NAME : HAMZA IQBAL

ID : 14784

PAPAR : OBJECT ORIENTED PROGRAMMING

DEPT : BS (SE)

TEACHER : M. AYUB KHAN

Q1. How many variables are being supported by java justify your answer with

 the help java coded example for each variable?

Answer:

 Variable Types

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

1. Local variables are declared in methods, constructors, or blocks.
2. 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.
3. Access modifiers cannot be used for local variables.
4. Local variables are visible only within the declared method, constructor, or block.
5. Local variables are implemented at stack level internally.
6. 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**

1. Instance variables are declared in a class, but outside a method, constructor or any block.
2. When a space is allocated for an object in the heap, a slot for each instance variable value is created.
3. Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.
4. 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.
5. Instance variables can be declared in class level before or after use.
6. Access modifiers can be given for instance variables.
7. 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.
8. 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.

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**

1. Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block.
2. There would only be one copy of each class variable per class, regardless of how many objects are created from it.
3. 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.
4. 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.
5. Static variables are created when the program starts and destroyed when the program stops.
6. Visibility is similar to instance variables. However, most static variables are declared public since they must be available for users of the class.
7. 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.
8. Static variables can be accessed by calling with the class name *ClassName.VariableName*.

**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

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: Loops:

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.

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.



Java programming language provides the following types of loop to handle looping requirements.

|  |  |
| --- | --- |
|  |  |
|  | While loop: Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body. |
|  | For loop :Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable. |
|  | 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.

|  |  |
| --- | --- |
|  |  |
|  | [Break statement](https://www.tutorialspoint.com/java/java_break_statement.htm) :Terminates the **loop** or **switch** statement and transfers execution to the statement immediately following the loop or switch. |
|  | Continue Statement:Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating. |

##  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,

Q3. Why “if else if” is used in java justify your answer with the help java coded

 example and explain in detail?

ANSWER:

When we need to execute a set of statements based on a condition then we need to use **control flow statements**. For example, if a number is greater than zero then we want to print “Positive Number” but if it is less than zero then we want to print “Negative Number”. In this case we have two print statements in the program, but only one print statement executes at a time based on the input value. We will see how to write such type of conditions in the java program using control statements.

There are four types

a) if statement
b) nested if statement
c) if-else statement
d) if-else-if statement

**If statement**

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.


**Example of if statement**

public class IfStatementExample {

 public static void main(String args[]){

 int num=70;

 if( num < 100 ){

 /\* This println statement will only execute,

 \* if the above condition is true

 \*/

 System.out.println("number is less than 100");

 }

 }

}

**Output:**

number is less than 100

**Nested if statement in Java**

When there is an if statement inside another if statement then it is called the **nested if statement**.
The structure of nested if looks like this:

if(condition\_1) {

 Statement1(s);

 if(condition\_2) {

 Statement2(s);

 }

}

Statement1 would execute if the condition\_1 is true. Statement2 would only execute if both the conditions( condition\_1 and condition\_2) are true.

**Example of Nested if statement**

public class NestedIfExample {

 public static void main(String args[]){

 int num=70;

 if( num < 100 ){

 System.out.println("number is less than 100");

 if(num > 50){

 System.out.println("number is greater than 50");

 }

 }

 }

}

**Output:**

number is less than 100

number is greater than 50

# If else statement in Java

This is how an if-else statement .

if(condition) {

 Statement(s);

}

else {

 Statement(s);

}

The statements inside “if” would execute if the condition is true, and the statements inside “else” would execute if the condition is false.


**Example of if-else statement**

public class IfElseExample {

 public static void main(String args[]){

 int num=120;

 if( num < 50 ){

 System.out.println("num is less than 50");

 }

 else {

 System.out.println("num is greater than or equal 50");

 }

 }

}

**Output:**

num is greater than or equal 50

**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(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);

}

**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**

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

Q2. Why “If” is used in java justify your answer with the help java coded

 example and explain in detail?

ANSWER: If statement

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.

The if Statement

Use the if statement to specify a block of Java code to be executed if a condition is true.

Syntax

if (*condition*) {

 *// block of code to be executed if the condition is true*

}

Java has the following conditional statements:

* Use if to specify a block of code to be executed, if a specified condition is true
* Use else to specify a block of code to be executed, if the same condition is false
* Use else if to specify a new condition to test, if the first condition is false
* Use switch to specify many alternative blocks of code to be executed

Example:

 import java.util.;/\*

 \* ali

 \*/

public class ali {

 public static void main(String[] args) {

 int a=50;

 int b=100;

 if (a>b) {

 System.out.println("A is greater");

 }

 }

}

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 Exercise7 {

 public static void main(String[] args) {

 Scanner in = new Scanner(System.in);

 System.out.print("Input a number: ");

 int num1 = in.nextInt();

 for (i

nt i=0; i< 10; i++){

 System.out.println(num1 + " x " + (i+1) + " = " +

 (num1 \* (i+1)));

 }

 }

}

OUTPUT

Input a number: 3

3 x 1 = 3

3 x 2 = 6

3 x 3 = 9

3 x 4 = 12

3 x 5 = 15

3 x 6 = 18

3 x 7 = 21

3 x 8 = 24

3 x 9 = 27

3 x 10 = 30