();
```

56.Why insertion and deletion in ArrayList is slow compared to LinkedList ?

- **ArrayList** internally uses and array to store the elements, when that array gets filled by inserting elements a new array of roughly 1.5 times the size of the original array is created and all the data of old array is copied to new array.
- During deletion, all elements present in the array after the deleted elements have to be moved one step back to fill the space created by deletion. In linked list data is stored in nodes that have reference to the previous node and the next node so adding element is simple as creating the node an updating the next pointer on the last node and the previous pointer on the new node. Deletion in linked list is fast because it involves only updating the next pointer in the node before the deleted node and updating the previous pointer in the node after the deleted node.

57.Why are Iterators returned by ArrayList called Fail Fast ?

Because, if list is structurally modified at any time after the iterator is created, in any way except through the iterator's own remove or add methods, the iterator will throw a ConcurrentModificationException. Thus, in the face of concurrent modification, the iterator fails quickly and cleanly, rather than risking arbitrary, non-deterministic behavior at an undetermined time in the future.

58.How do you decide when to use ArrayList and When to use LinkedList?

If you need to support random access, without inserting or removing elements from any place other than the end, then ArrayList offers the optimal collection. If, however, you need to frequently add and remove elements from the middle of the list and only access the list elements sequentially, then LinkedList offers the better implementation.

59.What is the Set interface ?

- The Set interface provides methods for accessing the elements of a finite mathematical set
- Sets do not allow duplicate elements
- Contains no methods other than those inherited from Collection
- It adds the restriction that duplicate elements are prohibited
- Two Set objects are equal if they contain the same elements

60.What are the main Implementations of the Set interface ?

The main implementations of the List interface are as follows:

- HashSet
- TreeSet
- LinkedHashSet
- EnumSet

61.What is a HashSet ?

- A HashSet is an unsorted, unordered Set.

- It uses the hashcode of the object being inserted (so the more efficient your hashcode() implementation the better access performance you'll get).
- Use this class when you want a collection with no duplicates and you don't care about order when you iterate through it.

62.What is a TreeSet ?

TreeSet is a Set implementation that keeps the elements in sorted order. The elements are sorted according to the natural order of elements or by the comparator provided at creation time.

63.What is an EnumSet ?

An EnumSet is a specialized set for use with enum types, all of the elements in the EnumSet type that is specified, explicitly or implicitly, when the set is created.

64.Difference between HashSet and TreeSet ?

HashSet	TreeSet
HashSet is under set interface i.e. it does not guarantee for either sorted order or sequence order.	TreeSet is under set i.e. it provides elements in a sorted order (acceding order).
We can add any type of elements to hash set.	We can add only similar types of elements to tree set.

65.What is a Map ?

- A map is an object that stores associations between keys and values (key/value pairs).
- Given a key, you can find its value. Both keys and values are objects.
- The keys must be unique, but the values may be duplicated.
- Some maps can accept a null key and null values, others cannot.

66.What are the main Implementations of the Map interface ?

The main implementations of the List interface are as follows:

- HashMap
- HashTable
- TreeMap
- EnumMap

67.What is a TreeMap ?

TreeMap actually implements the SortedMap interface which extends the Map interface. In a TreeMap the data will be sorted in ascending order of keys according to the natural order for the key's class, or by the comparator provided at creation time. TreeMap is based on the Red-Black tree data structure.

68. How do you decide when to use HashMap and when to use TreeMap ?

For inserting, deleting, and locating elements in a Map, the HashMap offers the best alternative. If, however, you need to traverse the keys in a sorted order, then TreeMap is your better alternative. Depending upon the size of your collection, it may be faster to add elements to a HashMap, then convert the map to a TreeMap for sorted key traversal.

69. Difference between HashMap and Hashtable ?

HashMap	Hashtable
HashMap lets you have null values as well as one null key.	HashTable does not allows null values as key and value.
The iterator in the HashMap is fail-safe (If you change the map while iterating, you'll know).	The enumerator for the Hashtable is not fail-safe.
HashMap is unsynchronized.	Hashtable is synchronized.

Note: Only one NULL is allowed as a key in HashMap. HashMap does not allow multiple keys to be NULL. Nevertheless, it can have multiple NULL values.

70. How does a Hashtable internally maintain the key-value pairs?

TreeMap actually implements the SortedMap interface which extends the Map interface. In a TreeMap the data will be sorted in ascending order of keys according to the natural order for the key's class, or by the comparator provided at creation time. TreeMap is based on the Red-Black tree data structure.

71. What Are the different Collection Views That Maps Provide?

Maps Provide Three Collection Views.

- **Key Set** - allow a map's contents to be viewed as a set of keys.
- **Values Collection** - allow a map's contents to be viewed as a set of values.
- **Entry Set** - allow a map's contents to be viewed as a set of key-value mappings.

72. What is a KeySet View ?

KeySet is a set returned by the ***keySet()*** method of the Map Interface, It is a set that contains all the keys present in the Map.

73. What is a Values Collection View ?

Values Collection View is a collection returned by the ***values()*** method of the Map Interface, It contains all the objects present as values in the map.

74. What is an EntrySet View ?

Entry Set view is a set that is returned by the ***entrySet()*** method in the map and contains Objects of type Map. Entry each of which has both Key and Value.

75.How do you sort an ArrayList (or any list) of user-defined objects ?

Create an implementation of the *java.lang.Comparable* interface that knows how to order your objects and pass it to *java.util.Collections.sort*(List, Comparator).

76.What is the Comparable interface ?

The Comparable interface is used to sort collections and arrays of objects using the `Collections.sort()` and `java.utils.Arrays.sort()` methods respectively. The objects of the class implementing the Comparable interface can be ordered.

The Comparable interface in the generic form is written as follows:

```
interface Comparable<T>
```

where T is the name of the type parameter.

All classes implementing the Comparable interface must implement the `compareTo()` method that has the return type as an integer. The signature of the`compareTo()` method is as follows:

```
int i = object1.compareTo(object2)
```

- If object1 < object2: The value of i returned will be negative.
- If object1 > object2: The value of i returned will be positive.
- If object1 = object2: The value of i returned will be zero.

77.What are the differences between the Comparable and Comparator interfaces ?

Comparable	Comparato
It uses the *compareTo()* method. *int objectOne.compareTo(objectTwo).*	t uses the *compare()* method. *int compare(ObjOne, ObjTwo)*
It is necessary to modify the class whose instance is going to be sorted.	A separate class can be created in order to sort the instances.
Only one sort sequence can be created.	Many sort sequences can be created.
It is frequently used by the API classes.	It used by third-party classes to sort instances.

78. Which two method you need to implement for key Object in HashMap ?

In order to use any object as Key in HashMap, it must implements equals and hashcode method in Java.

79. What is immutable object? Can you write immutable object?

Immutable classes are Java classes whose objects cannot be modified once created. Any modification in Immutable objects result in new object. For example is String is immutable in Java. Mostly Immutable are also final in Java, in order to prevent sub class from overriding methods in Java which

can compromise Immutability. You can achieve same functionality by making member as non-final but private and not modifying them except in constructor.

80. What is the difference between creating String as new() and literal?

When we create string with new() Operator, it's created in heap and not added into string pool while String created using literal are created in String pool itself which exists in PermGen area of heap.

String s = new String("Test");

does not put the object in String pool , we need to call String.intern() method which is used to put them into String pool explicitly. its only when you create String object as String literal e.g. String s = "Test" Java automatically put that into String pool.

81. What is difference between StringBuffer and StringBuilder in Java ?

Classic Java questions which some people thing tricky and some consider very easy. StringBuilder in Java is introduced in Java 5 and only difference between both of them is that Stringbuffer methods are synchronized while StringBuilder is non synchronized

82. Write code to find the First non repeated character in the String ?

A non repeated character occurs only once in the string , so if we store the number of times each alphabet appears in the string , it would help us identifying which characters are non repeated characters in the string .So we need to scan the whole string and determine the final counts of each character . Now scan the final values of each character in the string , the first character in the string with final count 1 is the first non repeated character in the string.

83. What is the difference between ArrayList and Vector ?

This question is mostly used as a start up question in Technical interviews on the topic of Collection framework . Answer is explained in detail here Difference between ArrayList and Vector .

84. How do you handle error condition while writing stored procedure or accessing stored procedure from java?

stored procedure should return error code if some operation fails but if stored procedure itself fail than catching SQLException is only choice.

85. What is difference between Executor.submit() and Executer.execute() method ?

There is a difference when looking at exception handling. If your tasks throws an exception and if it was submitted with execute this exception will go to the uncaught exception handler (when you don't have provided one explicitly, the default one will just print the stack trace to System.err). If you submitted the task with submit any thrown exception, checked exception or not, is then part of the task's return status. For a task that was submitted with submit and that terminates with an exception, the Future.get will re-throw this exception, wrapped in an Execution Exception.

86. What is the difference between factory and abstract factory pattern?

Abstract Factory provides one more level of abstraction. Consider different factories each extended from an Abstract Factory and responsible for creation of different hierarchies of objects based on the

type of factory. E.g. AbstractFactory extended by AutomobileFactory, UserFactory, RoleFactory etc. Each individual factory would be responsible for creation of objects in that genre.

87. What is Singleton? is it better to make whole method synchronized or only critical section synchronized ?
Singleton in Java is a class with just one instance in whole Java application, for example java.lang.Runtime is a Singleton class. Creating Singleton was tricky prior Java 4 but once Java 5 introduced Enum its very easy.

88. Can you write critical section code for singleton?
This core Java question is followup of previous question and expecting candidate to write Java singleton using double checked locking. Remember to use volatile variable to make Singleton thread-safe.

89. Can you write code for iterating over hashmap in Java 4 and Java 5 ?
Tricky one but he managed to write using while and for loop.

90. When do you override hashcode and equals() ?
Whenever necessary especially if you want to do equality check or want to use your object as key in HashMap.

91. What will be the problem if you don't override hashcode() method ?
You will not be able to recover your object from hash Map if that is used as key in HashMap.

92. Is it better to synchronize critical section of getInstance() method or whole getInstance() method ?
Answer is critical section because if we lock whole method than every time some one call this method will have to wait even though we are not creating any object)

93. What is the difference when String is gets created using literal or new() operator ?
When we create string with new() its created in heap and not added into string pool while String created using literal are created in String pool itself which exists in Perm area of heap.

94. Does not overriding hashcode() method has any performance implication ?
This is a good question and open to all , as per my knowledge a poor hashcode function will result in frequent collision in HashMap which eventually increase time for adding an object into Hash Map.

95. What's wrong using HashMap in multithreaded environment? When get() method go to infinite loop ?
Answer was during concurrent access and re-sizing.

96. What do you understand by thread-safety ? Why is it required? And finally, how to achieve thread-safety in Java Applications?
Java Memory Model defines the legal interaction of threads with the memory in a real computer system. In a way, it describes what behaviors are legal in multi-threaded code. It determines when a Thread can reliably see writes to variables made by other threads. It defines semantics for volatile, final & synchronized, that makes guarantee of visibility of memory operations across the Threads.

Let's first discuss about Memory Barrier which are the base for our further discussions. There are two

type of memory barrier instructions in JMM - read barriers and write barrier.

A read barrier invalidates the local memory (cache, registers, etc) and then reads the contents from the main memory, so that changes made by other threads become visible to the current Thread.
A write barrier flushes out the contents of the processor's local memory to the main memory, so that changes made by the current Thread become visible to the other threads.
JMM semantics for synchronized
When a thread acquires monitor of an object, by entering into a synchronized block of code, it performs a read barrier (invalidates the local memory and reads from the heap instead). Similarly exiting from a synchronized block as part of releasing the associated monitor, it performs a write barrier (flushes changes to the main memory)
Thus modifications to a shared state using synchronized block by one Thread, is guaranteed to be visible to subsequent synchronized reads by other threads. This guarantee is provided by JMM in presence of synchronized code block.

JMM semantics for Volatile fields
Read & write to volatile variables have same memory semantics as that of acquiring and releasing a monitor using synchronized code block. So the visibility of volatile field is guaranteed by the JMM. Moreover afterwards Java 1.5, volatile reads and writes are not reorderable with any other memory operations (volatile and non-volatile both). Thus when Thread A writes to a volatile variable V, and afterwards Thread B reads from variable V, any variable values that were visible to A at the time V was written are guaranteed now to be visible to B.

Let's try to understand the same using the following code

```
Data data = null;
volatile boolean flag = false;


Thread A
-------------
data = new Data();
flag = true;  <-- writing to volatile will flush data as well as flag to main memory


Thread B
------------
if(flag==true){
<-- as="" barrier="" data.="" flag="" font="" for="" from="" perform="" read="" reading="" volatile="" well="" will="">  →
use data;
 <!--- data is guaranteed to visible even though it is not declared volatile because of the JMM semantics of volatile flag.  →
}
```

97. What will happen if you call return statement or System.exit on try or catch block ? will finally block execute?

This is a very popular tricky Java question and its tricky because many programmer think that finally block always executed. This question challenge that concept by putting return statement in try or catch block or calling System.exit from try or catch block. Answer of this tricky question in Java is that finally block will execute even if you put return statement in try block or catch block but finally block **won't run if you call System.exit** form try or catch.

98. **Can you override private or static method in Java ?**
Another popular Java tricky question, As I said method overriding is a good topic to ask trick questions in Java. Anyway, you cannot override private or static method in Java, if you create similar method with same return type and same method arguments that's called method hiding.

99. **What will happen if we put a key object in a HashMap which is already there ?**
This tricky Java questions is part of How HashMap works in Java, which is also a popular topic to create confusing and tricky question in Java. Well if you put the same key again than it will replace the old mapping because HashMap doesn't allow duplicate keys.

100. **If a method throws NullPointerException in super class, can we override it with a method which throws RuntimeException?**
One more tricky Java questions from overloading and overriding concept. Answer is you can very well throw super class of RuntimeException in overridden method but you can not do same if its checked Exception.

101. **What is the issue with following implementation of compareTo() method in Java**

```
public int compareTo(Object o){
  Employee emp = (Employee) emp;
  return this.id - o.id;
}
```

102. **How do you ensure that N thread can access N resources without deadlock**
If you are not well versed in writing multi-threading code then this is real tricky question for you. This Java question can be tricky even for experienced and senior programmer, who are not really exposed to deadlock and race conditions. Key point here is order, if you acquire resources in a particular order and release resources in reverse order you can prevent deadlock.

103. **What is difference between CyclicBarrier and CountDownLatch in Java**
Relatively newer Java tricky question, only been introduced form Java 5. Main difference between both of them is that you can reuse CyclicBarrier even if Barrier is broken but you can not reuse CountDownLatch in Java. See CyclicBarrier vs CountDownLatch in Java for more differences.

104. **Can you access non static variable in static context?**
Another tricky Java question from Java fundamentals. No you can not access static variable in non static context in Java. Read why you can not access non-static variable from static method to learn more about this tricky Java questions.

Other Topic..