FULL STACK DEVELOPMENT – WORKSHEET 5

FIND OUTPUT OF THE PROGRAMS WITH EXPLANATION

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));

 }

 }
```

OutPut:-

```
java -cp /tmp/8NKwsOphYK Main

abcd abc false

abcd abcd true
```

Explanation : In Java, String is immutable and string buffer is mutable. So string s2 and s1 both pointing to the same string abc. And, after making the changes the string s1 points to abcd and s2 points to abc, hence false. While in string buffer, both sb1 and sb2 both point to the same object. Since string buffer are mutable, making changes in one string also make changes to the other string. So both string still pointing to the same object after making the changes to the object (here sb2).

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);

}

}
```

OutPut:-

```
java -cp /tmp/8NKwsOphYK Main

String
```

Explanation : In case of method overloading, the most specific method is chosen at compile time. As ‘java.lang.String’ is a more specific type than ‘java.lang.Object’. In this case the method which takes ‘String’ as a parameter is chosen.

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();
}
}
```

Output:

```
ERROR!
javac /tmp/8NKwsOphYK/Third.java
/tmp/8NKwsOphYK/Third.java:15: error: class MainClass is public, should be declared in a file named MainClass.java
public class MainClass
^
1 error
```

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();
}

}
```

Output:-

```
java -cp /tmp/8NKwsOphYK Calculator

20
```

Explanation : Here the class instance variable name(num) is same as *calc()* method local variable name(num). So for referencing class instance variable from *calc()* method, this keyword is used. So in statement **this.num = num * 10**, *num* represents local variable of the method whose value is 2 and *this.num* represents class instance variable whose initial value is 100. Now in *printNum()* method, as it has no local variable whose name is same as class instance variable, so we can directly use *num* to reference instance variable, although *this.num* can be used.

```
// Online Java Compiler

// Use this editor to write, compile and run your Java code online

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);

}

}
```

Output:-

```
java -cp /tmp/8NKwsOphYK Test

4
```

Explanation : *append(String str)* method,concatenate the str to *s1*. The *substring(int index)* method return the String from the given index to the end. But as there is no any String variable to store the returned string,so it will be destroyed.Now *indexOf(String s2)* method return the index of first occurrence of *s2*. So 4 is printed as s1=”JavaLove”.

```
// Online Java Compiler

// Use this editor to write, compile and run your Java code online
```

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();
}
}
```

Output

```
ERROR!

javac /tmp/DMHFmS7q70/Author.java

/tmp/DMHFmS7q70/Author.java:18: error: class Programmer is public, should be declared in a file named Programmer.java

public class Programmer extends Author

       ^

1 error
```

Explanation : Since static methods can't be overridden, it doesn't matter which class object is created. As *a* is a *Author* referenced type, so always *Author* class method is called. If we remove *write()* method from *Author* class then *Writer* class method is called, as *Author* class extends *Writer* class.

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");
}

}

java -cp /tmp/DMHFmS7q70 FlipRobo

Not equal
```

Explanation: Since, s1 and s2 are two different objects the references are not the same, and the == operator compares object reference. So it prints “Not equal”, to compare the actual characters in the string .equals() method must be used. 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");

}

}
```

Output:-

```
java -cp /tmp/DMHFmS7q70 FlipRobo


First statement of try block

15

finally block

Main method
```

Explanation:

Since there is no exception, the catch block is not called, but the finally block is always executed after a try block whether the exception is handled or not.

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();

}

}
```

Output:-

```
java -cp /tmp/DMHFmS7q70 FlipRobo


constructor called

constructor called
```

Explanation:
We know that static variables are called when a class loads and static variables are called only once. Now line 13 results to creation of object which inturn calls the constructor and "constructor called

" is printed second time.
If in line 8 static variable would not have been used the object would have been called recursively infinitely leading to StackOverFlow error. See this for a sample run.

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);
}
}
```

Output:-

```
java -cp /tmp/DMHFmS7q70 FlipRobo


Static Block 1Static Block 2Value of num = 100
Value of mystr = Constructor
```

Explanation:
Static block gets executed when the class is loaded in the memory. A class can have multiple Static blocks, which are executed in the same sequence in which they have been written into the program.