Name Muhammad Ali Khan REG 16550

Question No 1

Variable

A variable is a name which is associated with a value that can be changed. For example when I write int i=10; here variable name is i which is associated with value 10, int is a data type that represents that this variable can hold integer values. We will cover the data types in the next tutorial. In this tutorial, we will discuss about variables.

How to Declare a variable in Java

To declare a variable follow this syntax:

```
data_type variable_name = value;
```

here value is optional because in java, you can declare the variable first and then later assign the value to it.

For example: Here num is a variable and int is a data type. We will discuss the data type in next tutorial so do not worry too much about it, just understand that int data type allows this num variable to hold integer values. You can read data types here but I would recommend you to finish reading this guide before proceeding to the next one.

```
int num;
```

Similarly we can assign the values to the variables while declaring them, like this:

```
char ch = 'A';
```

```
int number = 100;
```

Name Muhammad Ali Khan REG 16550

or we can do it like this:

```
char ch;

int number;

...

ch = 'A';

number  = 100;
```

Variables naming convention in java

1) Variables naming cannot contain white spaces, for example: int num ber = 100; is invalid because the variable name has space in it.

2) Variable name can begin with special characters such as $ and _

3) As per the java coding standards the variable name should begin with a lower case letter, for example int number; For lengthy variables names that has more than one words do it like this: int smallNumber; int bigNumber; (start the second word with capital letter).

4) Variable names are case sensitive in Java.

Types of Variables in Java

There are **three types of variables** in Java.
1) Local variable 2) Static (or class) variable 3) Instance variable

Static (or class) Variable

Static variables are also known as class variable because they are associated with the class and common for all the instances of class. For example, If I create three objects of a class and access this static variable, it would be common for all, the changes made to the variable using one of the object would reflect when you access it through other objects.

Example of static variable

Name Muhammad Ali Khan REG 16550

```
public class StaticVarExample {
   public static String myClassVar="class or static variable";

   public static void main(String args[]){
      StaticVarExample obj = new StaticVarExample();
      StaticVarExample obj2 = new StaticVarExample();
      StaticVarExample obj3 = new StaticVarExample();

      //All three will display "class or static variable"
      System.out.println(obj.myClassVar);
      System.out.println(obj2.myClassVar);
      System.out.println(obj3.myClassVar);

      //changing the value of static variable using obj2
      obj2.myClassVar = "Changed Text";

      //All three will display "Changed Text"
      System.out.println(obj.myClassVar);
      System.out.println(obj2.myClassVar);
      System.out.println(obj3.myClassVar);
   }
}
```

Output:

```
class or static variable
class or static variable
class or static variable
Changed Text
Changed Text
Changed Text
```

As you can see all three statements displayed the same output irrespective of the instance through which it is being accessed. That's is why we can access the static variables without using the objects like this:

```
System.out.println(myClassVar);
```

Do note that only static variables can be accessed like this. This doesn't apply for instance and local variables.

Instance variable

Each instance(objects) of class has its own copy of instance variable. Unlike static variable, instance variables have their own separate copy of instance variable. We have changed the instance variable value using object obj2 in the following program and when we displayed the variable using all three objects, only the obj2 value got changed, others remain unchanged. This shows that they have their own copy of instance variable.

Example of Instance variable

```
public class InstanceVarExample {
   String myInstanceVar="instance variable";

   public static void main(String args[]){
       InstanceVarExample obj = new InstanceVarExample();
       InstanceVarExample obj2 = new InstanceVarExample();
       InstanceVarExample obj3 = new InstanceVarExample();

       System.out.println(obj.myInstanceVar);
       System.out.println(obj2.myInstanceVar);
       System.out.println(obj3.myInstanceVar);


       obj2.myInstanceVar = "Changed Text";


       System.out.println(obj.myInstanceVar);
       System.out.println(obj2.myInstanceVar);
       System.out.println(obj3.myInstanceVar);
   }
}
```

Output:

```
instance variable
instance variable
instance variable
instance variable
Changed Text
instance variable
```

Local Variable

These variables are declared inside method of the class. Their scope is limited to the method which means that You can’t change their values and access them outside of the method.

In this example, I have declared the instance variable with the same name as local variable, this is to demonstrate the scope of local variables.

Example of Local variable   **Name Muhammad Ali Khan REG 16550**

```
public class VariableExample {
   // instance variable
   public String myVar="instance variable";

   public void myMethod(){
       // local variable
       String myVar = "Inside Method";
       System.out.println(myVar);
   }
   public static void main(String args[]){
      // Creating object
      VariableExample obj = new VariableExample();

      /* We are calling the method, that changes the
       * value of myVar. We are displaying myVar again after
       * the method call, to demonstrate that the local
       * variable scope is limited to the method itself.
       */
      System.out.println("Calling Method");
      obj.myMethod();
      System.out.println(obj.myVar);
   }
}
```

Output:

```
Calling Method
Inside Method
instance variable
```

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

Java if, Statement

In computer programming, it's often desirable to execute a certain section of code based upon whether the specified condition is `true` or `false` (which is known only during the run time). For such cases, control flow statements are used.

Defination

If statement consists a condition, followed by statement or a set of statements as shown below:

```
if(condition){
  Statement(s);
}
```

The statements gets executed only when the given condition is true. If the condition is false then the statements inside if statement body are completely ignored.

Why we use if?

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. Control falls into the **if** block.

Java if (if-then) Statement

In Java, the syntax of the **if-then** statement is:

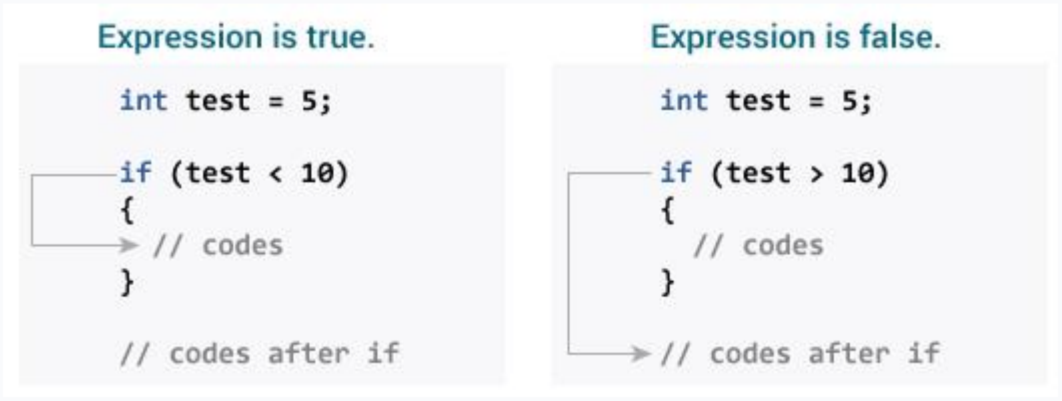
```
if (expression) {
    // statements
}
```

Here `expression` is a boolean expression. A boolean expression returns either true or false.

- if the expression is evaluated to `true`, statement(s) inside the body of `if` (statements inside parenthesis) are executed
- if the expression is evaluated to `false`, statement(s) inside the body of `if` are skipped from execution

---

How if statement works?



```
int test = 5;

if (test < 10)
{
   // codes
}

// codes after if
```

```
int test = 5;

if (test > 10)
{
   // codes
}

// codes after if
```

Working of Java if statement

Example 1: Java if Statement

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

        int number = 10;

        // checks if number is greater than 0
        if (number > 0) {
            System.out.println("The number is positive.");
        }
        System.out.println("This statement is always executed.");
    }
}
```

Output:

```
The number is positive.
This statement is always executed.
```

In the above example, we have a variable named `number`. Here, the test expression checks if the `number` is greater than `0` (`number > 0`).

Since the `number` is greater than `0`. So the test expression evaluates to `true`. Hence code inside the body of `if` is executed.

Now, change the value of the `number` to a negative integer. Let's say -5.

```
int number = -5;
```

If we run the above program with the new value of the `number`, the output will be:

```
This statement is always executed.
```

Here, the value of `number` is less than `0`. So, the test expression `number > 0` evaluates to `false`. Hence, the body of `if` is executed.

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

Java if..else..if Statement

In Java, we have an **if...else...if** ladder, that can be used to execute one block of code among multiple other blocks.

```
if (expression1) {
   // codes
}
else if(expression2) {
   // codes
}
else if (expression3) {
   // codes
}
.
.
else {
   // codes
}
```

Here, `if` statements are executed from the top towards the bottom. As soon as the test expression is `true`, codes inside the body of that the `if` statement is executed. Then, the control of the program jumps outside the `if-else-if` ladder. If all test expressions are `false`, codes inside the body of `else` is executed.

Example 3: Java if..else..if Statement

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

        int number = 0;

        // checks if number is greater than 0
        if (number > 0) {
            System.out.println("The number is positive.");
        }
        // checks if number is less than 0
        else if (number < 0) {
            System.out.println("The number is negative.");
        }
        else {
            System.out.println("The number is 0.");
        }
    }
}
```

Output:

```
The number is 0.
```

In the above example, we are checking whether the `number` is positive, negative or zero. Here, we have two test expressions:

- `number > 0` - checks if the `number` is greater than `0`
- `number < 0` - checks if the `number` is less than `0`

Here, the value of `number` is `0`. So both the test expression evaluates to `false`. Hence the statement inside the body of `else` is executed.

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

WHAT AND WHY WE USE LOOP

In computer programming, loops are used to repeat a block of code. For example, if you want to show a message 100 times, then rather than typing the same code 100 times, you can use a loop.

In Java, there are three types of loops.

- for loop
- while loop
- do...while loop

This tutorial focuses on the for loop. You will learn about the other type of loops in the upcoming tutorials.

Java for Loop

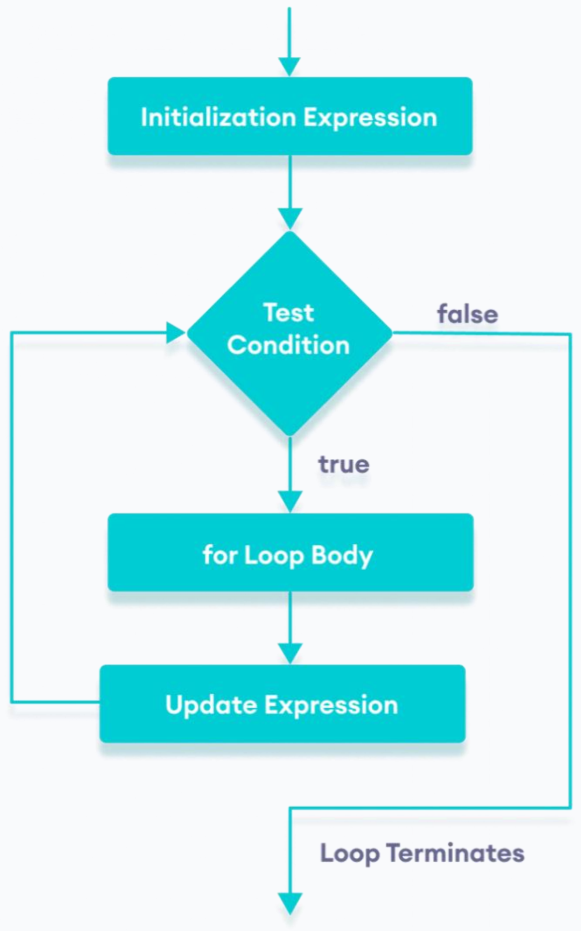
Java `for` loop is used to run a block of code for a certain number of times. The syntax of `for` loop is:

```
for (initialExpression; testExpression; updateExpression) {
    // body of the loop
}
```

Here,

1. The **initialExpression** initializes and/or declares variables and executes only once.
2. The **condition** is evaluated. If the **condition** is `true`, the body of the `for` loop is executed.
3. The **updateExpression** updates the value of **initialExpression**.
4. The **condition** is evaluated again. The process continues until the **condition** is `false`.

To learn more about the conditions, visit Java relational and logical operators.



Flowchart of Java for loop

Example 1: Display a Text Five Times

```
// Program to print a text 5 times

class Main {
  public static void main(String[] args) {

    int n = 5;
    // for loop
    for (int i = 1; i <= n; ++i) {
      System.out.println("Java is fun");
    }
  }
}
```

Output

```
Java is fun
Java is fun
Java is fun
Java is fun
Java is fun
```

Here is how this program works.

Iteration	Variable	Condition: i <= n	Action
1st	`i = 1` `n = 5`	`true`	`Java is fun` is printed. `i` is increased to **2**.
2nd	`i = 2` `n = 5`	`true`	`Java is fun` is printed. `i` is increased to **3**.
3rd	`i = 3` `n = 5`	`true`	`Java is fun` is printed. `i` is increased to **4**.
4th	`i = 4` `n = 5`	`true`	`Java is fun` is printed. `i` is increased to **5**.
5th	`i = 5` `n = 5`	`true`	`Java is fun` is printed. `i` is increased to **6**.
6th	`i = 6` `n = 5`	`false`	The loop is terminated.

Java while and do...while Loop

In this tutorial, we will learn how to use while and do while loop in Java with the help of examples.

In computer programming, loops are used to repeat a block of code. For example, if you want to show a message 100 times, then you can use a loop. It's just a simple example; you can achieve much more with loops.

In the previous tutorial, you learned about Java for loop. Here, you are going to learn about `while` and `do...while` loops.

Java while loop

Java `while` loop is used to run a specific code until a certain condition is met. The syntax of the `while` loop is:

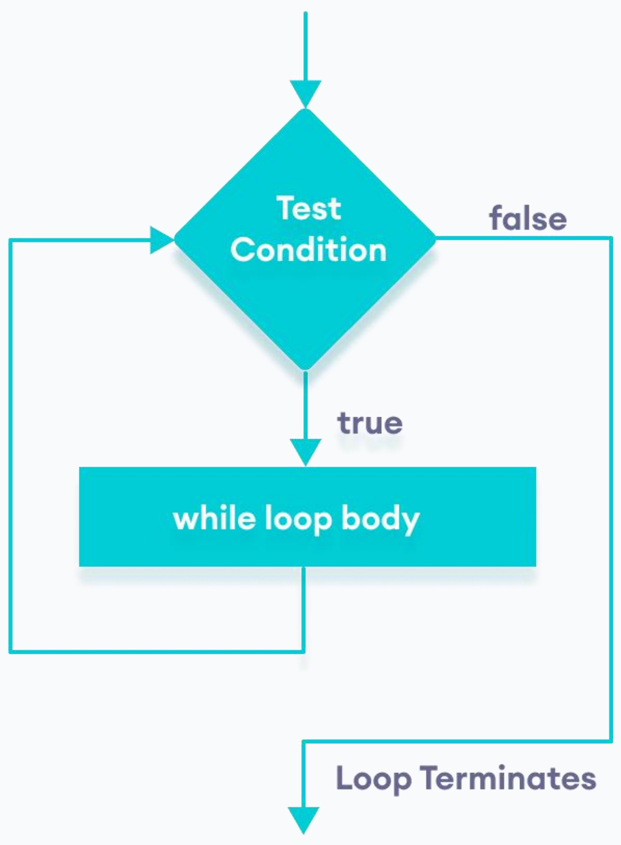
```
while (testExpression) {
    // body of loop
}
```

Here,

1. A `while` loop evaluates the **textExpression** inside the parenthesis `()`.
2. If the **textExpression** evaluates to `true`, the code inside the `while` loop is executed.
3. The **textExpression** is evaluated again.
4. This process continues until the **textExpression** is `false`.
5. When the **textExpression** evaluates to `false`, the loop stops.

To learn more about the conditions, visit Java relational and logical operators.

Flowchart of while loop



Flowchart of Java while loop

Example 1: Display Numbers from 1 to 5

```
// Program to display numbers from 1 to 5

class Main {
  public static void main(String[] args) {

    // declare variables
    int i = 1, n = 5;

    // while loop from 1 to 5
    while(i <= n) {
      System.out.println(i);
      i++;
    }
  }
}
```

Output

```
1
2
3
4
5
```

Here is how this program works.

Iteration	Variable	Condition: i <= n	Action

1st	`i = 1`
	`n = 5`
		`true`	`1` is printed.
			`i` is increased to **2**.

2nd	`i = 2`
	`n = 5`
		`true`	`2` is printed.
			`i` is increased to **3**.

3rd	`i = 3`
	`n = 5`
		`true`	`3` is printed.
			`i` is increased to **4**.

4th	`i = 4`
	`n = 5`
		`true`	`4` is printed.
			`i` is increased to **5**.

5th	`i = 5`
	`n = 5`
		`true`	`5` is printed.
			`i` is increased to **6**.

6th	`i = 6`
	`n = 5`
		`false`	The loop is terminated

Java do...while loop

The `do...while` loop is similar to while loop. However, the body of `do...while` loop is executed once before the test expression is checked. For example,

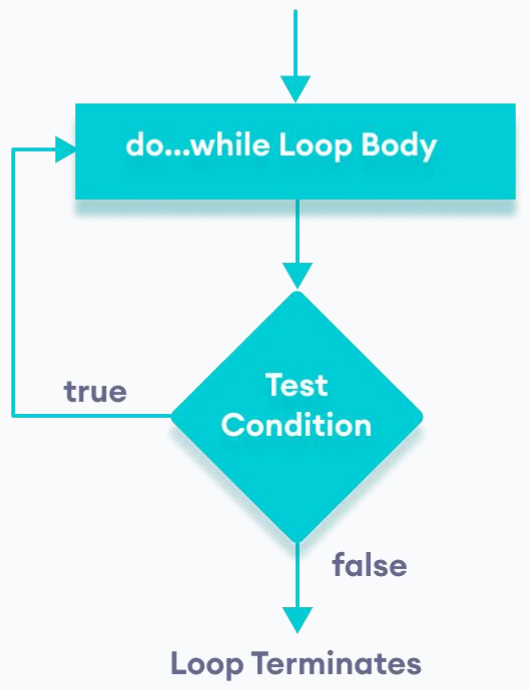
```
do {
    // body of loop
} while(textExpression)
```

Here,

1. The body of the loop is executed at first. Then the **textExpression** is evaluated.
2. If the **textExpression** evaluates to `true`, the body of the loop inside the `do` statement is executed again.
3. The **textExpression** is evaluated once again.
4. If the **textExpression** evaluates to `true`, the body of the loop inside the `do` statement is executed again.
5. This process continues until the **textExpression** evaluates to `false`. Then the loop stops.

Name Muhammad Ali Khan REG 16550

Flowchart of do...while loop



Flowchart of Java do while loop

Let's see the working of `do...while` loop.

Example 3: Display Numbers from 1 to 5

```
// Java Program to display numbers from 1 to 5

import java.util.Scanner;



// Program to find the sum of natural numbers from 1 to 100.

class Main {
  public static void main(String[] args) {

    int i = 1, n = 5;

    // do...while loop from 1 to 5
    do {
      System.out.println(i);
      i++;
    } while(i <= n);
  }
}
```

Output

```
1
2
3
4
5
```

Here is how this program works.

Iteration	Variable	Condition: i <= n	Action

	`i = 1`
	`n = 5`
		not checked	`1` is printed.
			`i` is increased to **2**.

1st	`i = 2`
	`n = 5`
		`true`	`2` is printed.
			`i` is increased to **3**.

2nd	`i = 3`
	`n = 5`
		`true`	`3` is printed.
			`i` is increased to **4**.

3rd	`i = 4`
	`n = 5`
		`true`	`4` is printed.
			`i` is increased to **5**.

4th	`i = 5`
	`n = 5`
		`true`	`6` is printed.
			`i` is increased to **6**.

5th	`i = 6`
	`n = 5`
		`false`	The loop is terminated

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

```
import java.util.Scanner;


public class Main {


 public static void main(String[] args) {

  Scanner in = new Scanner(System.in);

  System.out.println( "Muhammad Ali Khan"+"REG#16550" );

  System.out.println("Input the Number: ");

  int n = in .nextInt();

  for (int i = 10; i >= 1; i--) {

   System.out.println(n + "*" + i + " = " + (n * i));


  }

 }

}
```

Details

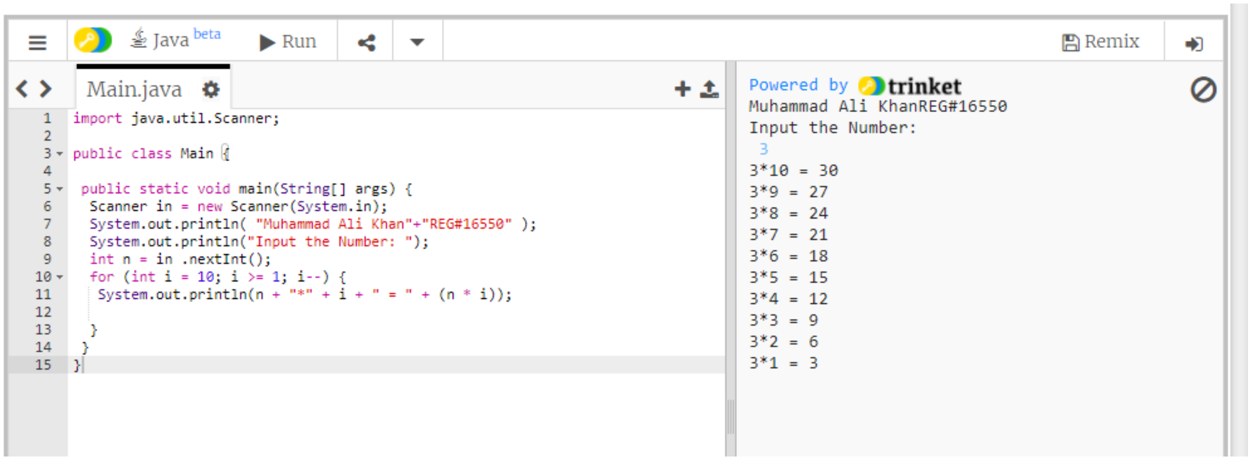
In Above code   Scanner in = new Scanner(System.in);   int n = in .nextInt();

are used for Input

And for the repetition of table we used for

  for (int i = 10; i >= 1; i--) {

   System.out.println(n + "*" + i + " = " + (n * i)); and this the output code of table



```java
import java.util.Scanner;

public class Main {

 public static void main(String[] args) {
  Scanner in = new Scanner(System.in);
  System.out.println( "Muhammad Ali Khan"+"REG#16550" );
  System.out.println("Input the Number: ");
  int n = in .nextInt();
  for (int i = 10; i >= 1; i--) {
   System.out.println(n + "*" + i + " = " + (n * i));

  }
 }
}
```

```
Muhammad Ali KhanREG#16550
Input the Number:
 3
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
```