Worksheet - 5

Q1.//Stringbuffer public class Main { public static void main(String args[]) { String s1 = "abc"; String s2 = s1; s1 += "d"; System.out.println(s1 + " " + s2 + " " + (s1 == s2)); StringBuffer sb1 = new StringBuffer("abc"); StringBuffer sb2 = sb1; sb1.append("d"); System.out.println(sb1 + " " + sb2 + " " + (sb1 == sb2)); } }

String s1 = “abc” ; creates a string s1 with value “abc”,
String s2 = s1; assigns s1 to s2 , so both s1 and s2 ref the same string “abc”
s1 += “d”; creates a new string “abcd” and assigns it to s1. Now, s1 points to “abcd”, but s2 still points to “abc” because the string is immutable.
The output for System.out.println(s1 + " " + s2 + " " + (s1 == s2)); is `(s1==s2)` is `false` because `s1` and `s2` now reference different objects.

StringBuffer sb1 = new StringBuffer(“abc”); creates a StringBuffer object sb1 with the value “abc”.
StringBuffer sb2 = new StringBuffer(“abc”); creates a StringBuffer object sb2 with the value “abc”.
The output for System.out.println(sb1 + “ “ + sb2 + “ “ + “ “ + (sb1 == sb2);
sb1 is “abcd”.
sb2 is “abcd”.(because both `sb1` and `sb2` reference the same `StringBuffer` object).`(sb1==sb2)` is `true` because `sb1` and `sb2` reference the same object.

So the output is **abcd abc false**
abcd abcd true

Q2.// Method overloading public class Main { public static void FlipRobo(String s) { System.out.println("String"); } public static void FlipRobo(Object o) { System.out.println("Object"); } public static void main(String args[]) { FlipRobo(null); } }

There are two overloaded methods named FlipRobo:

1. FlipRobo (String s) - Accepts a String as a parameter.
2. FlipRobo (Object o) - Accepts an Object as a parameter.

In the `main` method, the `FlipRobo(null)` call is ambiguous because `null` could be assigned to either a `String` or an `Object`.
In method overloading, when null is passed as an argument, Java picks the most specific method. In this case, the String type is more specific than Object because String is a subclass of Object.

The method FlipRobo(String s) will be invoked because String is more specific than Object.
So the output is **String.**

Q3. class First { public First() { System.out.println("a"); } } class Second extends First { public Second() { System.out.println("b"); } } class Third extends Second { public Third() { System.out.println("c"); } } public class MainClass { public static void main(String[] args) { Third c = new Third(); } }

Class Hierarchy: First is the superclass, Second extends First, Third extends Second.
When an object of Third class is created (Third c = new Third();), the constructor of Third is invoked.
Since Third extends Second, the constructor of Second is automatically called before the Third constructor.
Similarly, because Second extends First, the constructor of First is called before the Second constructor.
The First class constructor is executed first, printing "a".
The Second class constructor is executed next, printing "b".
Finally, the Third class constructor is executed, printing "c".
So the output is **a**
b
c

Q4.public class Calculator { int num = 100; public void calc(int num) { this.num = num * 10; } public void printNum() { System.out.println(num); } public static void main(String[] args) { Calculator obj = new Calculator(); obj.calc(2); obj.printNum(); } }

The class Calculator has an instance variable num initialized to 100.
This method takes an integer parameter num.
Inside the method, this.num refers to the instance variable num, and it is assigned the value of num * 10.
This keyword is used here to differentiate between the method's parameter num and the instance variable num.
This method prints the current value of the instance variable num.
An object of Calculator is created: Calculator obj = new Calculator();.
The calc(2) method is called, which sets this.num (the instance variable) to 2 * 10 = 20.
The printNum() method is then called, which prints the updated value of num.
So the output is **20.**

Q5.public class Test { public static void main(String[] args) { StringBuilder s1 = new StringBuilder("Java"); String s2 = "Love"; s1.append(s2); s1.substring(4); int foundAt = s1.indexOf(s2); System.out.println(foundAt); } }

StringBuilder Initialization:
- StringBuilder s1 = new StringBuilder("Java");
- A StringBuilder object s1 is created with the content "Java".

String Initialization:
- String s2 = "Love";
- A String object s2 is created with the content "Love".

Appending s2 to s1:
- s1.append(s2);
- The append method adds the string s2 ("Love") to the end of s1.
- Now, s1 contains "JavaLove".

Using substring Method:
- s1.substring(4);
- The substring method is called on s1, starting at index 4. However, the result of this method is not stored in any variable, so it doesn't alter s1.
- s1 still contains "JavaLove".

Finding the Index of s2 in s1:
- int foundAt = s1.indexOf(s2);
- The indexOf method returns the index where the substring s2 ("Love") first appears in s1 ("JavaLove").
- Since "Love" starts at index 4 in "JavaLove", foundAt will be 4.

So the output is **4.**

Q6. class Writer { public static void write() { System.out.println("Writing..."); } } class Author extends Writer { public static void write() { System.out.println("Writing book"); } } public class Programmer extends Author { public static void write() { System.out.println("Writing code"); } public static void main(String[] args) { Author a = new Programmer(); a.write(); } }

Author a = new Programmer();: a is a reference of type Author, but it is pointing to an object of type Programmer.
Since write() is a static method, it is bound at compile time. The method that gets called depends on the reference type (Author in this case), not the object type (Programmer).
So the output is **Writing book.**

Q7.class FlipRobo { public static void main(String args[]) { String s1 = new String("FlipRobo"); String s2 = new String("FlipRobo"); if (s1 == s2) System.out.println("Equal"); else System.out.println("Not equal"); } }

`s1` and `s2` are two different `String` objects created using the `new` keyword.
The `==` operator compares references, not the content of the strings. Since `s1` and `s2` are two different objects, they have different memory addresses.
So the output is **Not equal.**

Q8.class FlipRobo { public static void main(String args[]) { try { System.out.println("First statement of try block"); int num=45/3; System.out.println(num); } catch(Exception e) { System.out.println("FlipRobo caught Exception"); } finally { System.out.println("finally block"); } System.out.println("Main method"); } }

Try block: The first statement inside the try block print First statement of try block.Then, the integer num is calculated by dividing 45 by 3, which results in 15. This value is printed
Catch block: The catch block is designed to handle exceptions. However, since no exception is thrown in the try block (as the division 45/3 is valid), this block is not executed.
Finally: The finally block is always executed, whether an exception is caught or not. Therefore, "finally block" will be printed.
After the try-catch-finally blocks, the statement "Main method" will be printed.
So the output is **First statement of try block**
15
finally block
Main method

Q9.class FlipRobo { // constructor FlipRobo() { System.out.println("constructor called"); } static FlipRobo a = new FlipRobo(); //line 8 public static void main(String args[]) { FlipRobo b; //line 12 b = new FlipRobo(); } }

Static Field Initialization (line 8):

- static FlipRobo a = new FlipRobo(); is a static field initialization.
- When the class is loaded, the static fields are initialized first, and the static block is executed before the main method.
- Therefore, as soon as the class FlipRobo is loaded, the constructor FlipRobo() is called for the static field a, and "constructor called" is printed.

Main Method:

- The main method starts executing.
- At line 12, the reference variable b is declared but not initialized yet.
- Then, b = new FlipRobo(); creates a new instance of the FlipRobo class, which calls the constructor again, printing "constructor called"

So the output is **constructor called**
constructor called

Q10.class FlipRobo { static int num; static String mystr; // constructor FlipRobo() { num = 100; mystr = "Constructor"; } // First Static block static { System.out.println("Static Block 1"); num = 68; mystr = "Block1"; } // Second static block static { System.out.println("Static Block 2"); num = 98; mystr = "Block2"; } public static void main(String args[]) { FlipRobo a = new FlipRobo(); System.out.println("Value of num = " + a.num); System.out.println("Value of mystr = " + a.mystr); } }

Static blocks are executed when the class is loaded, before any instances of the class are created and before the main method is executed.
First Static Block:

- Prints "Static Block 1".
- Sets num to 68.
- Sets mystr to "Block1".

Second Static Block:

- Prints "Static Block 2".
- Sets num to 98.
- Sets mystr to "Block2".

The main method creates an instance of the FlipRobo class with FlipRobo a = new FlipRobo();.
The constructor is called, which sets num to 100 and mystr to "Constructor".
The main method then prints the values of num and mystr after the constructor has run.
So the output is **Static Block 1**
Static Block 2
Value of num = 100
Value of mystr = constructor