**1. Explain the usage of interfaces / abstract classes.**

**Interface**

>one of them is Java's feature to provide multiple inheritance at interface level

>It allows you to write flexible code, which can adapt to handle future requirements.

>You expect that unrelated classes would implement your interface. For example, the interfaces Comparable and Cloneable are implemented by many unrelated classes.

>You want to specify the behavior of a particular data type, but not concerned about who implements its behavior.

>You want to take advantage of multiple inheritances.

**Abstract**

>You want to share code among several closely related classes.

>You expect that classes that extend your abstract class have many common methods or fields or require access modifiers other than public (such as protected and private).

>You want to declare non-static or non-final fields. This enables you to define methods that can access and modify the state of the object to which they belong.

**2. Which features were added to interfaces in Java 8 and Java 9? Why? (like default methods, private static methods)**

>Lambda expression & Functional Interfaces

>forEach() method in Iterable interface.

>default and static methods in Interfaces.

>Java Stream API for Bulk Data Operations on Collections.

>Java Time API.

>Collection API improvements.

>Concurrency API improvements.

>Java IO improvements.

**3. What is an immutable class? Explain the steps involved in creating an immutable class.**

Immutable class means that once an object is created, we cannot change its content. In Java, all the wrapper class (like Integer, Boolean, Byte, Short) and String class is immutable. We can create our own immutable class as well.

steps involved in creating an immutable class

>The class must be declared as final (So that child classes can’t be created)

>Data members in the class must be declared as final (So that we can’t change the value of it after object creation)

>A parameterized constructor

>Getter method for all the variables in it

>No setters(To not have the option to change the value of the instance variable)

**4. Is it necessary that all properties of immutable class be final?**

Yes

**5. Why is String class in Java immutable?**

>The string is Immutable in Java because String objects are cached in String pool. Since cached String literals are shared between multiple clients there is always a risk, where one client's action would affect all another client. For example, if one client changes the value of String "demo" to "DEMO", all other clients will also see that value has been changed.

>Strings are very popular as HashMap key, it's important for them to be immutable so that they can retrieve the value object which was stored in HashMap. Since HashMap works in the principle of hashing, which requires same has value to function properly. Mutable String would produce two different hashcodes at the time of insertion and retrieval if contents of String was modified after insertion, potentially losing the value object in the map.

**6. Explain the marker interfaces in Java with example. Explain Serializable and Cloneable.**

Marker Interface in java is an interface with no fields or methods within it. It is used to convey to the JVM that the class implementing an interface of this category will have some special behavior.

>Serializable

Serializable interface is present in java.io package. It is used to make an object eligible for saving its state into a file. This is called Serialization.

import java.io.\*;

class A implements Serializable

{

int i;

String s;

// A class constructor

public A(int i,String s)

{

this.i = i;

this.s = s;

}

}

public class Test

{

public static void main(String[] args)

throws IOException, ClassNotFoundException

{

A a = new A(20,"GeeksForGeeks");

// Serializing 'a'

FileOutputStream fos = new FileOutputStream("xyz.txt");

ObjectOutputStream oos = new ObjectOutputStream(fos);

oos.writeObject(a);

// De-serializing 'a'

FileInputStream fis = new FileInputStream("xyz.txt");

ObjectInputStream ois = new ObjectInputStream(fis);

A b = (A)ois.readObject();//down-casting object

System.out.println(b.i+" "+b.s);

// closing streams

oos.close();

ois.close();

}

}

>Clonable

Cloneable interface is present in java.lang package. There is a method clone() in Object class. A class that implements the Cloneable interface indicates that it is legal for clone() method to make a field-for-field copy of instances of that class.

import java.lang.Cloneable;

class A implements Cloneable

{

int i;

String s;

// A class constructor

public A(int i,String s)

{

this.i = i;

this.s = s;

}

@Override

protected Object clone()

throws CloneNotSupportedException

{

return super.clone();

}

}

public class Test

{

public static void main(String[] args)

throws CloneNotSupportedException

{

A a = new A(20, "GeeksForGeeks");

// cloning 'a' and holding

// new cloned object reference in b

// down-casting as clone() return type is Object

A b = (A)a.clone();

System.out.println(b.i);

System.out.println(b.s);

}

}

**7. Is Externalizable a marker interface? Difference between Serializable and Externalizable.**

No,

As the name suggest it is externalizing your serialization.If you want to customize your serialization mechanism then you can use it.It uses custom written mechanism to perform marshaling and unmarshalling of objects.Externalizable interface extends Serializable interface. If you implement this interface, you need to provide implementation of readExternal() and writeExternal() method.

Parameter

Serializable

Externalizable

Marker interface

It is marker interface. You don’t have to provide implementation of any method.

Externalizable is not marker interface, you have to override writeExternal and readExternal method.

Control

Serializable interface has less control over serialization process and it is optional to override readObject and writeObject.

Externalizable interface has more control over serialization process and it is mandatory to override writeExternal and readExternal.

Performance

JVM uses reflection to perform serialization in the case of Serializable interface which is quite slow.

Programmer have to implement readExternal and writeExternal methods but it relatively results in better performance

Supersedes

NA

If you implement Externalizable interface and provide implementation of readExternal and writeExternal then it supersedes readObject and writeObject methods in that class. It is due to the fact that Externalizable extends Serializable interface.

Constructor called during Deserialization

Default constructor is not called during Deserialization process.

Default constructor is called during Deserialization process.

**8. What is the difference between checked and unchecked exception? Give examples for each from the JDK.**

>Checked: are the exceptions that are checked at compile time. If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using throws keyword.

ex:-FileNotFoundException , IOException,SQLException & ClassNotFoundException.

class Main {

    public static void main(String[] args) throws IOException

{

        FileReader file = new FileReader("C:\\test\\a.txt");

        BufferedReader fileInput = new BufferedReader(file);

        for (int counter = 0; counter < 3; counter++)

{

            System.out.println(fileInput.readLine());

}

        fileInput.close();

    }

}

>Unchecked are the exceptions that are not checked at compiled time.It means if your program is throwing an unchecked exception and even if you didn’t handle/declare that exception, the program won’t give a compilation error.

ex:-NullPointerException,ArrayIndexOutOfBoundsException,ArithmeticException,IllegalArgumentException & NumberFormatException

class Main {

   public static void main(String args[]) {

      int x = 0;

      int y = 10;

      int z = y/x;

  }

}

**9. Give examples for custom exceptions which are checked and unchecked exception.**

Java provides us facility to create our own exceptions which are basically derived classes of Exception.

>Custom Checked Exception

public class NameNotFoundException extends Exception {

    public NameNotFoundException(String message) {

        super(message);

    }

}

public class CustomerService {

    public Customer findByName(String name) throws NameNotFoundException {

        if ("".equals(name)) {

            throw new NameNotFoundException("Name is empty!");

        }

        return new Customer(name);

    }

    public static void main(String[] args) {

        CustomerService obj = new CustomerService();

        try {

            Customer cus = obj.findByName("");

        } catch (NameNotFoundException e) {

            e.printStackTrace();

        }

    }

}

>Custom Uncheked Exception

public class ListTooLargeException extends RuntimeException{

    public ListTooLargeException(String message) {

        super(message);

    }

}

public class CustomerService {

    public void analyze(List<String> data) {

        if (data.size() > 50) {

            //runtime exception

            throw new ListTooLargeException("List can't exceed 50 items!");

        }

//...

    }

    public static void main(String[] args) {

        CustomerService obj = new CustomerService();

//create 100 size

        List<String> data = new ArrayList<>(Collections.nCopies(100, "mkyong"));

        obj.analyze(data);

    }

}

**10. What is the difference between Exception and Error? Give examples of Errors.**

Error Exception

Classified as an unchecked type

Classified as checked and unchecked

It belongs to java.lang.error

It belongs to java.lang.Exception

It is irrecoverable

It is recoverable

It can't be occur at compile time

It can occur at run time compile time both

OutOfMemoryError ,IOError

NullPointerException , SqlException

**11. What is the difference between ClassNotFoundException and NoClassDefFoundError?**

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 runtime.

**12. Explain how class loader works in Java. Can one class be loaded by two class loaders in Java.**

The Java ClassLoader is a part of the Java Runtime Environment that dynamically loads Java classes into the Java Virtual Machine. The Java run time system does not need to know about files and file systems because of classloaders.

A Java Classloader is of **three types**:

1. **BootStrap ClassLoader:** A Bootstrap Classloader is a Machine code which kickstarts the operation when the JVM calls it. It is not a java class. Its job is to load the first pure Java ClassLoader. Bootstrap ClassLoader loads classes from the location ***rt.jar***. Bootstrap ClassLoader doesn’t have any parent ClassLoaders. It is also called as the **Primodial ClassLoader**.
2. **Extension ClassLoader:** The Extension ClassLoader is a child of Bootstrap ClassLoader and loads the extensions of core java classes from the respective JDK Extension library. It loads files from ***jre/lib/ext*** directory or any other directory pointed by the system property ***java.ext.dirs***.
3. **System ClassLoader:** An Application ClassLoader is also known as a System ClassLoader. It loads the Application type classes found in the environment variable ***CLASSPATH, -classpath or -cp command line option***. The Application ClassLoader is a child class of Extension ClassLoader.

How class Loader works:

When a class name is given, class loader first locates the class and then reads a class file of that name from the native file system. Therefore this loading process is platform dependent.

By default java.lang.ClassLoader is registered as a class loader that is capable of loading classes in parallel. But the subclasses needs to register as parallel or not at the time of instantiation.

Classes can also be loaded from network, constructed on runtime and loaded. ClassLoader class has a method name defineClass which takes input as byte array and loads a class.

Yes,same class be loaded twice

when a class is loaded into JVM, you have an entry as (package, classname, classloader). Therefore the same class can be loaded twice by two different ClassLoader instances.

**13. Explain the difference between Comparable and Comparator.**

1. Comparable interface can be used to provide single way of sorting whereas Comparator interface is used to provide different ways of sorting.
2. For using Comparable, Class needs to implement it whereas for using Comparator we don’t need to make any change in the class.
3. Comparable interface is in java.lang package whereas Comparator interface is present in java.util package.
4. We don’t need to make any code changes at client side for using Comparable, Arrays.sort() or Collection.sort() methods automatically uses the compareTo() method of the class. For Comparator, client needs to provide the Comparator class to use in compare() method.

**15. Explain the concept of constructor chaining.**

Constructor chaining is the process of calling one constructor from another constructor with respect to current object.  
Constructor chaining can be done in two ways:

* **Within same class**: It can be done using **this()** keyword for constructors in same class
* **From base class:**by using **super()** keyword to call constructor from the base class.

// Java program to illustrate Constructor Chaining

// within same class Using this() keyword

class Temp

{

// default constructor 1

// default constructor will call another constructor

// using this keyword from same class

Temp()

{

// calls constructor 2

this(5);

System.out.println("The Default constructor");

}

// parameterized constructor 2

Temp(int x)

{

// calls constructor 3

this(5, 15);

System.out.println(x);

}

// parameterized constructor 3

Temp(int x, int y)

{

System.out.println(x \* y);

}

public static void main(String args[])

{

// invokes default constructor first

new Temp();

}

}

**16. What is the difference between Iterator and ListIterator?**

1) Iterator is used for traversing List and Set both.

We can use ListIterator to traverse List only, we cannot traverse Set using ListIterator.

2) We can traverse in only forward direction using Iterator.

Using ListIterator, we can traverse a List in both the directions (forward and Backward).

3) We cannot obtain indexes while using Iterator

We can obtain indexes at any point of time while traversing a list using ListIterator. The methods nextIndex() and previousIndex() are used for this purpose.

4) We cannot add element to collection while traversing it using Iterator, it throws ConcurrentModificationException when you try to do it.

We can add element at any point of time while traversing a list using ListIterator.

5) We cannot replace the existing element value when using Iterator.

By using set(E e) method of ListIterator we can replace the last element returned by next() or previous() methods.

6) Methods of Iterator:

* hasNext()
* next()
* remove()

Methods of ListIterator:

* add(E e)
* hasNext()
* hasPrevious()
* next()
* nextIndex()
* previous()
* previousIndex()
* remove()
* set(E e)

**17. What is the difference between fail-fast and fail-safe iterator? Explain ConcurrentModificationException with an example.**

**FailSafe**

* Fail-safe iterators allow modifications of a collection while iterating over it.
* These iterators don’t throw any Exception if a collection is modified while iterating over it.
* They use copy of original collection to traverse over the elements of the collection.
* These iterators require extra memory for cloning of collection. Ex : ConcurrentHashMap, CopyOnWriteArrayList

FailFast

* These iterators throw ConcurrentModificationException if a collection is modified while iterating over it.
* They use original collection to traverse over the elements of the collection.
* These iterators don’t require extra memory.
* Ex : Iterators returned by ArrayList, Vector, HashMap.

public class FailFastExample {

public static void main(String[] args)

{

Map<String, String> cityCode = new HashMap<String, String>();

cityCode.put("Delhi", "India");

cityCode.put("Moscow", "Russia");

cityCode.put("New York", "USA");

Iterator iterator = cityCode.keySet().iterator();

while (iterator.hasNext()) {

System.out.println(cityCode.get(iterator.next()));

// adding an element to Map

// exception will be thrown on next call

// of next() method.

cityCode.put("Istanbul", "Turkey");

}

}

}

1. **What is reflection? Where is it used?**

Reflection is an API which is used to examine or modify the behavior of methods, classes, interfaces at runtime.

* The required classes for reflection are provided under java.lang.reflect package.
* Reflection gives us information about the class to which an object belongs and also the methods of that class which can be executed by using the object.
* Through reflection we can invoke methods at runtime irrespective of the access specifier used with them.

class Test

{

private String s;

public Test() { s = "GeeksforGeeks"; }

public void method1() {

System.out.println("The string is " + s);

}

public void method2(int n) {

System.out.println("The number is " + n);

}

private void method3() {

System.out.println("Private method invoked");

}

}

class Demo

{

public static void main(String args[]) throws Exception

{

Test obj = new Test();

Class cls = obj.getClass();

System.out.println("The name of class is " +

cls.getName());

Constructor constructor = cls.getConstructor();

System.out.println("The name of constructor is " +

constructor.getName());

System.out.println("The public methods of class are : ");

Method[] methods = cls.getMethods();

for (Method method:methods)

System.out.println(method.getName());

Method methodcall1 = cls.getDeclaredMethod("method2",

int.class);

methodcall1.invoke(obj, 19);

Field field = cls.getDeclaredField("s");

field.setAccessible(true);

field.set(obj, "JAVA");

Method methodcall2 = cls.getDeclaredMethod("method1");

methodcall2.invoke(obj);

Method methodcall3 = cls.getDeclaredMethod("method3");

methodcall3.setAccessible(true);

methodcall3.invoke(obj);

}

}

1. **What are the rules for overriding methods in Java?**

>**Overriding and Access-Modifiers :**The [access modifier](https://www.geeksforgeeks.org/access-modifiers-java/) for an overriding method can allow more, but not less, access than the overridden method. For example, a protected instance method in the super-class can be made public, but not private, in the subclass. Doing so, will generate compile-time error.

>Final methods can not be overridden : If we don’t want a method to be overridden, we declare it as [final](https://www.geeksforgeeks.org/final-keyword-java/).

>Static methods can not be overridden(Method Overriding vs Method Hiding) : When you defines a static method with same signature as a static method in base class, it is known as [method hiding](https://www.geeksforgeeks.org/can-we-overload-or-override-static-methods-in-java/).

>Private methods can not be overridden : [Private methods](https://www.geeksforgeeks.org/can-override-private-methods-java/)cannot be overridden as they are bonded during compile time. Therefore we can’t even override private methods in a subclass.

>**The overriding method must have same return type (or subtype) :**From Java 5.0 onwards it is possible to have different return type for a overriding method in child class, but child’s return type should be sub-type of parent’s return type. This phenomena is known as [covariant return type](https://www.geeksforgeeks.org/covariant-return-types-java/).

>**Invoking overridden method from sub-class :**We can call parent class method in overriding method using [super keyword](http://quiz.geeksforgeeks.org/super-keyword/).

>**Overriding and constructor :**We can not override constructor as parent and child class can never have constructor with same name(Constructor name must always be same as Class name)

1. **How to implement a Singleton in Java?**

public class GFG

{

private static GFG instance;

private GFG()

{

// private constructor

}

//method to return instance of class

public static GFG getInstance()

{

if (instance == null)

{

// if instance is null, initialize

instance = new GFG();

}

return instance;

}

}