1. **What do you mean by String class is immutable? Explain with the example.**

Ans:

String s1=new String(“hello”);

System.out.println(s1);

Will pring “hello”

Now if we perform

S1.concat(“ world”);

And print

System.out.println(s1);

Will still print “hello”

It is because whenever we perform any operation on an instance of String class , it does not modify the current instance rather it creates a new instance with the modifications. This is how we can say the String class is immutable class in java.

1. **Why String class is immutable?**

Ans:

String class is immutable because in case of String class it’s possible that more than one references of String class may share same instance ( on normal heap or string pool). In this scenario it should not happen that because of one reference (if we try to modify it) ,other references should get affected. Hence String class is immutable in java.

1. **Explain “intern()” method of String class.**

Ans:

When the intern method is invoked, if the pool already contains a string equal to this String object as determined by the equals(Object) method, then the string from the pool is returned.

Otherwise, this String object is added to the pool and a reference to this String object is returned

1. **What is the importance of “intern()” method of String class?**

Ans:

Using intern() can improve performance in situations where string comparisons are frequent. Interned strings can be compared using == (reference equality) instead of equals(), which can be faster since it avoids the overhead of comparing character by character.

1. **String s1="hello";**

**What happens in case of above statement?**

Ans:

If a String object containing "hello" is not already present in the String Pool, the JVM creates a new String object with "hello" in the String Pool and “s1” will refer to that object.

If the **String Pool** already has String object "hello", no new object is created, “s1” will refer to the same object.

1. **String s1="hello";**

**s1=null;**

**Assuming only "s1" refers to the String object inside string pool. After making it null, will the String object from the string pool be garbage collected as it has now become unreferenced?**

Ans:

String s1 = "hello";

* When the above line executes:
  1. The JVM **checks the String Pool** (inside Heap) to see if "hello" already exists.
  2. If "hello" **does not exist**, the JVM **creates a new String object** in the **String Pool** (which is part of the Heap).
  3. A **reference to this String object** is stored inside the **Constant Pool (Metaspace)**.
  4. s1 (which is stored on the Stack) **points to the String object in the String Pool**.

**Step 2: Making s1 Null**

s1 = null;

* Now, the stack-based reference s1 **no longer points to the "hello" object** in the String Pool.
* **Does the String object inside the String Pool get garbage collected?** **No!**
* **Why?**
  + The **JVM keeps a strong reference to the "hello" String object inside the Constant Pool** (Metaspace).
  + This ensures that even if all stack references are lost, the object remains **accessible** through the Constant Pool.

**Step 3: Can the String Object in the Pool Be Garbage Collected?**

**Normal garbage collection does not remove String objects from the String Pool** because they are **still referenced from the Constant Pool**.

* However, **if the class that loaded the String is unloaded (via a custom class loader), then the associated Constant Pool is removed, and the String Pool objects that are only referenced by that Constant Pool can be garbage collected**.

Even though s1 is null, the "hello" object **is not garbage collected** because the **Constant Pool still holds a reference to the object** .

1. **Given**

**String str=new String("hello");**

**if this is the very first statement creating string, then what happens in the memory?**

ans:

JVM checks whether String object with "hello" is there or not in the string pool. Since it's a very first statement, string object is not there in the string pool. String object containing "hello" is created in the string pool and the reference to that will be stored in the constant pool. Thereafter because we have used "new" keyword, a separate object of String class containing "hello" will be created in normal heap and the reference "str" which is on stack will refer to this normal heap object.

Final Memory Layout

**Metaspace** (Constant Pool)

Contains a reference to the "hello" object in the String Pool (Heap).

**Heap Memory**

String Pool contains one "hello" object.

Normal Heap contains a new "hello" object, created by new String("hello").

**Stack Memory**

The variable str is stored on the stack, and it refers to the Heap object created by new String("hello").

1. **How exactly protected member is accessed outside the package?**

Ans:

Protected members are “**accessible outside the package only through inheritance**“. i.e you can access a protected member of a class in its subclass present in some other package directly as if the member is present in the subclass itself. But that protected member will not be accessible in the subclass outside the package by using parent class’s reference.

1. **Why wrapper classes are introduced in java?**

Ans:

earlier versions of java:

public class Demo

{

static void show(Object ref)

{

S.o.p(ref);

}

main()

{

show(new String("hello")); // alld

show(new ArrayList()); // alld

show(new LinkedList()); // alld

int num=10;

show(num); // was not possible

}

}

Solution: Wrapper classes were introduced

above program can be written as:

public class Demo

{

static void show(Object ref)

{

S.o.p(ref);

}

main()

{

show(new String("hello")); // alld

show(new ArrayList()); // alld

show(new LinkedList()); // alld

int num=10;

// show(num); // was not possible

Integer ob=new Integer(num);

show(ob); // this is possible

}

}

1. **How many types of nested classes are there in java?**

Ans:

Nested classes are divided into two categories: static and non-static.

Nested classes that are declared static are called static nested classes.

Non-static nested classes are called inner classes.

classes which are defined inside any method are known as "Local Inner Classes".

1. **Given:**

**class Outer**

**{**

**class inner**

**{**

**}**

**}**

**Outer.inner i=o1.new inner();**

**What exactly compiler does in the above code?**

ans:

when we say:

Outer.inner i=o1.new inner();

i.e. when inner class gets instantiated, compiler puts a reference of "Outer" class inside inner class object. That reference refers to the object where "o1" refers to.

Why compiler does that?

because even though they are "outer" and "inner" classes for a developer, for JVM these two are different classes altogether.

How compiler does that?

compiler adds a parameterized constructor accepting "outer" class reference inside "inner" class and discards all the other available constructors.

1. **How do we instantiate inner class and static nested class?**

Ans:

In order to instantiate inner class we must use an instance of outer class whereas static nested class can be instantiated without the instance of outer class.

1. **What is the exact use of enum in java?**

Ans:

enum is a user defined data type.

it is used to define set of predefined values. It helps in making the program more readable and also helps to reduce programming bugs.

some of the examples where enum is used in java are:

compass directions (values of NORTH, SOUTH, EAST, and WEST)

The days of the week. MONDAY,TUESDAY etc.

sizes in case of Pizza

SMALL , MEDIUM,LARGE etc.

1. **How enums are different from the static constants?**

Ans:

An **enum (enumeration)** in Java is a **special class** that represents a **fixed set of constants**. It is used when a **variable should be assigned only predefined values**.

**Why Use enum Instead of Constants?**

**Before enum (Using static final constants):**

class Status {

public static final int SUCCESS = 1;

public static final int FAILURE = 0;

}

public class Main {

public static void main(String[] args) {

int status = Status.SUCCESS;

System.out.println(status); // 1

status = 3; // Invalid, but compiles fine!

}

}

**Problem:**

* The variable **can accept invalid values (e.g., 3)** since it's just an integer.
* **No type safety** – any integer can be assigned.
* **No methods/logic** can be attached to constants.

**Solution: Use enum!**

enum Status {

SUCCESS, FAILURE;

}

public class Main {

public static void main(String[] args) {

Status status = Status.SUCCESS;

System.out.println(status); // SUCCESS

// status = 3; // Compile-time error!

}

}

**Type safety**: Only predefined values are allowed.  
**Self-documenting**: Improves readability.

1. **What do you mean by autoboxing and unboxing are only at compiler level?**

Ans:

int num=10;

Integer ob=num; // Autoboxing

i.e. Autoboxing means assigning primitive to wrapper

here when we do autoboxing i.e.

Integer ob=num;

what compiler does is, compiler converts above statement into the statement which we used to write before jdk5

i.e.

Integer ob=new Integer(num);

why compiler does this?

it's because JVM doesn't understand autoboxing.

So we can say that autoboxing is only at compiler level. It is just a syntactical sugar for a developer.

Before JDK1.5

converting Wrapper to primitive

int num=10;

Integer ob=new Integer(num);

int temp=ob.intValue();

JDK1.5 onwards

int num=10;

Integer ob=num; // Autoboxing

int temp=ob; // unboxing

i.e. unboxing means assigning wrapper to primitive

here when we do unboxing i.e.

int temp=ob

what compiler does is, compiler converts above statement into the statement which we used to write before jdk5

i.e.

int temp=ob.intValue();

why compiler does this?

it's because JVM doesn't understand unboxing.

So we can say that unboxing is only at compiler level. It is just a syntactical sugar for a developer.

1. **What care we should take while defining a task based method ( performing some action on the object ) while designing immutable class?**

Ans:

Inside immutable class when you define any task based method ( performing some action on the object ) make sure it will not perform the task on an invoking or current object , rather it should create a new object, perform the task and return it.

1. **What care we should take while defining a task based method ( performing some action on the object ) while designing mutable class?**

Ans:

Inside mutable class when you define any task based method ( performing some action on the object ) make sure it will perform the task on an invoking or current object and return it.