**What is the difference between StringBuffer and String?**

String is an Immutable class, i.e. you cannot modify its content once created.

While StringBuffer is a mutable class, means we can change its content later.

Whenever we alter content of String object, it creates a new string and refer to that,it does not modify the existing one. This is the reason that the performance with StringBuffer is better than with String.

**Can a top level class be private or protected?**

Top level classes in java can’t be private or protected, but inner classes in java can.

The reason for not making a top level class as private is very obvious, because nobody can see a private class and thus they cannot use it.

Declaring a class as protected also doesn’t make any sense. The only difference between default visibility and protected visibility is that we can use it in any package by inheriting it.

Since in java there is no such concept of package inheritance, defining a class as protected is no different from default.

**What will happen if you put System.exit(0) on try or catch block? Will finally block execute?**  
   
By Calling System.exit(0) in try or catch block, we can skip the finally block. System.exit(int) method can throw a SecurityException.

If Sysytem.exit(0) exits the JVM without throwing that exception then finally block will not execute. But, if System.exit(0) does throw security exception then finally block will be executed.

**Can we** [**Overload or Override static methods in java**](https://www.geeksforgeeks.org/can-we-overload-or-override-static-methods-in-java/) **?**

* **Overriding:** Overriding is related to run-time polymorphism. A subclass (or derived class) provides a specific implementation of a method in superclass (or base class) at runtime.
* **Overloading:** Overloading is related to compile time (or static) polymorphism. This feature allows different methods to have same name, but different signatures, especially number of input parameters and type of input paramaters.
* **Can we overload static methods?**The answer is **‘Yes’**. We can have two or more static methods with same name, but differences in input parameters
* **Can we override static methods in java?**We can declare static methods with same signature in subclass, but it is not considered overriding as there won’t be any run-time polymorphism. Hence the answer is **‘No’**.
* Static methods cannot be overridden because method overriding only occurs in the context of dynamic (i.e. runtime) lookup of methods. Static methods (by their name) are looked up statically (i.e. at compile-time).

**What happens if you remove static modifier from the main method?**  
Program compiles successfully. But at runtime throws an error “NoSuchMethodError”.

**What is** [**“this” keyword in java**](http://quiz.geeksforgeeks.org/this-reference-in-java/)**?**  
Within an instance method or a constructor, this is a reference to the current object — the object whose method or constructor is being called.

We can refer to any member of the current object from within an instance method or a constructor by using this.  
Usage of this keyword

* Used to refer current class instance variable.
* To invoke current class constructor.
* It can be passed as an argument in the method call.
* It can be passed as argument in the constructor call.
* Used to return the current class instance.
* Used to invoke current class method (implicitly)

**What is** [**object cloning**](http://quiz.geeksforgeeks.org/cloning-in-java/)**?**

Object cloning means to create an exact copy of the original object. If a class needs to support cloning, it must implement java.lang.Cloneable interface and override clone() method from Object class.

Syntax of the clone() method is :

protected Object clone() throws CloneNotSupportedException

If the object’s class doesn’t implement Cloneable interface then it throws an exception ‘CloneNotSupportedException’ .

**Different ways to create objects in Java**

There are many different ways to create objects in Java.  
**Following are some ways in which you can create objects in Java:**

1. **Using new Keyword :**

Using new keyword is the most basic way to create an object. This is the most common way to create an object in java.

Almost 99% of objects are created in this way.

By using this method we can call any constructor we want to call (no argument or parameterized constructors).

1. **Using** [**New Instance**](https://www.geeksforgeeks.org/new-operator-vs-newinstance-method-java/) **:**

If we know the name of the class & if it has a public default constructor we can create an object –**Class.forName**.

We can use it to create the Object of a Class. Class.forName actually loads the Class in Java but doesn’t create any Object.

To Create an Object of the Class you have to use the new Instance Method of the Class.

**Class cls = Class.forName("NewInstanceExample");**

**NewInstanceExample obj =**

**(NewInstanceExample) cls.newInstance();**

**System.out.println(obj.name);**

1. **Using** [**clone() method:**](https://www.geeksforgeeks.org/clone-method-in-java-2/)

Whenever clone() is called on any object, the JVM actually creates a new object and copies all content of the previous object into it.

Creating an object using the clone method does not invoke any constructor.  
To use clone() method on an object we need to implement **Cloneable** and define the clone() method in it.

**public** **class** CloneExample **implements** Cloneable {

@Override

**protected** Object clone() **throws** CloneNotSupportedException {

**return** **super**.clone();

}

String name = "GeeksForGeeks";

**public** **static** **void** main(String[] args) {

CloneExample obj1 = **new** CloneExample();

**try** {

CloneExample obj2 = (CloneExample) obj1.clone();

System.***out***.println(obj2.name);

} **catch** (CloneNotSupportedException e) {

e.printStackTrace();

}

}

}

**Note :**

* Here we are creating the clone of an existing Object and not any new Object.
* Class need to implement Cloneable Interface otherwise it will throw **CloneNotSupportedException**.

1. **Using** [**deserialization**](https://www.geeksforgeeks.org/serialization-in-java/) **:**

Whenever we serialize and then deserialize an object, JVM creates a separate object.

In **deserialization**, JVM doesn’t use any constructor to create the object.  
To deserialize an object we need to implement the Serializable interface in the class.

**Serializing an Object:**

**class** DeserializationExample **implements** Serializable {

**private** String name;

DeserializationExample(String name) {

**this**.name = name;

}

**public** **static** **void** main(String[] args) {

**try** {

DeserializationExample d = **new** DeserializationExample("GeeksForGeeks");

FileOutputStream f = **new** FileOutputStream("file.txt");

ObjectOutputStream oos = **new** ObjectOutputStream(f);

oos.writeObject(d);

oos.close();

f.close();

} **catch** (Exception e) {

e.printStackTrace();

}

}

}

Object of DeserializationExample class is serialized using writeObject() method and written to file.txt file.

**Deserialization of Object :**

**public** **class** DeserializationExample {

**public** **static** **void** main(String[] args) {

**try** {

DeserializationExample d;

FileInputStream f = **new** FileInputStream("file.txt");

ObjectInputStream oos = **new** ObjectInputStream(f);

d = (DeserializationExample) oos.readObject();

} **catch** (Exception e) {

e.printStackTrace();

}

System.***out***.println(d.name);

}

}

**5) Using newInstance() method of Constructor class :**

This is similar to the newInstance() method of a class.

There is one newInstance() method in the **java.lang.reflect.Constructor** class which we can use to create objects. It can also call parameterized constructor, and private constructor by using this newInstance() method.

Both newInstance() methods are known as reflective ways to create objects. In fact newInstance() method of Class internally uses newInstance() method of Constructor class.

**public** **class** ReflectionExample {

**private** String name;

**public** **static** **void** main(String[] args) {

**try** {

Constructor<ReflectionExample> constructor = ReflectionExample.**class**.getDeclaredConstructor();

ReflectionExample r = constructor.newInstance();

r.setName("GeeksForGeeks");

System.***out***.println(r.name);

} **catch** (Exception e) {

e.printStackTrace();

}

}

}

**Q5. Define class and object. Explain them with an example using java.**

**Class:** A class is a user defined blueprint or prototype from which objects are created.  It represents the set of properties or methods that are common to all objects of one type. In general, class declarations can include these components, in order:

**Superclass(if any):** The name of the class’s parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.

**Interfaces:** A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.

**Object:** It is a basic unit of Object Oriented Programming and represents the real life entities.  A typical Java program creates many objects, which as you know, interact by invoking methods.

An object consists of:

**State :** It is represented by attributes of an object. It also reflect the properties of an object.

**Behavior :** It is represented by methods of an object. It also reflects the response of an object with other objects.

**Identity :** It gives a unique name to an object and enables one object to interact with other objects.  
For Example: Employee is an example of a class  
A specific employee with unique identification is an example of an object.

**class** Employee {

// instance variables declaration

// Methods definition

}

An object of employee is a specific employee

Employee empObj = new Employee();

One of the objects of Employee is referred by ‘empObj