Department of Computer Science

Final Term Exam Summer 2020

Subject: Object Oriented Programming

BS (CS,SE) Instructor: M.Ayub Khan

There are total **5** questions in this paper. Max Marks: 50

Note:

At the top of the answer sheet there must be the ID, Name and semester of the concerned Student.

Students must have to provide the output of their respective programs. Students have same answers or programs will be considered fail. Programs in Java or codes should be explained clearly.

As this paper is online so incase of any ambiguity my Whatsapp no. is 03449121116.

Name : Shayan Khan ID : 6833

Subject : OOP Submitted To : Sir Ayub

Class : BS(SE)

Each question carry equal marks.
Please answer briefly.

Q1. How many variables are being supported by java justify your answer with the help java coded example for each variable?

Answer:

There are three kinds of variables in Java:

- Local variables
- Instance variables
- Class/Static variables

Local Variables

- Local variables are declared in methods, constructors, or blocks.
- Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor, or block.
- Access modifiers cannot be used for local variables.
- Local variables are visible only within the declared method, constructor, or block.
- Local variables are implemented at stack level internally.
- There is no default value for local variables, so local variables should be declared and an initial value should be assigned before the first use.

Example

Here, *age* is a local variable. This is defined inside *pupAge()* method and its scope is limited to only this method.

```
public class Test {
   public void pupAge() {
      int age = 0;
      age = age + 7;
      System.out.println("Puppy age is : " + age);
   }

   public static void main(String args[]) {
      Test test = new Test();
      test.pupAge();
   }
}
```

This will produce the following result −

Output

```
Puppy age is: 7
```

Instance Variables

- Instance variables are declared in a class, but outside a method, constructor or any block.
- When a space is allocated for an object in the heap, a slot for each instance variable value is created.
- Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.
- Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object's state that must be present throughout the class.
- Instance variables can be declared in class level before or after use.
- Access modifiers can be given for instance variables.
- The instance variables are visible for all methods, constructors and block in the class. Normally, it is recommended to make these variables private (access level). However, visibility for subclasses can be given for these variables with the use of access modifiers.
- Instance variables have default values. For numbers, the default value is 0, for Booleans it is false, and for object references it is null. Values can be assigned during the declaration or within the constructor.
- Instance variables can be accessed directly by calling the variable name inside the class. However, within static methods (when instance variables are given accessibility), they should be called using the fully qualified name. *ObjectReference.VariableName*.

Example

```
import java.io.*;
public class Employee {

   // this instance variable is visible for any child class.
   public String name;

   // salary  variable is visible in Employee class only.
   private double salary;

   // The name variable is assigned in the constructor.
   public Employee (String empName) {
      name = empName;
   }
```

```
   // The salary variable is assigned a value.
   public void setSalary(double empSal) {
      salary = empSal;
   }

   // This method prints the employee details.
   public void printEmp() {
      System.out.println("name  : " + name );
      System.out.println("salary :" + salary);
   }

   public static void main(String args[]) {
      Employee empOne = new Employee("Ransika");
      empOne.setSalary(1000);
      empOne.printEmp();
   }
}
```

This will produce the following result −

Output

```
name  : Ransika
salary :1000.0
```

Class/Static Variables

- Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block.
- There would only be one copy of each class variable per class, regardless of how many objects are created from it.
- Static variables are rarely used other than being declared as constants. Constants are variables that are declared as public/private, final, and static. Constant variables never change from their initial value.
- Static variables are stored in the static memory. It is rare to use static variables other than declared final and used as either public or private constants.
- Static variables are created when the program starts and destroyed when the program stops.
- Visibility is similar to instance variables. However, most static variables are declared public since they must be available for users of the class. Default values are same as instance variables. For numbers, the default value is 0; for Booleans, it is false; and for object references, it is null. Values can be assigned

during the declaration or within the constructor. Additionally, values can be assigned in special static initializer blocks.

- Static variables can be accessed by calling with the class name *ClassName.VariableName*.
- When declaring class variables as public static final, then variable names (constants) are all in upper case. If the static variables are not public and final, the naming syntax is the same as instance and local variables.

Example

```
import java.io.*;
public class Employee {

   // salary  variable is a private static variable
   private static double salary;

   // DEPARTMENT is a constant
   public static final String DEPARTMENT = "Development ";

   public static void main(String args[]) {
      salary = 1000;
      System.out.println(DEPARTMENT + "average salary:" + salary);
   }
}
```

This will produce the following result −

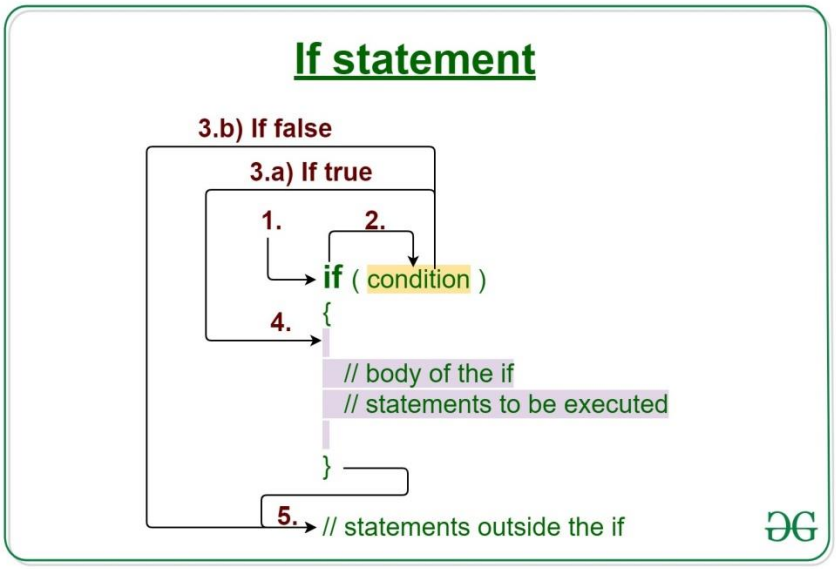
Output

```
Development average salary:1000
```

Q2. Why "If" is used in java justify your answer with the help java coded example and explain in detail?

Answer:

The **Java if statement** is the most simple decision-making statement. It is used to decide whether a certain statement or block of statements will be executed or not i.e if a certain condition is true then a block of statement is executed otherwise not.



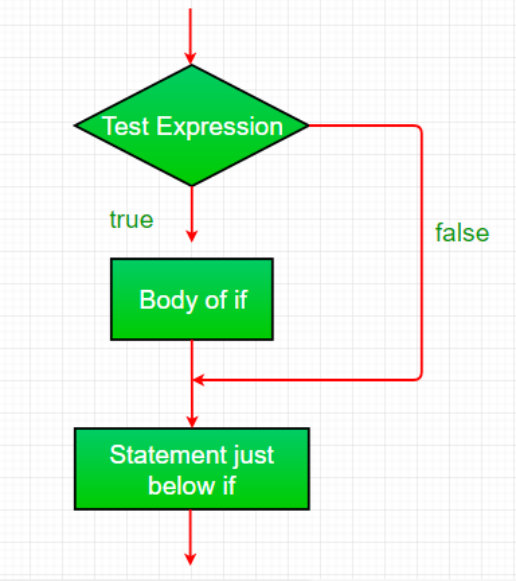
Syntax:

```
if(condition)
{
   // Statements to execute if
   // condition is true
}
```

Working of if statement

1. Control falls into the if block.
2. The flow jumps to Condition.
3. Condition is tested.
   - If Condition yields true, goto Step 4.
   - If Condition yields false, goto Step 5.
4. The if-block or the body inside the if is executed.
5. Flow steps out of the if block.

Flowchart if statement:



Operation:
The condition after evaluation of if-statement will be either true or false. The if statement in Java accepts boolean values and if the value is true then it will execute the block of statements under it.

Note: If we do not provide the curly braces '{' and '}' after if(condition) then by default if statement will consider the immediate one statement to be inside its block. For example,

```
if(condition)
   statement1;
   statement2;


// Here if the condition is true, if block
// will consider only statement1 to be inside
// its block.
```

Example 1:

```
// Java program to illustrate If statement

class IfDemo {
    public static void main(String args[])
    {
        int i = 10;

        if (i < 15)
            System.out.println("10 is less than 15");
```

```
        // This statement will be executed
        // as if considers one statement by default
        System.out.println("Outside if-block");
    }
}
```

Output:

```
10 is less than 15
Outside if-block
```

Q3. Why "if else if" is used in java justify your answer with the help java coded example and explain in detail?

Answer:

if-else-if Statement

if-else-if statement is used when we need to check multiple conditions. In this statement we have only one "if" and one "else", however we can have multiple "else if". It is also known as **if else if ladder**.

Note: The most important point to note here is that in if-else-if statement, as soon as the condition is met, the corresponding set of statements get executed, rest gets ignored. If none of the condition is met then the statements inside "else" gets executed.

Example of if-else-if

The use of the 'if-else-if' statement is shown in the following example. Here, a string value will be taken as input from the user. The first 'if' condition will check the input value, and if it returns false, then the value will check by the next 'if' condition and so on. The message of the else part will print if all 'if' conditions return false.

```
//Import Scanner package
import java.util.Scanner;
public class if3 {
```

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

        // Create a Scanner object
        Scanner in = new Scanner(System.in);
        System.out.print("Enter your name : ");

        // Take string data from the user
        String name = in.next();

        // Check the input value equal to 'Jolly' or not
        if(name.equals("Jolly"))
        {
            System.out.print("You have achieved the first price");
        }
        // Check the input value equal to 'Janifer' or not
        else if(name.equals("Janifer"))
        {
            System.out.print("You have achieved the second price");
        }
        // Check the input value equal to 'Jony' or not
        else if(name.equals("Jony"))
        {
            System.out.print("You have achieved the third price");
        }
        else
        {
            System.out.print("Try for next time");
        }
        //Close the scanner object
            in.close();

    }
}
```

```
            System.out.println("Its a five digit number");
          }
          else {
            System.out.println("number is not between 1 & 99999");
          }
    }
}
```

Q4. What are loops, why they are used in java and how many types of loops are being supported by java explain in detail?

Answer:

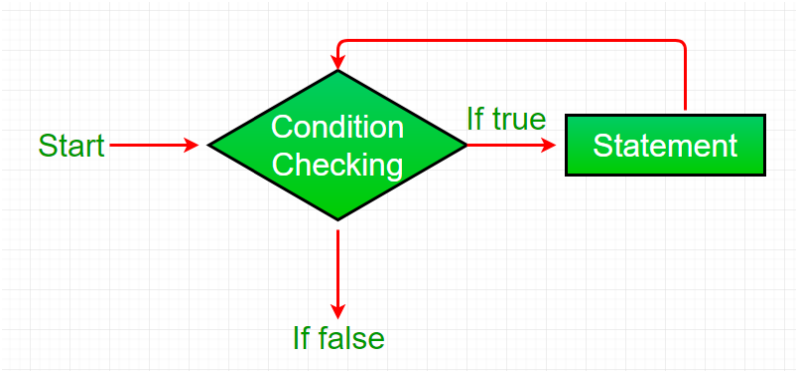
Loop:

Looping in programming languages is a feature which facilitates the execution of a set of instructions/functions repeatedly while some condition evaluates to true.
Java provides three ways for executing the loops. While all the ways provide similar basic functionality, they differ in their syntax and condition checking time.

1. **while loop:** A while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition. The while loop can be thought of as a repeating if statement.
   Syntax :

```
while (boolean condition)
{
   loop statements...
}
```

Flowchart:



- While loop starts with the checking of condition. If it evaluated to true, then the loop body statements are executed otherwise first statement following the loop is executed. For this reason it is also called **Entry control loop**
- Once the condition is evaluated to true, the statements in the loop body are executed. Normally the statements contain an update value for the variable being processed for the next iteration.
- When the condition becomes false, the loop terminates which marks the end of its life cycle.

```
// Java program to illustrate while loop
class whileLoopDemo
{
    public static void main(String args[])
    {
        int x = 1;
```

```
        // Exit when x becomes greater than 4
        while (x <= 4)
        {
            System.out.println("Value of x:" + x);

            // Increment the value of x for
            // next iteration
            x++;
        }
    }
}
```

Output:

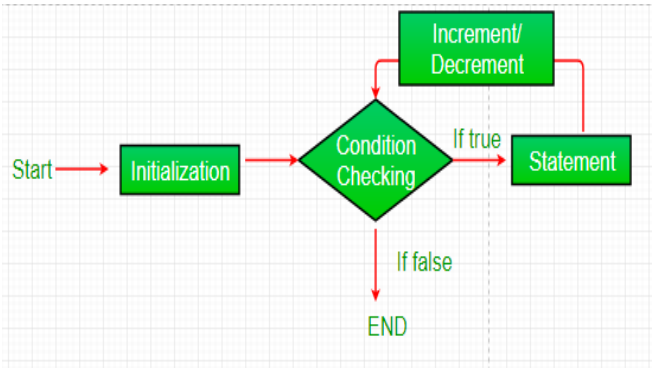
```
Value of x:1
Value of x:2
Value of x:3
Value of x:4
```

6. **for loop:** for loop provides a concise way of writing the loop structure. Unlike a while loop, a for statement consumes the initialization, condition and increment/decrement in one line thereby providing a shorter, easy to debug structure of looping.

   Syntax:

```
for (initialization condition; testing condition;
                                increment/decrement)
{
    statement(s)
}
```

   Flowchart:



1. **Initialization condition:** Here, we initialize the variable in use. It marks the start of a for loop. An already declared variable can be used or a variable can be declared, local to loop only.
2. **Testing Condition:** It is used for testing the exit condition for a loop. It must return a boolean value. It is also an **Entry Control Loop** as the condition is checked prior to the execution of the loop statements.
3. **Statement execution:** Once the condition is evaluated to true, the statements in the loop body are executed.
4. **Increment/ Decrement:** It is used for updating the variable for next iteration.
5. **Loop termination:**When the condition becomes false, the loop terminates marking the end of its life cycle.

```
// Java program to illustrate for loop.
class forLoopDemo
{
    public static void main(String args[])
    {
        // for loop begins when x=2
        // and runs till x <=4
        for (int x = 2; x <= 4; x++)
            System.out.println("Value of x:" + x);
    }
}
```

Output:

```
Value of x:2

Value of x:3

Value of x:4
```

Enhanced For loop:

Java also includes another version of for loop introduced in Java 5. Enhanced for loop provides a simpler way to iterate through the elements of a collection or array. It is inflexible and should be used only when there is a need to iterate through the elements in sequential manner without knowing the index of currently processed

element.
Also note that the object/variable is immutable when enhanced for loop is used i.e it ensures that the values in the array can not be modified, so it can be said as read only loop where you can't update the values as opposite to other loops where values can be modified.
We recommend using this form of the for statement instead of the general form whenever possible.(as per JAVA doc.)

Syntax:

```
for (T element:Collection obj/array)
{
    statement(s)
}
```

Lets take an example to demonstrate how enhanced for loop can be used to simpify the work. Suppose there is an array of names and we want to print all the names in that array. Let's see the difference with these two examples
Enhanced for loop simplifies the work as follows-

```
// Java program to illustrate enhanced for loop
public class enhancedforloop
{
    public static void main(String args[])
    {
        String array[] = {"Ron", "Harry", "Hermoine"};

        //enhanced for loop
        for (String x:array)
        {
            System.out.println(x);
        }

        /* for loop for same function
        for (int i = 0; i < array.length; i++)
        {
            System.out.println(array[i]);
        }
        */
    }
}
```

Output:

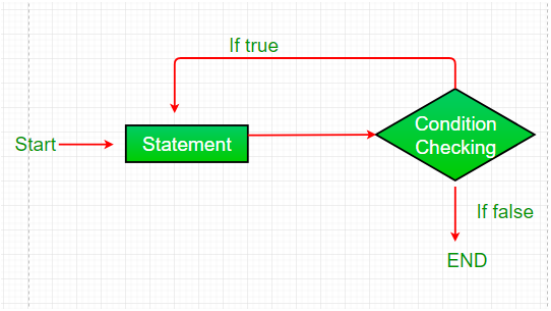
```
Ron

Harry

Hermoine
```

7. **do while:** do while loop is similar to while loop with only difference that it checks for condition after executing the statements, and therefore is an example of **Exit**

   Control Loop.
   Syntax:

```
do
{
    statements..
}
while (condition);
```

   Flowchart:



1. do while loop starts with the execution of the statement(s). There is no checking of any condition for the first time.
2. After the execution of the statements, and update of the variable value, the condition is checked for true or false value. If it is evaluated to true, next iteration of loop starts.
3. When the condition becomes false, the loop terminates which marks the end of its life cycle.
4. It is important to note that the do-while loop will execute its statements atleast once before any condition is checked, and therefore is an example of exit control loop.

```
// Java program to illustrate do-while loop
class dowhileloopDemo
{
    public static void main(String args[])
    {
        int x = 21;
        do
        {
```

```
            // The line will be printed even
            // if the condition is false
            System.out.println("Value of x:" + x);
            x++;
        }
        while (x < 20);
    }
}
```

Output:

```
Value of x: 21
```

Pitfalls of Loops

1. **Infinite loop:** One of the most common mistakes while implementing any sort of looping is that that it may not ever exit, that is the loop runs for infinite time. This happens when the condition fails for some reason.
   Examples:

```
//Java program to illustrate various pitfalls.
public class LooppitfallsDemo
{
    public static void main(String[] args)
    {

        // infinite loop because condition is not apt
        // condition should have been i>0.
        for (int i = 5; i != 0; i -= 2)
        {
            System.out.println(i);
        }
        int x = 5;

        // infinite loop because update statement
        // is not provided.
        while (x == 5)
        {
            System.out.println("In the loop");
        }
    }
}
```

2. Another pitfall is that you might be adding something into you collection object through loop and you can run out of memory. If you try and execute the below program, after some time, out of memory exception will be thrown.

```
//Java program for out of memory exception.
import java.util.ArrayList;
public class Integer1
{
    public static void main(String[] args)
```

```
    {
        ArrayList<Integer> ar = new ArrayList<>();
        for (int i = 0; i < Integer.MAX_VALUE; i++)
        {
            ar.add(i);
        }
    }
}
```

Output:

```
Exception in thread "main" java.lang.OutOfMemoryError: Java heap
space
at java.util.Arrays.copyOf(Unknown Source)
at java.util.Arrays.copyOf(Unknown Source)
at java.util.ArrayList.grow(Unknown Source)
at java.util.ArrayList.ensureCapacityInternal(Unknown Source)
at java.util.ArrayList.add(Unknown Source)
at article.Integer1.main(Integer1.java:9)
```

Q5. Write 3's table in decremented form in java which takes input from user write java coded program and explain in detail?

Answer:

This is a Java Program to Print Multiplication Table for any Number.
Enter any integer number as input of which you want multiplication table. After that we use for loop from one to ten to generate multiplication of that number.

Here is the source code of the Java Program to Print Multiplication Table for any Number. The Java program is successfully compiled and run on a Windows system. The program output is also shown below

```
public class MultiplicationTable {

    public static void main(String[] args) {

        int num = 3;
        for(int i = 10; i >= 1; --i)
```

```
        {
            System.out.printf("%d * %d = %d \n", num, i, num *i);
        }
    }
}
```

```
$javac MultiplicationTable.java
$java -Xmx128M -Xms16M MultiplicationTable
3 * 10 = 30
3 * 9 = 27
3 * 8 = 24
3 * 7 = 21
3 * 6 = 18
3 * 5 = 15
3 * 4 = 12
3 * 3 = 9
3 * 2 = 6
3 * 1 = 3
```

In the above program, we have to decrement the value of i inside the body of the loop. Though both programs are technically correct, it is better to use for loop in this case. It's because the number of iteration (from 10 to 1) is known.

<<<<<<<<<<<<<<<<<<<<<<<<<<<END>>>>>>>>>>>>>>>>>>>>>>>>>>>