NAME:HIDAYAT UR RAHMAN
ID:15125
BS (SE) 5TH
SUB:OBJECT ORIENTED PROGRAM
SIR:M AYUB KHAN

Q1:

ANS: A variable provides us with named storage that our programs can manipulate. Each variable in Java has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.

You must declare all variables before they can be used. Following is the basic form of a variable declaration − data type variable [= value][, variable [= value] ...] ;

Here *data type* is one of Java's datatypes and *variable* is the name of the variable. To declare more than one variable of the specified type, you can use a comma-separated list.

Following are valid examples of variable declaration and initialization in Java −

Example
```
int a, b, c;         // Declares three ints, a, b, and c.
int a = 10, b = 10;  // Example of initialization
byte B = 22;         // initializes a byte type variable B.
double pi = 3.14159; // declares and assigns a value of PI.
char a = 'a';        // the char variable a iis initialized with value 'a'
```

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();
  }
}
```

Output
Puppy age is: 7

Example

Following example uses *age* without initializing it, so it would give an error at the time of compilation.

```
public class Test {
  public void pupAge() {
    int age;
    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 error while compiling it −

Output

```
Test.java:4:variable number might not have been initialized
age = age + 7;
      ^
1 error
```

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();
  }
}
```

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);
  }
}
```

Output
Development average salary:1000

Q4:

ANS: Loop

There may be a situation when you need to execute a block of code several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.

Programming languages provide various control structures that allow for more complicated execution paths.

A **loop** statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages −

Java programming language provides the following types of loop to handle looping requirements. Click the following links to check their detail.

1	while loop Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
2	for loop Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.
3	do...while loop **Like a while statement, except that it tests the condition at the end of the loop body.**

Loop Control Statements

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

Java supports the following control statements. Click the following links to check their detail.

1	break statement Terminates the **loop** or **switch** statement and transfers execution to the statement immediately following the loop or switch.
2	continue statement Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

Enhanced for loop in Java

As of Java 5, the enhanced for loop was introduced. This is mainly used to traverse collection of elements including arrays.

Syntax

Following is the syntax of enhanced for loop −

```
for(declaration : expression) {
   // Statements
}
```

- **Declaration** − The newly declared block variable, is of a type compatible with the elements of the array you are accessing. The variable will be available within the for block and its value would be the same as the current array element.

- **Expression** − This evaluates to the array you need to loop through. The expression can be an array variable or method call that returns an array.

Example

```
public class Test {

   public static void main(String args[]) {
      int [] numbers = {10, 20, 30, 40, 50};

      for(int x : numbers ) {
         System.out.print( x );
         System.out.print(",");
      }
      System.out.print("\n");
      String [] names = {"James", "Larry", "Tom", "Lacy"};

      for( String name : names ) {
         System.out.print( name );
         System.out.print(",");
      }
   }
}
```

This will produce the following result −

Output

```
10, 20, 30, 40, 50,

James, Larry, Tom, Lacy,
```

Q 2:

ANS: If 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.

   a. If Condition yields true, goto Step 4.
   b. 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:

filter_none
edit
play_arrow
brightness_4

```
// 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
```

Dry-Running Example 1:

```
1. Program starts.
2. i is initialized to 10.
3. if-condition is checked. 10<15, yields true.
  3.a) "10 is less than 15" gets printed.
4. "Outside if-block" is printed.
```

Example 2:

filter_none
edit
play_arrow

brightness_4

```
// Java program to illustrate If statement

class IfDemo {
    public static void main(String args[])
    {
        String str = "GeeksforGeeks";
        int i = 4;

        // if block
        if (i == 4) {
            i++;
            System.out.println(str);
        }

        // Executed by default
        System.out.println("i = " + i);
    }
}
```

Output:

```
GeeksforGeeks
i = 5
```

Q3:

ANS: 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**. **if...else...if** ladder, that can be used to execute one block of code among multiple other blocks

```
if(condition_1) {
   /*if condition_1 is true execute this*/
   statement(s);
}
else if(condition_2) {
   /* execute this if condition_1 is not met and
    * condition_2 is met
    */
   statement(s);
}
else if(condition_3) {
   /* execute this if condition_1 & condition_2 are
    * not met and condition_3 is met
    */
   statement(s);
}
.
.
.
else {
   /* if none of the condition is true
    * then these statements gets executed
    */
   statement(s);
}
```

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

```
public class IfElseIfExample {

   public static void main(String args[]){
          int num=1234;
          if(num <100 && num>=1) {
            System.out.println("Its a two digit number");
          }
          else if(num <1000 && num>=100) {
            System.out.println("Its a three digit number");
          }
          else if(num <10000 && num>=1000) {
            System.out.println("Its a four digit number");
          }
          else if(num <100000 && num>=10000) {
            System.out.println("Its a five digit number");
          }
          else {
            System.out.println("number is not between 1 & 99999");
          }
   }
}
```

Output:

```
Its a four digit number
```

Q5:

ANS: import java.util.Scanner;

public class Main {

 public static void main(String[] args) {
  Scanner in = new Scanner(System.in);
  System.out.println("Input the Number: ");
  int n = in .nextInt();
  for (int i = 1; i <= 10; i++) {
   System.out.println(n + "*" + i + " = " + (n * i));
  }
 }
}

THE END