1. What will be the output of the below program?

```
public class StrEqual {
    public static void main(String[] args) {
        String s1 = "hello";
        String s2 = new String("hello");
        String s3 = "hello";
        if (s1 == s2) {
            System.out.println("s1 and s2 equal");
        } else {
            System.out.println("s1 and s2 not equal");
        }
        if (s1 == s3) {
            System.out.println("s1 and s3 equal");
        } else {
            System.out.println("s1 and s3 not equal");
        }
    }
}
```

Answer

```
s1 and s2 not equal
s1 and s3 equal
```

JVM sets a constant pool in which it stores all the string constants used in the type. If two references are declared with a constant, then both refer to the same constant object.

The == operator checks the similarity of objects themselves (and not the values in it). Here, the first comparison is between two distinct objects, so we get s1 and s2 not equal. On the other hand, since references to s1 and s3 refer to the same object, we get s1 and s3 equal.

2. What will be the output of the below program?

```
public class Test {
    public static void main(String[] args) {
        String s = new String("5");
        System.out.println(1 + 10 + s + 1 + 10);
    }
}
```

Answer

115110

The string concatenation operator works as follows: if both the operands are numbers, it performs the addition; otherwise, it concatenates the arguments by calling the `toString()` method if needed. It evaluates from left to right. Hence, the expression in the program results in the string **115110**.

3. What will be the output of the below program?

```
public class Test {
    public static void main(String[] args) {
        String str = null;
        System.out.println(str.valueOf(10));
    }
}
```

```
This program will print 10 in the console.
```

Explanation: The `valueOf(int)` method is a static method in String that returns the String representation of the integer value that is passed as its argument.

Since calling a static method does not require dereferencing the reference variable on which it is called, this program does not throw a `NullPointerException` .

4. What will be the output of the below statements?

```
public class Test {

    public static void main(String[] args) {
        String s1 = "abc";
        StringBuffer s2 = new StringBuffer(s1);
        System.out.println(s1.equals(s2));
    }
}
```

Answer

```
false
```

It will print false because **s2** is not of type String. If you will look at the equals method implementation in the String class, you will find a check using `instanceof` operator to check if the type of passed object is String? If not, then return `false`.

5. What is the output of the below program?

```
public class A {
    public static void main(String[] args) {
        String s1 = new String("javaguides");
        String s2 = new String("javaguides");
        System.out.println(s1 = s2);
    }
}
```

The above program prints “javaguides” because we are assigning s2 String to s1. Don’t get confused with == comparison operator.

6. What is the output of the following program?

```
public class Test {

    public static void main(String[] args) {
        String s1 = "hello";
        String s2 = new String("hello");

        s2 = s2.intern();
        System.out.println(s1 == s2);
    }
}
```

Answer

```
true
```

We know that the `intern()` method will return the String object reference from the string pool since we assign it back to **s2** and now both **s1** and **s2** are having the same reference. It means that **s1** and **s2** references point to the same object.

7. How many objects will be created in the following code and where they will be stored in the memory?

```
String s1 = "javaguides";

String s2 = "javaguides";
```

Answer

Only one object will be created and this object will be stored in the string constant pool.

8. How many objects will be created in the following code and where they will be stored?

```
String s1 = new String("javaguides");

String s2 = "javaguides";
```

Answer

Here, two string objects will be created. An object created using a new operator(s1) will be stored in the heap memory. The object created using a string literal(s2) is stored in the string constant pool.

9. What is the output of the below code snippet?

```
String s1 = new String("javaguides");
String s2 = new String("javaguides");
System.out.println(s1 == s2);
```

The above code snippet prints output as `false` because we are using a **new** operator to create String, so it will be created in the heap memory and both s1, s2 will have different references.

If we create them using double quotes, then they will be part of the string pool and it will print true.

10. What is the output of the below code snippet?

```
String s1 = "javaguides"
String s2 = "javaguides";
System.out.println(s1 == s2);
```

The above code snippet prints output as `true` because we have created both String objects using double quotes (Sting literals), then they will be part of the string pool and it will print true.