2016

Core Java Material

NOTES

SAGAR CHEKE

Object Oriented Concepts:-

1. **What are the principle concepts of OOPS?**

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

   a) Abstraction
   b) Polymorphism
   c) Inheritance
   d) Encapsulation

   (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?**

   a) **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.
   b) **Abstraction** solves the problem in the design side while **Encapsulation** is the Implementation.
   c) **Encapsulation** is the deliverables of Abstraction. Encapsulation barely talks about grouping up your abstraction to suit the developer needs.

5. **What is Inheritance?**

   a) Inheritance is the process by which objects of one class acquire the properties of objects of another class.
   b) A class that is inherited is called a superclass.
   c) The class that does the inheriting is called a subclass.
   d) Inheritance is done by using the keyword extends.
   e) 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.

   a) In some cases, multiple methods have the same name, but different formal argument lists (overloaded methods).
   b) 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:*
a) Method overloading
b) Method overriding through inheritance
c) Method overriding through the Java interface

9. What is runtime polymorphism or dynamic method dispatch?

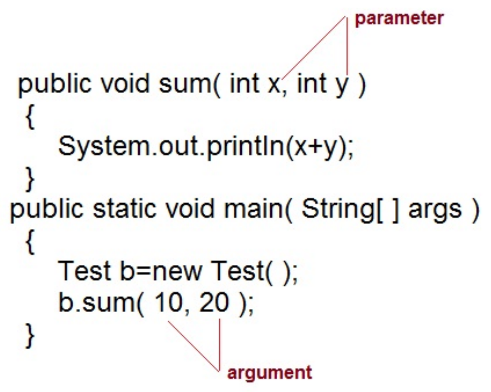
⇨ In Java, runtime polymorphism or dynamic method dispatch is a process in which a call to an overridden method is resolved at runtime rather than at compile-time. In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called 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. Parameter Vs. Argument

⇨ While talking about method, it is important to know the difference between two terms parameter and argument. Parameter is variable defined by a method that receives value when the method is called. Parameter are always local to the method they don't have scope outside the method. While argument is a value that is passed to a method when it is called.



```
 public void sum( int x, int y )
 {
    System.out.println(x+y);
 }
public static void main( String[ ] args )
 {
    Test b=new Test( );
    b.sum( 10, 20 );
 }
```

There are two ways to pass an argument to a method
a) **Call-by-value:** In this approach copy of an argument value is pass to a method. Changes made to the argument value inside the method will have no effect on the arguments.
b) **Call-by-reference:** In this reference of an argument is pass to a method. Any changes made inside the method will affect the argument value.

NOTE: In Java, when you pass a primitive type to a method it is passed by value whereas when you pass an object of any type to a method it is passed as reference.

12. 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.
Method overloading is one of the ways through which java supports polymorphism. Method overloading can be done **by changing number of arguments or by changing the data type of arguments**.

Different ways of Method overloading:

a) Method overloading by changing data type of Arguments
b) Method overloading by changing no. of argument.

Note:

a) Overloaded methods MUST change the argument list
b) Overloaded methods CAN change the return type
c) Overloaded methods CAN change the access modifier
d) Overloaded methods CAN declare new or broader checked exceptions
e) A method can be overloaded in the same class or in a subclass

13. 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:

a) 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).
b) You cannot override a method marked final
c) You cannot override a method marked static

14. **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 (based on declared argument types) is selected. Happens at compile time. The actual method that's invoked is still a virtual method invocation that happens at runtime, but the compiler will already know the signature of the method to be invoked. So at runtime, the argument match will already have been nailed down, just not the class in which the method lives.	Object type (in other words, the type of the actual instance on the heap) determines which method is selected. Happens at runtime.

15. **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.

16. **Is it possible to override the main method?**
⇨ NO, because main is a static method. A static method can't be overridden in Java.

17. 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();
```

18. 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:
a) You can only go back one level.
b) 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.

19. **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
 }
```

20. What is an Interface?
⇨ An interface is a description of a set of methods that conforming implementing classes must have.
Note:
a) You can't mark an interface as final.
b) Interface variables must be static.
c) An Interface cannot extend anything but another interfaces.

21. Can we instantiate an interface?
⇨ You can't instantiate an interface directly, but you can instantiate a class that implements an interface.

22. 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.

23. 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.

24. **What modifiers are allowed for methods in an Interface?**
⇨ Only public and abstract modifiers are allowed for methods in interfaces.

25. **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.

26. 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:

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

27. Can we instantiate an abstract class?

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

28. 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.

29. **When should I use abstract classes and when should I use interfaces?**

Use Interfaces when…

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

a) If various implementations are of the same kind and use common behavior or status then abstract class is better to use.

b) When you want to provide a generalized form of abstraction and leave the implementation task with the inheriting subclass.
c) Abstract classes are an excellent way to create planned inheritance hierarchies. They're also a good choice for nonleaf classes in class hierarchies.

30. **When you declare a method as abstract, can other no abstract methods access it?**
⇨ Yes, other no abstract methods can access a method that you declare as abstract.

31. **Can there be an abstract class with no abstract methods in it?**
⇨ Yes, there can be an abstract class without abstract methods.

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

There are two types of Constructor
a) Default Constructor
b) Parameterized constructor

```
Car c = new Car()       //Default constructor invoked
Car c = new Car(name); //Parameterized constructor invoked
```

33. **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.

34. **Can constructor be inherited**?
⇨ No, constructor cannot be inherited, though a derived class can call the base class constructor.

35. Constructor Overloading
⇨ Constructor overloading is done to construct object in different ways.
Like methods, a constructor can also be overloaded. Overloaded constructors are differentiated on the basis of their type of parameters or number of parameters. Constructor overloading is not much different than method overloading. In case of method overloading you have multiple methods with same name but different signature, whereas in Constructor overloading you have multiple constructor with different signature but only difference is that Constructor doesn't have return type in Java.

36. **What are the differences between Constructor's 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
	Constructors can only called once	Methods could be called many times and it can return a value or void.

37. *this* keyword

a) *this* keyword is used to refer to current object.
b) *this* is always a reference to the object on which method was invoked.
c) *this* can be used to invoke current class constructor.
d) *this* can be passed as an argument to another method.
e) The *this* is used to call overloaded constructor in java
f) The *this* is also used to call Method of that class.
g) The *this* is used to return current Object

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

a) Constructors use *this* to refer to another constructor in the same class with a different parameter list.
b) 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.

39. 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.

40. 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*.

41. **What are Access Specifiers available in Java**?

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

a) **Public**- *public* classes, methods, and fields can be accessed from everywhere.
b) **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.
c) **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.
d) **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

42. **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.

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

43. What are the uses of final method?

⇨ There are two reasons for marking a method as final:

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

44. **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.

45. **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.

46. **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.

47. **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.

<u>Note</u>: The use of a static method suffers from the following restrictions:

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

48. **Difference between Java 6 and Java 7**

Java 6	**Java 7**
Support for older win9x versions dropped.	Upgrade class-loader architecture: A method that frees the underlying resources, such as open files, held by a URLClassLoader
Scripting lang support: Generic API for integration with scripting languages, & built-in mozilla javascript rhino integration	Concurrency and collections updates: A lightweight fork/join framework, flexible and reusable synchronization barriers, transfer queues, concurrent linked double-ended queues, and thread-local pseudo-random number generators.
Dramatic performance improvements for the core platform, and swing.	Internationalization Upgrade: Upgrade on Unicode 6.0, Locale enhancement and Separate user locale and user-interface locale.
Improved web service support through JAX-WS JDBC 4.0 support	More new I/O APIs for the Java platform (NIO.2), NIO.2 file system provider for zip/jar archives, SCTP, SDP, TLS 1.2 support.
Java compiler API: an API allowing a java program to select and invoke a java compiler programmatically.	Security & Cryptography implemented Elliptic-curve cryptography (ECC)
Upgrade of JAXB to version 2.0: including integration of a stax parser	Upgrade to JDBC 4.1 and Rowset 1.1.
Support for pluggable annotations	XRender pipeline for Java 2D, Create new platform APIs for 6u10 graphics features, Nimbus look-and-feel for Swing, Swing JLayer component, Gervill sound synthesizer.
Many GUI improvements, such as integration of swing worker in the API, table sorting and filtering, and true swing double-buffering (eliminating the gray-area effect).	Upgrade the components of the XML stack to the most recent stable versions: JAXP 1.4, JAXB 2.2a, and JAX-WS 2.2.
	Enhanced MBeans." Support for dynamically-typed languages (InvokeDynamic): Extensions to the JVM, the Java language, and the Java SE API to support the implementation of dynamically-typed languages at

	performance levels near to that of the Java language itself
	Strict class-file checking: Class files of version 51 (SE 7) or later must be verified with the typechecking verifier; the VM must not fail over to the old inferencing verifier.
	Small language enhancements (Project Coin): A set of small language changes intended to simplify common, day-to-day programming tasks: Strings in switch statements, try-with-resources statements, improved type inference for generic instance creation ("diamond"), simplified var args method invocation, better integral literals, and improved exception handling (multi-catch).

49. **Difference between Java 8 and Java 9**

Java 8	Java 9
JSR 335, JEP 126: Language-level support for lambda expressions.	A lightweight JSON API for consuming and generating JSON documents and data streams.
JSR 223, JEP 174: Project Nashorn, a JavaScript runtime which allows developers to embed JavaScript code within applications.	A HTTP 2 Client that will bring HTTP 2.0 and web sockets, while replacing the legacy HttpURLConnection.
JSR 308, JEP 104: Annotation on Java Types.	Process API Updates to improve controlling and managing operating-system process (developers were often forced to use native code with the current API). Along with several other smaller features, as well as dozens of proposals already being tracked by the JEP Index, Oracle has also promised another trio of performance features
Unsigned Integer Arithmetic.	Improve contended locking, which aims at improving performance when threads compete over access to objects.
JSR 337, JEP 120: Repeating annotations.	Segmented code cache with better performance, shorter sweep times, less fragmentation and further extensions to come.
JSR 310, JEP 150: Date and Time API.	The Smart Java compiler, or sjavac, will be improved to allow default use in the JDK build and general use for building larger projects.
JEP 178: Statically-linked JNI libraries.	
JEP 153: Launch JavaFX applications (direct launching of JavaFX application JARs).	
JEP 122: Remove the permanent generation.	
Java 8 is not supported on Windows XP. But as of JDK 8 update 5, it still can run under Windows XP after forced	

installation by directly unzipping from the installation executable.	

⍰ *Java Collections:-*

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

2. **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:
   a) Obtain an iterator to the start of the collection by calling the collections ***iterator()*** method.
   b) Set up a loop that makes a call to ***hasNext()***. Have the loop iterate as long as ***hasNext()*** returns **true**.
   c) Within the loop, obtain each element by calling **next()**.

3. **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.

4. **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.

5. **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

6. **What is the List interface**?
   a) The List interface provides support for ordered collections of objects.
   b) Lists may contain duplicate elements.

7. **What are the main implementations of the List interface**?
⇨ The main implementations of the List interface are as follows:
   a) **ArrayList**: Resizable-array implementation of the List interface. The best all-around implementation of the List interface.
   b) **Vector**: Synchronized resizable-array implementation of the List interface with additional "legacy methods."
   c) **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).

8. **What are the advantages of ArrayList over arrays?**

⇨ Some of the advantages ArrayList has over arrays are:

   a) It can grow dynamically
   b) It provides more powerful insertion and search mechanisms than arrays.

9. **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.

10. **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();
```

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

   a) ArrayList internally uses an 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.
   b) 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.

12. **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.

13. **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.

14. **What is the Set interface?**
    a) The Set interface provides methods for accessing the elements of a finite mathematical set
    b) Sets do not allow duplicate elements
    c) Contains no methods other than those inherited from Collection
    d) It adds the restriction that duplicate elements are prohibited
    e) Two Set objects are equal if they contain the same elements

15. **What are the main Implementations of the Set interface**?
- ⇨ The main implementations of the Set interface are as follows:
    a) HashSet
    b) TreeSet
    c) LinkedHashSet
    d) EnumSet

16. **What is a HashSet**?
    a) A HashSet is an unsorted, unordered Set.
    b) It uses the hashcode of the object being inserted (so the more efficient your hashcode() implementation the better access performance you'll get).
    c) Use this class when you want a collection with no duplicates and you don't care about order when you iterate through it.

17. **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.

18. **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.

19. **Difference between HashSet and TreeSet**?

HashSet	**TreeSet**
It does not guarantee for either sorted order or sequence order.	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.

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

21. **What are the main Implementations of the Map interface**?
- ⇨ The main implementations of the Map interface are as follows:

a) HashMap
b) HashTable
c) TreeMap
d) EnumMap

22. **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.

23. **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.

24. **Difference between HashMap and Hashtable**?

HashMap	**Hashtable**
1) HashMap is **non synchronized**. It is not-thread safe and can't be shared between many threads without proper synchronization code.	Hashtable is **synchronized**. It is thread-safe and can be shared with many threads.
2) HashMap **allows one null key and multiple null values**.	Hashtable **doesn't allow any null key or value**.
3) HashMap is a **new class introduced in JDK 1.2**.	Hashtable is a **legacy class**.
4) HashMap is **fast**.	Hashtable is **slow**.
5) We can make the HashMap as synchronized by calling this code Map m = Collections.synchronizedMap(hashMap);	Hashtable is internally synchronized and can't be unsynchronized.
6) HashMap is **traversed by Iterator**.	Hashtable is **traversed by Enumerator and Iterator**.
7) Iterator in HashMap is **fail-fast**.	Enumerator in Hashtable is **not fail-fast**.
8) HashMap inherits **AbstractMap** class.	Hashtable inherits **Dictionary** class.

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.

25. **How does a TreeMap 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.

26. **What are the different Collection Views That Maps Provide?**

⇨ Maps provide three collection views.

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

27. **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.

28. **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.

29. **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.

30. **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).

31. **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)

a) If object1 < object2: The value of i returned will be negative.
b) If object1 > object2: The value of i returned will be positive.
c) If object1 = object2: The value of i returned will be zero.

32. **What are the differences between the Comparable and Comparator interfaces**?

Comparable	Comparator
It uses the *compareTo()* method. *int objectOne.compareTo(objectTwo).*	It 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.

33. Difference between List and Set in Java Collection?

a. Fundamental difference between List and Set in Java is allowing duplicate elements. List in Java allows duplicates while Set doesn't allow any duplicate. If you insert duplicate in Set it will replace the older value. Any implementation of Set in Java will only contains unique elements.
b. Another significant difference between List and Set in Java is order. List is an Ordered Collection while Set is an unordered Collection. List maintains insertion order of elements, means any element which is inserted before will go on lower index than any element which is inserted after. Set in Java doesn't maintain any order. Though Set provide another alternative called SortedSet which can store Set elements in specific Sorting order defined by Comparable and Comparator methods of Objects stored in Set.

c. Set uses equals () method to check uniqueness of elements stored in Set, while SortedSet uses compareTo() method to implement natural sorting order of elements. In order for an element to behave properly in Set and SortedSet, equals and compareTo must be consistent to each other.
d. Popular implementation of List interface in Java includes ArrayList, Vector and LinkedList. While popular implementation of Set interface includes HashSet, TreeSet and LinkedHashSet.

34. What is the difference between poll() and remove() method of Queue interface?

Though both poll() and remove() method from Queue is used to remove the object and returns the head of the queue, there is a subtle difference between them. If Queue is empty() then a call to remove() method will throw Exception, while a call to poll() method returns null. By the way, exactly which element is removed from the queue depends upon queue's ordering policy and varies between different implementation, for example, PriorityQueue keeps the lowest element as per Comparator or Comparable at head position.

35. What is fail safe and fail fast Iterator in Java?

Java Collections supports two types of Iterator, fail safe and fail fast. The main distinction between a fail-fast and fail-safe Iterator is whether or not the underlying collection can be modified while it's begin iterated. If you have used Collection like ArrayList then you know that when you iterate over them, no other thread should modify the collection. If Iterator detects any structural change after iteration has begun e.g adding or removing a new element then it throws ConcurrentModificationException, this is known as fail-fast behavior and these iterators are called fail-fast iterator because they fail as soon as they detect any modification . Though it's not necessary that iterator will throw this exception when multiple threads modified it simultaneously. it can happen even with the single thread when you try to remove elements by using ArrayList's remove() method instead of Iterator's remove method.

Most of the Collection classes from Java 1.4 e.g. Vector, ArrayList, HashMap, HashSet has fail-fast iterators. The other type of iterator was introduced in Java 1.5 when a concurrent collection class e.g. ConcurrentHashMap, CopyOnWriteArrayList and CopyOnWriteArraySet was introduced. This iterator uses a view of original collection for doing iteration and that's why they don't throw ConcurrentModificationException even when original collection was modified after iteration has begun. This means you could iterate and work with stale value, but this is the cost you need to pay for fail-safe iterator

36. Difference between Fail Safe and Fail Fast Iterator in Java.

a) Fail-fast Iterator throws ConcurrentModfiicationException as soon as they detect any structural change in collection during iteration, basically which changes the modCount variable hold by Iterator. While fail-fast iterator doesn't throw CME.
b) Fail-fast iterator traverse over original collection class while fail-safe iterator traverse over a copy or view of original collection. That's why they don't detect any change on original collection classes and this also means that you could operate with stale value.
c) Iterators from Java 1.4 Collection classes e.g. ArrayList, HashSet and Vector are fail-fast while Iterators returned by concurrent collection classes e.g. CopyOnWriteArrayList or CopyOnWriteArraySet are fail-safe.

d) Iterator returned by synchronized Collection are fail-fast while iterator returned by concurrent collections are fail-safe in Java.
e) Fail fast iterator works in live data but become invalid when data is modified while fail-safe iterator are weekly consistent.

<u>When to use fail fast and fail-safe Iterator</u>

Use fail-safe iterator when you are not bothered about Collection to be modified during iteration, as fail-fast iterator will not allow that. Unfortunate you can't choose fail safe or fail-fast iterator, it depends on upon which Collection class you are using. Most of the JDK 1.4 Collections e.g. HashSet, Vector, ArrayList has fail-fast Iterator and only Concurrent Collections introduced in JDK 1.5 e.g. CopyOnWriteArrayList and CopyOnWriteArraySet supports fail safe Iteration. Also, if you want to remove elements during iteration please use iterator's remove() method and don't use remove method provided by Collection classes e.g. ArrayList or HashSet because that will result in ConcurrentModificationException.

37. **How do you remove an entry from a Collection? and subsequently what is the difference between the remove() method of Collection and remove() method of Iterator, which one you will use while removing elements during iteration?**

Collection interface defines remove(Object obj) method to remove objects from Collection. List interface adds another method remove(int index), which is used to remove object at specific index. You can use any of these method to remove an entry from Collection, while not iterating. Things change, when you iterate. Suppose you are traversing a List and removing only certain elements based on logic, then you need to use Iterator's remove() method. This method removes current element from Iterator's perspective. If you use Collection's or List's remove() method during iteration then your code will throw ConcurrentModificationException. That's why it's advised to use Iterator remove() method to remove objects from Collection.

38. Difference between ConcurrentHashMap, Hashtable and Synchronized Map in Java

Though all three collection classes are thread-safe and can be used in multi-threaded, concurrent Java application, there is a significant difference between them, which arise from the fact that how they achieve their thread-safety. Hashtable is a legacy class from JDK 1.1 itself, which uses synchronized methods to achieve thread-safety. All methods of Hashtable are synchronized which makes them quite slow due to contention if a number of thread increases. Synchronized Map is also not very different than Hashtable and provides similar performance in concurrent Java programs. The only difference between Hashtable and Synchronized Map is that later is not a legacy and you can wrap any Map to create it's synchronized version by using Collections.synchronizedMap() method.

On the other hand, ConcurrentHashMap is specially designed for concurrent use i.e. more than one thread. By default it simultaneously allows 16 threads to read and write from Map without any external synchronization. It is also very scalable because of stripped locking technique used in the internal implementation of ConcurrentHashMap class. Unlike Hashtable and Synchronized Map, it never locks whole Map, instead, it divides the map into segments and locking is done on those. Though it performs better if a number of reader threads are greater than the number of writer threads.

39. Why need ConcurrentHashMap and CopyOnWriteArrayList

The synchronized collections classes, Hashtable, and Vector, and the synchronized wrapper classes, Collections.synchronizedMap() and Collections.synchronizedList(), provide a basic conditionally thread-safe implementation of Map and List. However, several factors make them unsuitable for use in highly concurrent applications, for example, their single collection-wide lock is an impediment to scalability and it often becomes necessary to lock a collection for a considerable time during iteration to prevent ConcurrentModificationException. ConcurrentHashMap and CopyOnWriteArrayList implementations provide much higher concurrency while preserving thread safety, with some minor compromises in their promises to callers. ConcurrentHashMap and CopyOnWriteArrayList are not necessarily useful everywhere you might use HashMap or ArrayList, but are designed to optimize specific common situations. Many concurrent applications will benefit from their use.

40. Difference between ConcurrentHashMap and Hashtable

Both can be used in the multithreaded environment but once the size of Hashtable becomes considerable large performance degrade because for iteration it has to be locked for a longer duration.
Since ConcurrentHashMap introduced the concept of segmentation, how large it becomes only certain part of it get locked to provide thread safety so many other readers can still access map without waiting for iteration to complete.
In Summary, ConcurrentHashMap only locked certain portion of Map while Hashtable locks full map while doing iteration.

41. The difference between ConcurrentHashMap and Collections.synchronizedMap

ConcurrentHashMap is designed for concurrency and improve performance while HashMap which is non-synchronized by nature can be synchronized by applying a wrapper using synchronized Map. Here are some of the common differences between ConcurrentHashMap and synchronized map in Java

ConcurrentHashMap does not allow null keys or null values while synchronized HashMap allows one null key.

Exception:-

1. What is an exception?

⇨ An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions.

2. What is error?

⇨ An Error indicates that a non-recoverable condition has occurred that should not be caught. Error, a subclass of Throwable, is intended for drastic problems, such as OutOfMemoryError, which would be reported by the JVM itself.

3. What is the difference between error and exception in java?

Errors are mainly caused by the environment in which an application is running. For example, OutOfMemoryError happens when JVM runs out of memory.
Whereas exceptions are mainly caused by the application itself. For example, NullPointerException occurs when an application tries to access null object.

4. Which is superclass of Exception?

⇨ **"Throwable"**, the parent class of all exception related classes.

5. What are the advantages of using exception handling?

⇨ Exception handling provides the following advantages over "traditional" error management techniques:
a) Separating Error Handling Code from "Regular" Code.
b) Propagating Errors up the Call Stack.
c) Grouping Error Types and Error Differentiation.

6. What are the types of Exceptions in Java?

⇨ There are two types of exceptions in Java, unchecked exceptions and checked exceptions.

a) **Checked exceptions:** A checked exception is some subclass of Exception (or Exception itself), excluding class RuntimeException and its subclasses. Each method must either handle all checked exceptions by supplying a catch clause or list each unhandled checked exception as a thrown exception.

b) **Unchecked exceptions:** All Exceptions that extend the RuntimeException class are unchecked exceptions. Class Error and its subclasses also are unchecked.

7. Why Errors are Not Checked?

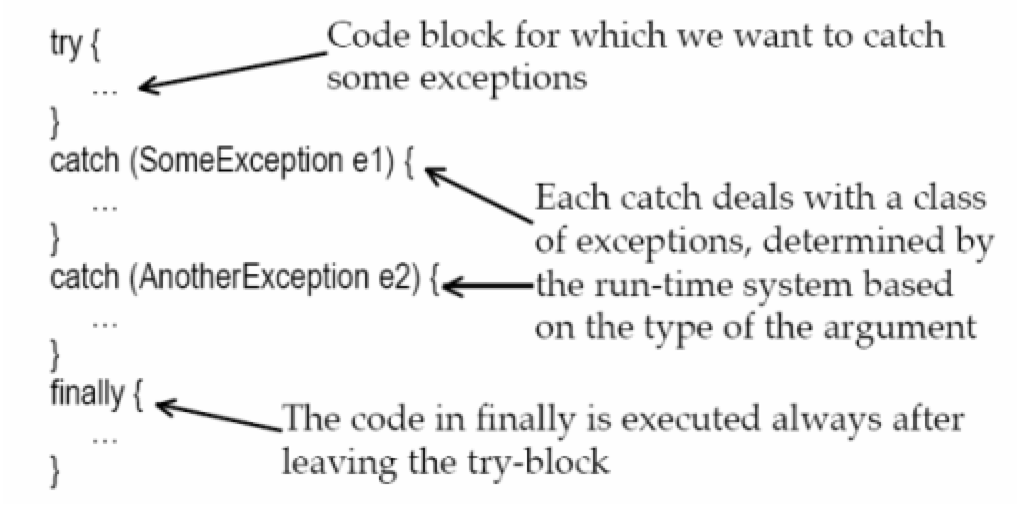
⇨ An unchecked exception classes which are the error classes (Error and its subclasses) are exempted from compile-time checking because they can occur at many points in the program and recovery from them is difficult or impossible. A program declaring such exceptions would be pointlessly.

8. Why Runtime Exceptions are Not Checked?

⇨ The runtime exception classes (RuntimeException and its subclasses) are exempted from compile-time checking because, in the judgment of the designers of the Java programming language, having to declare such exceptions would not aid significantly in establishing the correctness of programs. Many of the operations and constructs of the Java programming language can result in runtime exceptions. The information available to a compiler, and the level of analysis the compiler performs, are usually not sufficient to establish that such run-time exceptions cannot occur, even though this may be obvious to the programmer. Requiring such exception classes to be declared would simply be an irritation to programmers.

9. Explain the significance of try-catch blocks?

⇨ Whenever the exception occurs in Java, we need a way to tell the JVM what code to execute. To do this, we use the try and catch keywords. The try is used to define a block of code in which exceptions may occur. One or more catch clauses match a specific exception to a block of code that handles it.



```
try {
    ...
}
catch (SomeException e1) {
    ...
}
catch (AnotherException e2) {
    ...
}
finally {
    ...
}
```

Code block for which we want to catch some exceptions
Each catch deals with a class of exceptions, determined by the run-time system based on the type of the argument
The code in finally is executed always after leaving the try-block

10. **What is the use of finally block?**

⇨ The finally block encloses code that is always executed at some point after the try block, whether an exception was thrown or not. This is right place to close files, release your network sockets, connections, and perform any other cleanup your code requires.

Note: If the try block executes with no exceptions, the finally block is executed immediately after the try block completes. It there was an exception thrown, the finally block executes immediately after the proper catch block completes

11. **What if there is a break or return statement in try block followed by finally block**?

⇨ If there is a return statement in the try block, the finally block executes right after the return statement encountered, and before the return executes.

12. **Can we have the try block without catch block?**

⇨ Yes, we can have the try block without catch block, but finally block should follow the try block.
Note: It is not valid to use a try clause without either a catch clause or a finally clause.

13. **What is the difference throw and throws?**

⇨ **Throws:** Used in a method's signature if a method is capable of causing an exception that it does not handle, so that callers of the method can guard themselves against that exception. If a method is declared as throwing a particular class of exceptions, then any other method that calls it must either have a try-catch clause to handle that exception or must be declared to throw that exception (or its superclass) itself.

A method that does not handle an exception it throws has to announce this:

```
 public void myfunc(int arg) throws MyException {
      …
   }
```

Throw: Used to trigger an exception. The exception will be caught by the nearest try-catch clause that can catch that type of exception. The flow of execution stops immediately after the throw statement; any subsequent statements are not executed.

To throw an user-defined exception within a block, we use the throw command:

```
throw new MyException("I always wanted to throw an exception!");
```

14. **How to create custom exceptions?**

⇨ By extending the Exception class or one of its subclasses.

Example:

```
class MyException extends Exception {
  public MyException() { super(); }
  public MyException(String s) { super(s); }
  }
```

15. **What are the different ways to handle exceptions?**

⇨ There are two ways to handle exceptions:

a) Wrapping the desired code in a try block followed by a catch block to catch the exceptions.

b) List the desired exceptions in the throws clause of the method and let the caller of the method handle those exceptions.

16. **Difference between Checked and Unchecked Exceptions**

a) Unchecked Exception:-

The exceptions that are not checked at compile time are called unchecked exceptions, classes that extends RuntimeException comes under unchecked exceptions. Examples of some unchecked exceptions are listed below.

1. ArithmeticException

Mathematical operations that are not permitted in normal conditions i.e. dividing a number from '0'.

view plainprint?

```
package com.beingjavaguys.core;
 public class ExceptionTest {
```

```
            public static void main(String[] args) {
        int i = 10/0;
        }
}
```

Console:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
 at com.beingjavaguys.core.ExceptionTest.main(ExceptionTest.java:6)
```

2. ArrayIndexOutOfBoundsException

Trying to access an index that does not exists or inserting values to wrong indexes results to ArrayIndesOutOfBound Exception at runtime.
view plainprint?

```
package com.beingjavaguys.core
public class ExceptionTest {

public static void main(String[] args) {
        int arr[] = {'0','1','2'};
        System.out.println(arr[4]);
        }
}
```

Console :

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 4
 at com.beingjavaguys.core.ExceptionTest.main(ExceptionTest.java:7)
```

3. NullPointerException

Trying to access a null object or performing operations on an object having a null value.
view plainprint?

```
package com.beingjavaguys.core;
import java.util.ArrayList;

public class ExceptionTest {

        public static void main(String[] args) {
        String string = null;
        System.out.println(string.length());
        }
}
```

Console

```
Exception in thread "main" java.lang.NullPointerException
 at com.beingjavaguys.core.ExceptionTest.main(ExceptionTest.java:9)
```

Some common Unchecked Exceptions in Java:
Here is a list of some common Unchecked Exceptions in Java language. ArithmeticException ArrayStoreException, BufferOverflowException, BufferUnderflowException, CannotRedoException, CannotUndoException, ClassCastException, CMMException, ConcurrentModificationException, DOMException, EmptyStackException, IllegalArgumentException, IllegalMonitorStateException, IllegalPathStateException, IllegalStateException, ImagingOpException, IndexOutOfBoundsException, MissingResourceException, NegativeArraySizeException, NoSuchElementException, NullPointerException, ProfileDataException,

ProviderException, RasterFormatException, SecurityException, SystemException, UndeclaredThrowableException, UnmodifiableSetException, UnsupportedOperationException

b) CheckedExceptions:-

Exceptions that are checked at compile-time are called checked exceptions, in Exception hierarchy all classes that extends Exception class except UncheckedException comes under checked exception category.

In certain situations Java Compiler forced the programmer to write Exception handler at compile time, the Exceptions thrown in such situation are called Checked Exception, see the example below
plain print?

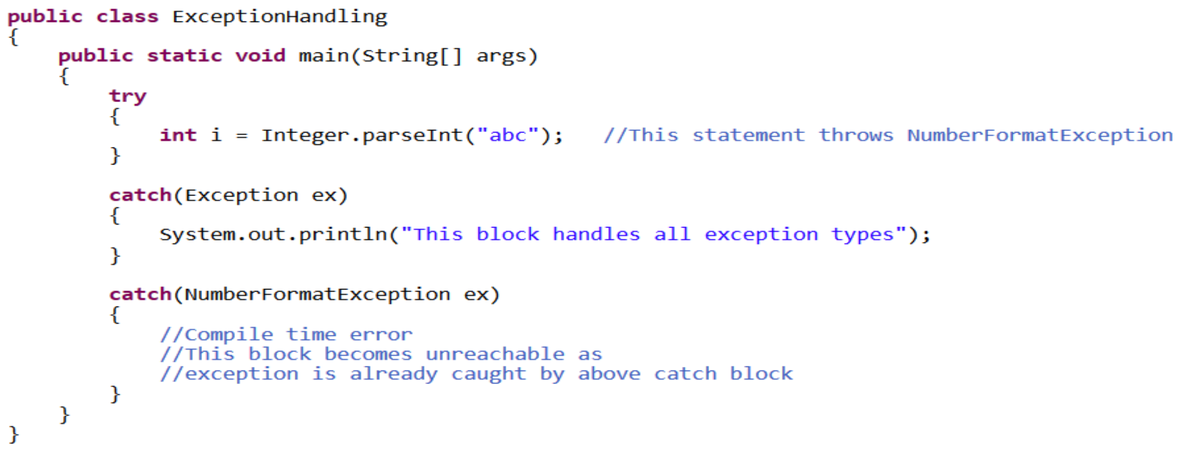
```
try {
    String input = reader.readLine();
    System.out.println("You typed : "+input); // Exception prone area
    } catch (IOException e) {
      e.printStackTrace();
}

```

While writing a code to read or write something from files or even from or to console, an checked Exception i.e. IOException is thrown, these exceptions are checked at compile time and we are forced to write a handler at compile time. That’s why we called these exceptions Checked Exceptions.

Some common CheckedExceptions in Java
Here is a list of some common Unchecked Exceptions in Java language. IOException
FileNotFoundException, ParseException, ClassNotFoundException, CloneNotSupportedException, InstantiationException, InterruptedException, NoSuchMethodException

17. There are three statements in a try block – statement1, statement2 and statement3. After that there is a catch block to catch the exceptions occurred in the try block. Assume that exception has occurred in statement2. Does statement3 get executed or not?
No. Once a try block throws an exception, remaining statements will not be executed. Control comes directly to catch block.

18. What is unreachable catch block error?
When you are keeping multiple catch blocks, the order of catch blocks must be from most specific to most general ones. i.e sub classes of Exception must come first and super classes later. If you keep super classes first and sub classes later, compiler will show unreachable catch block error.



```
public class ExceptionHandling
{
    public static void main(String[] args)
    {
        try
        {
            int i = Integer.parseInt("abc");   //This statement throws NumberFormatException
        }

        catch(Exception ex)
        {
            System.out.println("This block handles all exception types");
        }

        catch(NumberFormatException ex)
        {
            //Compile time error
            //This block becomes unreachable as
            //exception is already caught by above catch block
        }
    }
}
```

19. What is OutOfMemoryError in java?

OutOfMemoryError is the sub class of java.lang.Error which occurs when JVM runs out of memory.

20. What is the difference between ClassNotFoundException and NoClassDefFoundError in java?

ClassNotFoundException	NoClassDefFoundError
It is an exception. It is of type java.lang.Exception.	It is an error. It is of type java.lang.Error.
It occurs when an application tries to load a class at run time which is not updated in the classpath.	It occurs when java runtime system doesn't find a class definition, which is present at compile time, but missing at run time.
It is thrown by the application itself. It is thrown by the methods like Class.forName(), loadClass() and findSystemClass().	It is thrown by the Java Runtime System.
It occurs when classpath is not updated with required JAR files	It occurs when required class definition is missing at run time.

21. Give the list of Java Object class methods.

a) clone() - Creates and returns a copy of this object.
b) equals() - Indicates whether some other object is "equal to" this one.
c) finalize() - Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
d) getClass() - Returns the runtime class of an object.
e) hashCode() - Returns a hash code value for the object.
f) notify() - Wakes up a single thread that is waiting on this object's monitor.
g) notifyAll() - Wakes up all threads that are waiting on this object's monitor.
h) toString() - Returns a string representation of the object.
i) wait() - Causes current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object.

22. What is immutable class in Java?

⇨ Immutable classes are those class, whose object cannot be modified once created, it means any modification on immutable object will result in another immutable object. Best example to understand immutable and mutable objects are, String and StringBuffer. Since String is immutable class, any change on existing string object will result in another string e.g. replacing a character into String, creating substring from String, all result in a new objects. While in case of mutable object like StringBuffer, any modification is done on object itself and no new objects are created. Sometimes this immutability of String can also cause security hole, and that the reason why password should be stored on char array instead of String.

23. How to write immutable class in Java?

⇨ Despite of few disadvantages, Immutable object still offers several benefits in multi-threaded programming and it's a great choice to achieve thread safety in Java code. here are few rules, which helps to make a class immutable in Java :

a) State of immutable object cannot be modified after construction, any modification should result in new immutable object.
b) All fields of Immutable class should be final.
c) Object must be properly constructed i.e. object reference must not leak during construction process.
d) Object should be final in order to restrict sub-class for altering immutability of parent class.

By the way, you can still create immutable object by violating few rules, like String has its hashcode in non-final field, but it's always guaranteed to be same. No matter how many times you calculate it, because it's calculated from final fields, which is guaranteed to be same. This required a deep knowledge of Java memory model, and can create subtle race conditions if not addressed properly. In next section we will see simple example of writing immutable class in Java. By the way, if your Immutable class has lots of optional and mandatory fields, then you can also use Builder design pattern to make a class Immutable in Java.

24. Benefits of Immutable Classes in Java?

a) Immutable objects are by default thread safe, can be shared without synchronization in concurrent environment.
b) Immutable object simplifies development, because it's easier to share between multiple threads without external synchronization.
c) Immutable object boost performance of Java application by reducing synchronization in code.
d) Another important benefit of Immutable objects is reusability, you can cache Immutable object and reuse them, much like String literals and Integers. You can use static factory methods to provide methods like valueOf(), which can return an existing Immutable object from cache, instead of creating a new one.

Apart from above advantages, immutable object has disadvantage of creating garbage as well. Since immutable object cannot be reused and they are just a use and throw. String being a prime example, which can create lot of garbage and can potentially slow down application due to heavy garbage collection, but again that's extreme case and if used properly Immutable object adds lot of values.

Serialization:-

1. What is Serialization in Java

⇨ Object Serialization in Java is a process used to convert Object into a binary format which can be persisted into disk or sent over network to any other running Java virtual machine; the reverse process of creating object from binary stream is called deserialization in Java. Java provides Serialization API for serializing and deserializing object which includes java.io.Serializable, java.io.Externalizable, ObjectInputStream and ObjectOutputStream etc. Java programmers are free to use default Serialization mechanism which Java uses based upon structure of class but they are also free to use their own custom binary format, which is often advised as Serialization best practice, Because serialized binary format becomes part of Class's exported API and it can potentially break Encapsulation in Java provided by private and package-private fields.

2. **How to make a Java class Serializable?**

⇨ Making a class Serializable in Java is very easy, Your Java class just needs to implements java.io.Serializable interface and JVM will take care of serializing object in default format. Decision to making a Class Serializable should be taken concisely because though near term cost of making a Class Serializable is low, long term cost is substantial and it can potentially limit your ability to further modify and change its implementation because like any public API, serialized form of an object becomes part of public API and when you change structure of your class by implementing addition interface, adding or removing any field can potentially break default serialization, this can be minimized by using a custom binary format but still requires lot of effort to ensure backward compatibility. One example of How Serialization can put constraints on your ability to change class is SerialVersionUID. If you don't explicitly declare SerialVersionUID then JVM generates its based upon structure of class which depends upon interfaces a class implements and several other factors which is subject to change. Suppose you implement another interface than JVM will generate a different SerialVersionUID for new version of class files and when you try to load old object serialized by old version of your program you will get InvalidClassException.

3. **What is the difference between Serializable and Externalizable interface in Java?**

⇨ This is most frequently asked question in Java serialization interview. Here is my version Externalizable provides us writeExternal() and readExternal() method which gives us flexibility to control java serialization mechanism instead of relying on Java's default serialization. Correct implementation of Externalizable interface can improve performance of application drastically.

4. **How many methods Serializable has? If no method then what is the purpose of Serializable interface?**

⇨ Serializable interface exists in java.io package and forms core of java serialization mechanism. It doesn't have any method and also called Marker Interface in Java. When your class implements java.io.Serializable interface it becomes Serializable in Java and gives compiler an indication that use Java Serialization mechanism to serialize this object.

5. **What is serialVersionUID? What would happen if you don't define this?**

⇨ SerialVersionUID is an ID which is stamped on object when it get serialized usually hash code of object, you can use tool serialver to see serialVersionUID of a serialized object. SerialVersionUID is used for version control of object. You can specify serialVersionUID in your class file also. Consequence of not specifying serialVersionUID is that when you add or modify any field in class then already serialized class will not be able to recover because serialVersionUID generated for new class and for old serialized object will be different. Java serialization process relies on correct serialVersionUID for recovering state of serialized object and throws java.io.InvalidClassException in case of serialVersionUID mismatch.

6. **While serializing you want some of the members not to serialize? How do you achieve it?**

⇨ This is sometime also asked as what is the use of transient variable, does transient and static variable gets serialized or not etc. so if you don't want any field to be part of object's state then declare it either static or transient based on your need and it will not be included during Java serialization process.

7. **What will happen if one of the members in the class doesn't implement Serializable interface?**

⇨ If you try to serialize an object of a class which implements Serializable, but the object includes a reference to an non- Serializable class then a 'NotSerializableException' will be thrown at runtime and this is why I always put a Serializable Alert (comment section in my code) , one of the code comment best practices, to instruct developer to remember this fact while adding a new field in a Serializable class.

8. **If a class is Serializable but its super class in not, what will be the state of the instance variables inherited from super class after deserialization?**

⇨ Java serialization process only continues in object hierarchy till the class is Serializable i.e. implements Serializable interface in Java and values of the instance variables inherited from super class will be initialized by calling constructor of Non-Serializable Super class during deserialization process. Once the constructor chaining will started it wouldn't be possible to stop that, hence even if classes higher in hierarchy implements Serializable interface, there constructor will be executed. As you see from the statement this Serialization interview question looks very tricky and tough but if you are familiar with key concepts it's not that difficult.

9. **Can you Customize Serialization process or can you override default Serialization process in Java?**

⇨ The answer is yes you can. We all know that for serializing an object ObjectOutputStream.writeObject (saveThisobject) is invoked and for reading object ObjectInputStream.readObject() is invoked but there is one more thing which Java Virtual Machine provides you is to define these two method in your class. If you define these two methods in your class then JVM will invoke these two methods instead of applying default serialization mechanism. You can customize behavior of object serialization and deserialization here by doing any kind of pre or post processing task. Important point to note is making these methods private to avoid being inherited, overridden or overloaded. Since only Java Virtual Machine can call private method integrity of your class will remain and Java Serialization will work as normal. In my opinion this is one of the best question one can ask in any Java Serialization interview, a good follow-up question is why should you provide custom serialized form for your object?

10. **Suppose super class of a new class implement Serializable interface, how can you avoid new class to being serialized?**

⇨ If Super Class of a Class already implements Serializable interface in Java then its already Serializable in Java, since you cannot unimplemented an interface it's not really possible to make it Non Serializable class but yes there is a way to avoid serialization of new class. To avoid java serialization you need to implement writeObject() and readObject() method in your Class and need to throw NotSerializableException from those method. This is another benefit of customizing java serialization process as described in above Serialization interview question and normally it asked as follow-up question as interview progresses.

11. **Which methods are used during Serialization and Deserialization process in java?**

⇨ This is very common interview question in Serialization basically interviewer is trying to know; Whether you are familiar with usage of readObject(), writeObject(), readExternal() and writeExternal () or not. Java Serialization is done by java.io.ObjectOutputStream class. That class is a filter stream which is wrapped around a lower-level byte stream to handle the serialization mechanism. To store any object via serialization mechanism we call ObjectOutputStream.writeObject(saveThisobject) and to deserialize that object we call ObjectInputStream.readObject() method. Call to writeObject() method trigger serialization process in java. one important thing to note about readObject() method is that it is used to read bytes from the persistence and to create object from those bytes and its return an Object which needs to be casted on correct type.

12.**Suppose you have a class which you serialized it and stored in persistence and later modified that class to add a new field. What will happen if you deserialize the object already serialized?**

⇨ It depends on whether class has its own serialVersionUID or not. As we know from above question that if we don't provide serialVersionUID in our code java compiler will generate it and normally it's equal to hashCode of object. by adding any new field there is chance that new serialVersionUID generated for that class version is not the same of already serialized object and in this case Java Serialization API will throw java.io.InvalidClassException and this is the reason its recommended to have your own serialVersionUID in code and make sure to keep it same always for a single class.

13. **What are the compatible changes and incompatible changes in Java Serialization Mechanism?**

⇨ The real challenge lies with change in class structure by adding any field, method or removing any field or method is that with already serialized object***.*** *As per Java Serialization specification adding any field or method comes under* compatible change and changing class hierarchy or UN-implementing Serializable interfaces some under non compatible changes. For complete list of compatible and non-compatible changes I would advise reading Java serialization specification.

14. **Can we transfer a Serialized object vie network?**

⇨ *Yes you can transfer a Serialized object via network* because java serialized object remains in form of bytes which can be transmitted via network. You can also store serialized object in Disk or database as Blob.

15. **Which kind of variables is not serialized during Java Serialization?**

⇨ This question asked sometime differently but the purpose is same whether Java developer knows specifics about static and transient variable or not. Since *static variables belong to the class* and not to an object they are not the part of the state of object so they are not saved during Java Serialization process. As Java Serialization only persist state of object and not object itself. Transient variables are also not included in java serialization process and are not the part of the object's serialized state. After this question sometime interviewer ask a follow-up if you don't store values of these variables then what would be value of these variable once you deserialize and recreate those object?

Threads:-

1. **Difference between preemptive scheduling and time slicing?**

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

2. **Sleep method in Java?**

⇨ The sleep () method of Thread class is used to sleep a thread for the specified amount of time.
<u>Syntax of sleep () method:</u>
The Thread class provides two methods for sleeping a thread:
a) public static void sleep(long miliseconds)throws InterruptedException
b) public static void sleep(long miliseconds, int nanos)throws InterruptedException

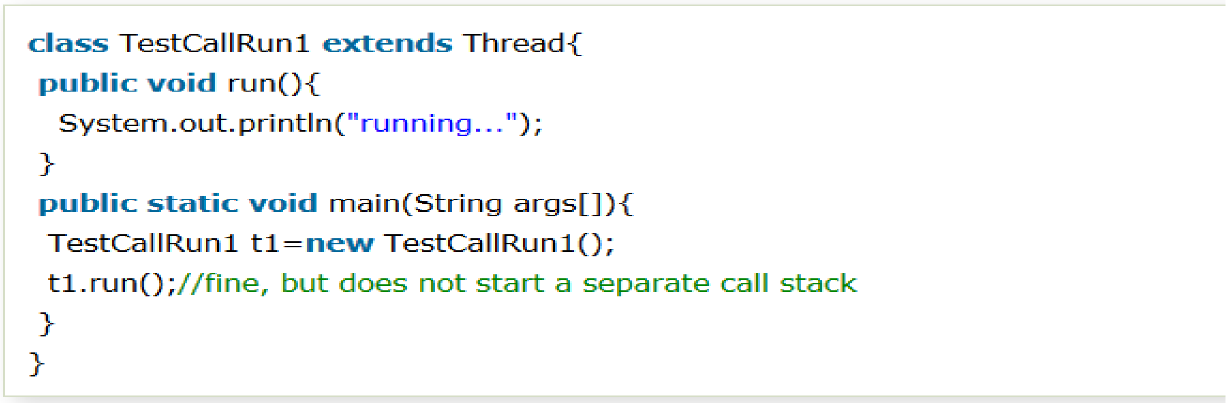
If you sleep a thread for the specified time,the thread shedular picks up another thread and so on.

3. **Can we start a Thread twice?**

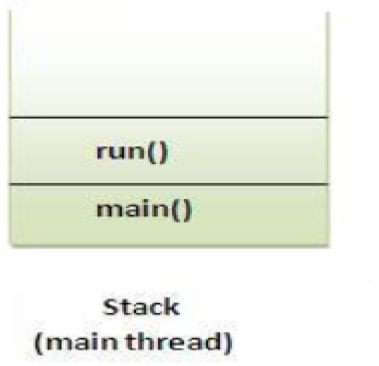
⇨ No. After starting a thread, it can never be started again. If you does so, an IllegalThreadStateException is thrown. In such case, thread will run once but for second time, it will throw exception.

4. **What if we call run() method directly instead start() method?**

a) Each thread starts in a separate call stack.
b) Invoking the run() method from main thread, the run() method goes onto the current call stack rather than at the beginning of a new call stack.
c) Thread will be treated as normal object not thread object.



```java
class TestCallRun1 extends Thread{
 public void run(){
   System.out.println("running...");
 }
 public static void main(String args[]){
  TestCallRun1 t1=new TestCallRun1();
  t1.run();//fine, but does not start a separate call stack
 }
}
```



Problem if you direct call run() method

5. **The join() method**

⇨ The non-static join() method of class Thread lets one thread "join onto the end" of another thread. If you have a thread B that can't do its work until another thread A has completed its work, then you want thread B to "join" thread A. This means that thread B will not become runnable until A has finished (and entered the dead state).

   a) public void join()throws InterruptedException
   b) public void join(long milliseconds)throws InterruptedException

6. **What is Thread in Java?**

⇨ The thread is an independent path of execution. It's way to take advantage of multiple CPU available in a machine. By employing multiple threads you can speed up CPU bound task. For example, if one thread takes 100 milliseconds to do a job, you can use 10 thread to reduce that task into 10 milliseconds. Java provides excellent support for multithreading at the language level, and it's also one of the strong selling points.

7. **What is the difference between Thread and Process in Java?**

⇨ The thread is a subset of Process, in other words, one process can contain multiple threads. Two process runs on different memory space, but all threads share same memory space. Don't confuse this with stack memory, which is different for the different thread and used to store local data to that thread. For more detail see the answer.

8. **How do you implement Thread in Java?**

⇨ At the language level, there are two ways to implement Thread in Java. An instance of java.lang.Thread represent a thread but it needs a task to execute, which is an instance of interface java.lang.Runnable. Since Thread class itself implement Runnable, you can override run() method either by extending Thread class or just implementing Runnable interface. For detailed answer and discussion see this article.

9. **When to use Runnable vs Thread in Java?**

⇨ This is a follow-up of previous multi-threading interview question. As we know we can implement thread either by extending Thread class or implementing Runnable interface, the question arise, which one is better and when to use one? This question will be easy to answer if you know that Java programming language doesn't support multiple inheritances of class, but it allows you to implement multiple interfaces. Which means, it's better to implement Runnable then extends Thread if you also want to extend another class e.g. Canvas or CommandListener. For more points and discussion you can also refer this post.

10. **What is the difference between start () and run () method of Thread class?**

⇨ When program calls start () method a new Thread is created and code inside run () method is executed in new Thread while if you call run () method directly no new Thread is created and code inside run () will execute on current Thread.
⇨ If you want to perform time consuming task than always call start () method otherwise your main thread will stuck while performing time consuming task if you call run () method directly.
⇨ You cannot call start () method twice on thread object. Once started, second call of start () will throw IllegalStateException in Java while you can call run () method twice.

```
public class StartVsRunCall{

    public static void main(String args[]) {

        //creating two threads for start and run method call
        Thread startThread = new Thread(new Task("start"));
        Thread runThread = new Thread(new Task("run"));

        startThread.start(); //calling start method of Thread - will execute in new Thread
        runThread.run();  //calling run method of Thread - will execute in current Thread

    }

    /*
     * Simple Runnable implementation
     */
    private static class Task implements Runnable{
        private String caller;

        public Task(String caller){
            this.caller = caller;
        }

        @Override
        public void run() {
            System.out.println("Caller: "+ caller + " and code on this Thread is executed by :
" + Thread.currentThread().getName());

        }
    }
}

Output:
Caller: start and code on this Thread is executed by : Thread-0
Caller: run and code on this Thread is executed by : main
```

11. **What is the difference between Runnable and Callable in Java?**

⇨ The Runnable interface is older than Callable, there from JDK 1.0, while Callable is added on Java 5.0.
⇨ Runnable interface has run() method to define task while Callable interface uses call() method for task definition.

- ⇨ run() method does not return any value, it's return type is void while call method returns value. The Callable interface is a generic parameterized interface and Type of value is provided when an instance of Callable implementation is created.
- ⇨ Another difference on run and call method is that run method cannot throw checked exception while call method can throw checked exception in Java

12. **What is the difference between CyclicBarrier and CountDownLatch in Java?**

⇨ **<u>What is CyclicBarrier:-</u>**

CyclicBarrier in Java is a synchronizer introduced in JDK 5 on java.util.Concurrent package along with other concurrent utility like Counting Semaphore, BlockingQueue, ConcurrentHashMap etc. CyclicBarrier is similar to CountDownLatch allows multiple threads to wait for each other (barrier) before proceeding.

CyclicBarrier is a natural requirement for a concurrent program because it can be used to perform final part of the task once individual tasks are completed. All threads which wait for each other to reach barrier are called parties, CyclicBarrier is initialized with a number of parties to wait and threads wait for each other by calling CyclicBarrier.await() method which is a blocking method in Java and blocks until all Thread or parties call await(). In general calling await() is shout out that Thread is waiting on the barrier. await() is a blocking call but can be timed out or Interrupted by other thread. In this Java concurrency tutorial, we will see What is CyclicBarrier in Java and an example of CyclicBarrier on which three Threads will wait for each other.

⇨ If you look at CyclicBarrier is also the does the same thing but there is different you cannot reuse CountDownLatch once the count reaches zero while you can reuse CyclicBarrier by calling reset () method which resets Barrier to its initial State. What it implies that CountDownLatch is a good for one-time events like application start-up time and CyclicBarrier can be used to in case of the recurrent event e.g. concurrently calculating a solution of the big problem etc.

⇨ **<u>When to use CyclicBarrier:-</u>**

Given the nature of CyclicBarrier it can be very handy to implement map reduce kind of task similar to fork-join framework of Java 7, where a big task is broken down into smaller pieces and to complete the task you need output from individual small task e.g. to count population of India you can have 4 threads which count population from North, South, East, and West and once complete they can wait for each other, When last thread completed their task, Main thread or any other thread can add result from each zone and print total population. You can use CyclicBarrier in Java:

a) To implement multi player game which cannot begin until all player has joined.
b) Perform lengthy calculation by breaking it into smaller individual tasks, In general, to implement Map reduce technique.

⇨ **<u>Important point of CyclicBarrier:-</u>**

a) CyclicBarrier can perform a completion task once all thread reaches to the barrier; this can be provided while creating CyclicBarrier.
b) If CyclicBarrier is initialized with 3 parties means 3 threads needs to call await method to break the barrier.
c) The thread will block on await () until all parties reach to the barrier, another thread interrupt or await timed out.
d) If another thread interrupts the thread which is waiting on barrier it will throw BrokernBarrierException

e) CyclicBarrier.reset() put Barrier on its initial state, other thread which is waiting or not yet reached barrier will terminate with java.util.concurrent.BrokenBarrierException.

13. **What is Java Memory model?**

⇨ Java Memory model is set of rules and guidelines which allows Java programs to behave deterministically across multiple memory architecture, CPU, and operating system. It's particularly important in case of multi-threading. Java Memory Model provides some guarantee on which changes made by one thread should be visible to others, one of them is happens-before relationship. This relationship defines several rules which allows programmers to anticipate and reason behavior of concurrent Java programs. For example, happens-before relationship guarantees:

a) Each action in a thread happens-before every action in that thread that comes later in the program order, this is known as program order rule.
b) An unlock on a monitor lock happens-before every subsequent lock on that same monitor lock, also known as Monitor lock rule.
c) A write to a volatile field happens-before every subsequent read of that same field, known as Volatile variable rule.
d) A call to Thread.start on a thread happens-before any other thread detects that thread has terminated, either by successfully return from Thread.join() or by Thread.isAlive() returning false, also known as Thread start rule.
e) A thread calling interrupt on another thread happens-before the interrupted thread detects the interrupt (either by having InterruptedException thrown, or invoking isInterrupted or interrupted), popularly known as Thread Interruption rule.
f) The end of a constructor for an object happens-before the start of the finalizer for that object, known as Finalizer rule.
g) If A happens-before B, and B happens-before C, then A happens-before C, which means happens-before guarantees Transitivity.

14. **What is volatile variable in Java?**

⇨ The volatile keyword in Java is used as an indicator to Java compiler and Thread that do not cache value of this variable and always read it from main memory. So if you want to share any variable in which read and write operation is atomic by implementation e.g. read and write in an int or a boolean variable then you can declare them as volatile variable.

15. **What is thread-safety? Is Vector a thread-safe class?**

⇨ Thread-safety is a property of an object or code which guarantees that if executed or used by multiple threads in any manner e.g. read vs write it will behave as expected. For example, a thread-safe counter object will not miss any count if same instance of that counter is shared among multiple threads. Apparently, you can also divide collection classes in two category, thread-safe and non-thread-safe. Vector is indeed a thread-safe class and it achieves thread-safety by synchronizing methods which modify state of Vector, on the other hand, its counterpart ArrayList is not thread-safe.

16. **What is race condition in Java? Given one example?**

Race condition in Java is a type of concurrency bug or issue which is introduced in your program because parallel execution of your program by multiple threads at same time, Since Java is a multi-threaded programming language hence risk of Race condition is higher in Java which demands clear understanding of what causes a race condition and how to avoid that. Anyway Race conditions are just one of hazards or risk presented by use of multi-threading in Java just like deadlock in Java. Race condition occurs when two threads operate on same object without proper synchronization and there operation interleaves on each other. Classical example of Race condition is incrementing a counter since increment is not an atomic operation and can be further divided into three steps like read, update and write. if two threads tries to increment count at same time and if they read same value because of interleaving of read operation of one thread to update operation of another thread, one count will be lost when one thread overwrite increment done by other thread. Atomic operations are not subject to race conditions because those operations cannot be interleaved.

Read more: http://javarevisited.blogspot.com/2012/02/what-is-race-condition-in.html#ixzz4V6yh4Mn1

Code Example of Race Condition in Java

a) "Check and Act" race condition pattern

Classical example of "check and act" race condition in Java is getInstance() method of Singleton Class. getInstace() method first check for whether instance is null and then initialized the instance and return to caller. Whole purpose of Singleton is that getInstance () should always return same instance of Singleton. if you call getInstance() method from two thread simultaneously it's possible that while one thread is initializing singleton after null check, another thread sees value of _instance reference variable as null (quite possible in java) especially if your object takes longer time to initialize and enters into critical section which eventually results in getInstance() returning two separate instance of Singleton. This may not happen always because a fraction of delay may result in value of _instance updated in main memory. Here is a code example

```
public Singleton getInstance(){
if(_instance == null){   //race condition if two threads sees _instance= null
_instance = new Singleton();
}
}
```

An easy way to fix "check and act" race conditions is to synchronized keyword and enforce locking which will make this operation atomic and guarantees that block or method will only be executed by one thread and result of operation will be visible to all threads once synchronized blocks completed or thread exited form synchronized block.

b) read-modify-update race conditions

This is another code pattern in Java which causes race condition; classical example is the non-thread safe counter. Read-modify-update pattern also comes due to improper synchronization of non-atomic operations or combination of two individual atomic operations which is not atomic together e.g. put if absent scenario. Consider below code

```
if(!hashtable.contains(key)){
hashtable.put(key,value);
}
```

Here we only insert object into hashtable if it's not already there. Point is both contains () and put () are atomic but still this code can result in race condition since both operation together is not atomic. Consider thread T1 checks for conditions and goes inside if block now CPU is switched from T1 to thread T2 which also checks condition and goes inside if block. Now we have two threads inside if block which result in either T1 overwriting T2 value or vice-versa based on which thread has CPU for execution. In order to fix this race condition in Java you need to wrap this code inside synchronized block which makes them atomic together because no thread can go inside synchronized block if one thread is already there.

17. How to stop a thread in Java?

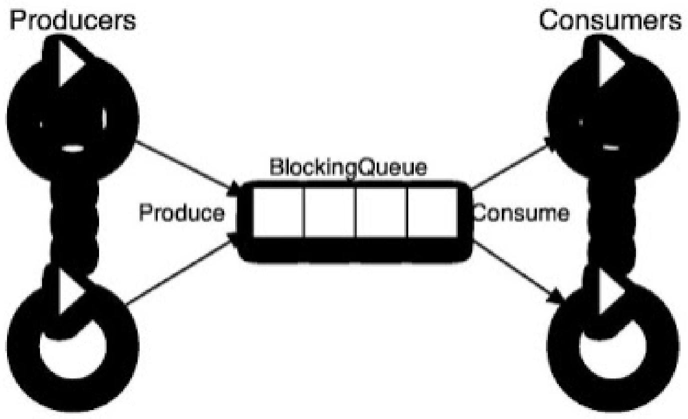
⇨ I always said that Java provides rich APIs for everything but ironically Java doesn't provide a sure shot way of stopping thread. There was some control methods in JDK 1.0 e.g. stop(), suspend() and resume() which was deprecated in later releases due to potential deadlock threats, from then Java API designers has not made any effort to provide a consistent, thread-safe and elegant way to stop threads. Programmers mainly rely on the fact that thread stops automatically as soon as they finish execution of run() or call() method. To manually stop, programmers either take advantage of volatile boolean variable and check in every iteration if run method has loops or interrupt threads to abruptly cancel tasks. See this tutorial for sample code of stopping thread in Java.

18. **What happens when an Exception occurs in a thread?**

⇨ This is one of the good tricky Java question I have seen in interviews. In simple words, If not caught thread will die, if an uncaught exception handler is registered then it will get a call back. Thread.UncaughtExceptionHandler is an interface, defined as nested interface for handlers invoked when a Thread abruptly terminates due to an uncaught exception. When a thread is about to terminate due to an uncaught exception the Java Virtual Machine will query the thread for its UncaughtExceptionHandler using Thread.getUncaughtExceptionHandler() and will invoke the handler's uncaughtException() method, passing the thread and the exception as arguments.

19. **How do you share data between two thread in Java?**

⇨ You can share data between threads by using shared object, or concurrent data structure like BlockingQueue. See this tutorial to learn inter-thread communication in Java. It implements Producer consumer pattern using wait and notify methods, which involves sharing objects between two threads.



20. **What is the difference between notify and notifyAll in Java?**

⇨ This is another tricky questions from core Java interviews, since multiple threads can wait on single monitor lock, Java API designer provides method to inform only one of them or all of them, once waiting condition changes, but they provide half implementation. There notify() method doesn't provide any way to choose a particular thread, that's why it's only useful when you know that there is only one thread is waiting. On the other hand, notify All() sends notification to all threads and allows them to compete for locks, which ensures that at-least one thread will proceed further.

21. **Why wait, notify and notifyAll are not inside thread class?**

⇨ This is a design related question, which checks what candidate thinks about existing system or does he ever thought of something which is so common but looks in-appropriate at first. In order to answer this question, you have to give some reasons why it make sense for these three method to be in Object class, and why not on Thread class. One reason which is obvious is that Java provides lock at object level not at thread level. Every object has lock, which is acquired by thread. Now if thread needs to wait for certain lock it make sense to call wait() on that object rather than on that thread. Had wait() method declared on Thread class, it was not clear that for which lock thread was waiting.

In short, since wait, notify and notifyAll operate at lock level, it make sense to defined it on object class because lock belongs to object.

22. **What is ThreadLocal variable in Java?**

⇨ ThreadLocal variables are special kind of variable available to Java programmer. Just like instance variable is per instance, ThreadLocal variable is per thread. It's a nice way to achieve thread-safety of expensive-to-create objects, for example you can make SimpleDateFormat thread-safe using ThreadLocal. Since that class is expensive, it's not good to use it in local scope, which requires separate instance on each invocation. By providing each thread their own copy, you shoot two birds with one arrow. First, you reduce number of instance of expensive object by reusing fixed number of instances, and second, you achieve thread-safety without paying cost of synchronization or immutability. Another good example of thread local variable is ThreadLocalRandom class, which reduces number of instances of expensive-to-create Random object in multi-threading environment.

23. **What is FutureTask in Java?**

⇨ FutureTask represents a cancellable asynchronous computation in concurrent Java application. This class provides a base implementation of Future, with methods to start and cancel a computation, query to see if the computation is complete, and retrieve the result of the computation. The result can only be retrieved when the computation has completed; the get methods will block if the computation has not yet completed. A FutureTask object can be used to wrap a Callable or Runnable object. Since FutureTask also implements Runnable, it can be submitted to an Executor for execution.

24. **What is the difference between the interrupted () and isInterrupted() method in Java?**

⇨ Main difference between interrupted () and isInterrupted() is that former clears the interrupt status while later does not. The interrupt mechanism in Java multi-threading is implemented using an internal flag known as the interrupt status. Interrupting a thread by calling Thread.interrupt() sets this flag. When interrupted thread checks for an interrupt by invoking the static method Thread.interrupted(), interrupt status is cleared. The non-static isInterrupted() method, which is used by one thread to query the interrupt status of another, does not change the interrupt status flag. By convention, any method that exits by throwing an InterruptedException clears interrupt status when it does so. However, it's always possible that interrupt status will immediately be set again, by another thread invoking interrupt

25. **Why wait and notify method are called from synchronized block?**

⇨ Main reason for calling wait and notify method from either synchronized block or method is that it made mandatory by Java API. If you don't call them from synchronized context, your code will throw IllegalMonitorStateException. A more subtle reason is to avoid the race condition between wait and notify calls.

26. **Why should you check condition for waiting in a loop?**

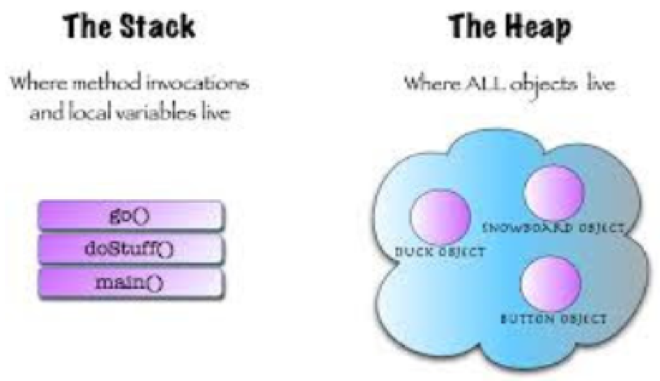
⇨ It's possible for a waiting thread to receive false alerts and spurious wake up calls, if it doesn't check the waiting condition in loop, it will simply exit even if condition is not met. As such, when a waiting thread wakes up, it cannot assume that the state it was waiting for is still valid. It may have been valid in the past, but the state may have been changed after the notify() method was called and before the waiting thread woke up. That's why it always better to call wait() method from loop, you can even create template for calling wait and notify in Eclipse. To learn more about this question, I would recommend you to read Effective Java items on thread and synchronization.

27. **What is the difference between synchronized and concurrent collection in Java?**

⇨ Though both synchronized and concurrent collection provides thread-safe collection suitable for multi-threaded and concurrent access, later is more scalable than former. Before Java 1.5, Java programmers only had synchronized collection which becomes source of contention if multiple thread access them concurrently, which hampers scalability of system. Java 5 introduced concurrent collections like ConcurrentHashMap, which not only provides thread-safety but also improves scalability by using modern techniques like lock stripping and partitioning internal table.

28. **What is the difference between Stack and Heap in Java?**

⇨ Why does someone this question as part of multi-threading and concurrency? Because Stack is a memory area which is closely associated with threads. To answer this question, both stack and heap are specific memories in Java application. Each thread has their own stack, which is used to store local variables, method parameters and call stack. Variable stored in one Thread's stack is not visible to other. On another hand, the heap is a common memory area which is shared by all threads. Objects whether local or at any level is created inside heap. To improve performance thread tends to cache values from heap into their stack, which can create problems if that variable is modified by more than one thread, this is where volatile variables come into the picture. volatile suggest threads read the value of variable always from main memory.



29. **What is thread pool? Why should you thread pool in Java?**

⇨ Creating thread is expensive in terms of time and resource. If you create thread at time of request processing it will slow down your response time, also there is only a limited number of threads a process can create. To avoid both of these issues, a pool of thread is created when application starts-up and threads are reused for request processing. This pool of thread is known as "thread pool" and threads are known as worker thread. From JDK 1.5 release, Java API provides Executor framework, which allows you to create different types of thread pools e.g. single thread pool, which process one task at a time, fixed thread pool (a pool of fixed number of threads) or cached thread pool (an expandable thread pool suitable for applications with many short lived tasks).

30. **Write code to solve Producer Consumer problem in Java?**

⇨ Most of the threading problem you solved in the real world are of the category of Producer consumer pattern, where one thread is producing task and another thread is consuming that. You must know how to do inter thread communication to solve this problem. At the lowest level, you can use wait and notify to solve this problem, and at a high level, you can leverage Semaphore or BlockingQueue to implement Producer consumer pattern, as shown in this tutorial.

31. **What is the difference between livelock and deadlock in Java?**

⇨ This question is extension of previous interview question. A livelock is similar to a deadlock, except that the states of the threads or processes involved in the livelock constantly change with regard to one another, without any one progressing further. Livelock is a special case of resource starvation. A real-world example of livelock occurs when two people meet in a narrow corridor, and each tries to be polite by moving aside to let the other pass, but they end up swaying from side to side without making any progress because they both repeatedly move the same way at the same time. In short, the main difference between livelock and deadlock is that in former state of process change but no progress is made.

32. **How do you check if a Thread holds a lock or not?**

⇨ I didn't even know that you can check if a Thread already holds lock before this question hits me in a telephonic round of Java interview. There is a method called holdsLock() on java.lang.Thread, it returns true if and only if the current thread holds the monitor lock on the specified object.

33. **How do you take thread dump in Java?**

⇨ There are multiple ways to take thread dump of Java process depending upon operating system. When you take thread dump, JVM dumps state of all threads in log files or standard error console. In windows you can use Ctrl + Break key combination to take thread dump, on Linux you can use kill -3 command for same. You can also use a tool called jstack for taking thread dump, it operate on process id, which can be found using another tool called jps.

34. **Which JVM parameter is used to control stack size of a thread?**

⇨ This is the simple one, -Xss parameter is used to control stack size of Thread in Java. You can see this list of JVM options to learn more about this parameter.

35. **What is the difference between synchronized and ReentrantLock in Java?**

⇨ There were days when the only way to provide mutual exclusion in Java was via synchronized keyword, but it has several shortcomings e.g. you cannot extend lock beyond a method or block boundary, you cannot give up trying for a lock etc. Java 5 solves this problem by providing more sophisticated control via Lock interface. ReentrantLock is a common implementation of Lock interface and provides re-entrant mutual exclusion Lock with the same basic behavior and semantics as the implicit monitor lock accessed using synchronized methods and statements, but with extended capabilities.

36. **There are three threads T1, T2, and T3? How do you ensure sequence T1, T2, T3 in Java?**

⇨ Sequencing in multi-threading can be achieved by different means but you can simply use the join() method of thread class to start a thread when another one has finished its execution. To ensure three threads execute you need to start the last one first e.g. T3 and then call join methods in reverse order e.g. T3 calls T2. Join and T2 calls T1.join, these ways T1 will finish first and T3 will finish last.

37. **What does yield method of Thread class do?**

⇨ Yield method is one way to request current thread to relinquish CPU so that other thread can get a chance to execute. Yield is a static method and only guarantees that current thread will relinquish the CPU but doesn't say anything about which other thread will get CPU. It's possible for the same thread to get CPU back and start its execution again.

38. **What is the concurrency level of ConcurrentHashMap in Java?**

⇨ ConcurrentHashMap achieves its scalability and thread-safety by partitioning actual map into a number of sections. This partitioning is achieved using concurrency level. Its optional parameter of ConcurrentHashMap constructor and its default value is 16. The table is internally partitioned to try to permit the indicated number of concurrent updates without contention.

39. **What is Semaphore in Java?**

⇨ Semaphore in Java is a new kind of synchronizer. It's a counting semaphore. Conceptually, a semaphore maintains a set of permits. Each acquire() blocks if necessary until a permit is available, and then takes it. Each release() adds a permit, potentially releasing a blocking acquirer. However, no actual permit objects are used; the Semaphore just keeps a count of the number available and acts accordingly. Semaphore is used to protect an expensive resource which is available in fixed number e.g. database connection in the pool.

40. **What happens if you submit a task when the queue of the thread pool is already filled?**

⇨ This is another tricky question on my list. Many programmers will think that it will block until a task is cleared but its true. ThreadPoolExecutor's submit() method throws RejectedExecutionException if the task cannot be scheduled for execution.

41. **What is the difference between the submit() and execute() method thread pool in Java?**
⇨ Both methods are ways to submit a task to thread pools but there is a slight difference between them. execute(Runnable command) is defined in Executor interface and executes given task in future, but more importantly, it does not return anything. Its return type is void. On other hand submit() is an overloaded method, it can take either Runnable or Callable task and can return Future object which can hold the pending result of computation. This method is defined on ExecutorService interface, which extends Executor interface, and every other thread pool class e.g. ThreadPoolExecutor or ScheduledThreadPoolExecutor gets these methods.

42. **What is blocking method in Java?**
⇨ A blocking method is a method which blocks until the task is done, for example, accept() method of ServerSocket blocks until a client is connected. Here blocking means control will not return to the caller until the task is finished. On the other hand, there is an asynchronous or non-blocking method which returns even before the task is finished.

43. **Is Swing thread-safe? What do you mean by Swing thread-safe?**
⇨ You can simply this question as No, Swing is not thread-safe, but you have to explain what you mean by that even if the interviewer doesn't ask about it. When we say swing is not thread-safe we usually refer its component, which cannot be modified in multiple threads. All update to GUI components has to be done on AWT thread, and Swing provides synchronous and asynchronous callback methods to schedule such updates. You can also read my article to learn more about swing and thread-safety to better answer this question.

44. **What is the difference between invokeAndWait and invokeLater in Java?**
⇨ These are two methods Swing API provides Java developers for updating GUI components from threads other than Event dispatcher thread. InvokeAndWait() synchronously update GUI component, for example, a progress bar, once progress is made, the bar should also be updated to reflect that change. If progress is tracked in a different thread, it has to call invokeAndWait() to schedule an update of that component by Event dispatcher thread. On another hand, invokeLater() is an asynchronous call to update components.

45. **Which method of Swing API are thread-safe in Java?**
⇨ This question is again related to swing and thread-safety though components are not thread-safe there is a certain method which can be safely called from multiple threads. I know about repaint (), and revalidate () being thread-safe but there are other methods on different swing components e.g. setText() method of JTextComponent, insert() and append() method of JTextArea class.

46. **How to create an Immutable object in Java?**
⇨ This question might not look related to multi-threading and concurrency, but it is. Immutability helps to simplify already complex concurrent code in Java. Since immutable object can be shared without any synchronization it’s very dear to Java developers. Core value object, which is meant to be shared among thread should be immutable for performance and simplicity. Unfortunately there is no @Immutable annotation in Java, which can make your object immutable, hard work must be done by Java developers. You need to keep basics like initializing state in constructor, no setter methods, no leaking of reference, keeping separate copy of mutable object to create Immutable object.

47. **What is ReadWriteLock in Java?**
⇨ In general, read write lock is the result of lock stripping technique to improve the performance of concurrent applications. In Java, ReadWriteLock is an interface which was added in Java 5 release. A ReadWriteLock maintains a pair of associated locks, one for read-only operations and one for writing. The read lock may be held simultaneously by multiple reader threads, so long as there are no writers. The write lock is exclusive. If you want you can implement this interface with your own set of rules, otherwise you can use ReentrantReadWriteLock, which comes along with JDK and supports a maximum of 65535 recursive write locks and 65535 read locks.

48. **What is busy spin in multi-threading?**
⇨ Busy spin is a technique which concurrent programmers employ to make a thread wait on certain condition. Unlike traditional methods e.g. wait(), sleep() or yield() which all involves relinquishing CPU control, this method does not relinquish CPU, instead it the just runs empty loop. Why would someone do that? To preserve CPU caches. In a multi-core system, it's possible for a paused thread to resume on a different core, which means rebuilding cache again. To avoid cost of rebuilding cache, programmer prefer to wait for much smaller time doing busy spin

49. **What is the difference between the volatile and atomic variable in Java?**
⇨ This is an interesting question for Java programmer, at first, volatile and atomic variable look very similar, but they are different. Volatile variable provides you happens-before guarantee that a write will happen before any subsequent write, it doesn't guarantee atomicity. For example count++ operation will not become atomic just by declaring count variable as volatile. On the other hand AtomicInteger class provides atomic method to perform such compound operation atomically e.g. getAndIncrement() is atomic replacement of increment operator. It can be used to atomically increment current value by one. Similarly you have atomic version for other data type and reference variable as well.

50. **What happens if a thread throws an Exception inside synchronized block?**
⇨ This is one trickier question for average Java programmer, if he can bring the fact about whether lock is released or not is a key indicator of his understanding. To answer this question, no matter how you exist synchronized block, either normally by finishing execution or abruptly by throwing exception, thread releases the lock it acquired while entering that synchronized block. This is actually one of the reasons I like synchronized block over lock interface, which requires explicit attention to release lock, generally this is achieved by releasing the lock in a finally block.

51. **What is double checked locking of Singleton?**
⇨ This is one of the very popular question on Java interviews, and despite its popularity, chances of candidate answering this question satisfactory is only 50%. Half of the time, they failed to write code for double checked locking and half of the time they failed how it was broken and fixed on Java 1.5. This is actually an old way of creating thread-safe singleton, which tries to optimize performance by only locking when Singleton instance is created first time, but because of complexity and the fact it was broken for JDK 1.4, I personally don't like it. Anyway, even if you not prefer this approach it’s good to know from interview point of view.

52. **How to create thread-safe Singleton in Java?**
⇨ This question is actually follow-up of the previous question. If you say you don't like double checked locking then Interviewer is bound to ask about alternative ways of creating thread-safe Singleton class. There are actually man, you can take advantage of class loading and static variable initialization feature of JVM to create instance of Singleton, or you can leverage powerful enumeration type in Java to create Singleton..

53. **How do you force to start a Thread in Java?**
⇨ This question is like how do you force garbage collection in Java, there is no way though you can make a request using System.gc() but it's not guaranteed. On Java multi-threading there is absolute no way to force start a thread, this is controlled by thread scheduler and Java exposes no API to control thread schedule. This is still a random bit in Java.

54. **What is the fork-join framework in Java?**
⇨ The fork join framework, introduced in JDK 7 is a powerful tool available to Java developer to take advantage of multiple processors of modern day servers. It is designed for work that can be broken into smaller pieces recursively. The goal is to use all the available processing power to enhance the performance of your application. One significant advantage of The fork/join framework is that it uses a work-stealing algorithm. Worker threads that run out of things to do can steal tasks from other threads that are still busy.

55. **What is the difference between calling wait() and sleep() method in Java multi-threading?**

⇨ Though both wait and sleep introduce some form of pause in Java application, they are the tool for different needs. Wait method is used for inter thread communication, it relinquishes lock if waiting for a condition is true and wait for notification when due to an action of another thread waiting condition becomes false. On the other hand sleep() method is just to relinquish CPU or stop execution of current thread for specified time duration. Calling sleep method doesn't release the lock held by current thread. You can also take look at this article to answer this question with more details.

Delloite:-

Interview Questions:

1. Design Pattern
2. Singleton Program
3. How you will write and call custom Exception class in your web component
4. How you will ensure that your custom exceptions that you have written is a compile time exception or run time exception
5. Throw and Throws
6. What are the JDBC Drivers
7. Steps to create JDBC connection
8. If you have frequent data changes operation then what you will use in terms of List, Set and Map and why?
9. Why you use Interface? And why you use Abstraction
10. How many levels you can go in Abstraction to define Abstract method in it.
11. What are the advantages of using Abstraction and Interface other than common Behavior and common contract?
12. JSP Include Action and Include directive
13. What will happen if you define a new class in your JSP what the compiler will do?
14. Synchronized Thread.
15. Given a List [a,b,c,d,a,b,d,r] write the efficient way to find the unique values in it.
16. What is a Session and how you will create a Session.
17. Inner Join and Self Join, explain with Example.
18. Have you work on Struts/Spring/
19. What is the DOM and SAX
20. What is the diff between DOM and SAX parser
21. Web Service.
22. What are the Inner classes and Anonymous class?
23. Reflection
24. JSP Life cycle
25. What do you mean by a Transition in JSP lifecycle?

When to use Interface and when Abstraction

1. If I have only the requirement, and we don't know anything about implementation then go for the Interface: Example **Servlet** itself a interface. Which means we have only the specification.
2. If we know the partial implementation then we should go with the Abstract ex. Generic Servlet && HttpServlet

? *Honeywell:-*

1) If you have a static variable in your class. You have created new class and create an instance of that class is null Then what will be the value of the variable.
e.g: if you have Employee class and Static Id variable. You have created new class Address then you have instantiated Employee class which is equal to null then what will be the value of Id variable?

2) How do you instantiate a class using fully qualified class name as a string?

3) Create Employee and Address class, Using Map add elements in it and iterate through it.

4) Method overloading and overriding example.
   Suppose testOveridingMethod(int id, string Name) and testOverride(int age, String MName);
   When I call it with testOveridingMethod(1, "abc") which method will be called?

5) Can we declare Protected variable in Interface?
6) Can we create an instance of the Abstract class?
==>> An abstract class can never be instantiated. Its sole purpose is to be extended (sub classed).

7) What are the differences between Interface and Abstract class?

8) Exception Handling: In try block you are having an arithmetic operation such as divide by zero and you have two catch blocks, 1st catch block declare java exception and 2nd catch block declare custom exception which catch block will be executed?

9) How to make a Java class Serializable practical implementation for example Employee class?

10) Suppose Employee is parent class and 3rd party class you cannot modified it then how will you make variables in parent class as transient?