**MARION WAITHERA**

**SCT221-0805/2022**

**ASSIGNMENT**

**Section 1**

**1. Differences between Primitive and Reference Data Types**

* **Primitive Data Types:**
  + **Definition:** Basic data types provided by Java that hold their values directly.
  + **Examples:** int, char, boolean, double, etc.
  + **Size:** Fixed size (e.g., int is 4 bytes).
  + **Storage:** Directly stored in memory.
  + **Default Values:** Have default values (e.g., 0 for int, false for boolean).
* **Reference Data Types:**
  + **Definition:** Types that refer to objects and hold references (or pointers) to the actual data.
  + **Examples:** String, Arrays, Classes, Interfaces.
  + **Size:** Variable size depending on the object.
  + **Storage:** Store memory address (reference) where the actual object data is kept.
  + **Default Values:** Default to null if not initialized.

**2. Scope of a Variable**

* **Local Variable:**
  + **Definition:** Declared within a method or block and can only be accessed within that method or block.
  + **Lifetime:** Exists only while the method or block is executing.
* **Global Variable (Instance Variable):**
  + **Definition:** Declared within a class but outside any method.
  + **Scope:** Accessible by all methods within the class.
  + **Lifetime:** Exists as long as the object instance exists.

**3. Why Initialization of Variables is Required**

* **Reason:** Variables must be initialized before use to ensure they have a defined value. Uninitialized variables can lead to unpredictable behavior or runtime errors, as the default value might not be meaningful for certain operations.

**4. Static, Instance, and Local Variables**

* **Static Variables:**
  + **Definition:** Declared with the static keyword.
  + **Scope:** Shared among all instances of a class.
  + **Lifetime:** Exists for the duration of the program execution.
* **Instance Variables:**
  + **Definition:** Declared without the static keyword.
  + **Scope:** Each instance of the class has its own copy.
  + **Lifetime:** Exists as long as the object instance exists.
* **Local Variables:**
  + **Definition:** Declared inside a method or block.
  + **Scope:** Accessible only within the method or block where declared.
  + **Lifetime:** Exists only during the execution of the method or block.

**5. Widening vs. Narrowing Casting**

* **Widening Casting (Implicit):**
  + **Definition:** Automatic conversion from a smaller data type to a larger data type (e.g., int to double).
  + **Example:** int to double is widening because double can hold all values of int.
* **Narrowing Casting (Explicit):**
  + **Definition:** Manual conversion from a larger data type to a smaller data type (e.g., double to int).
  + **Example:** double to int requires explicit casting and may lose precision.

**6. Data Type Table**

| **TYPE** | **SIZE (IN BYTES)** | **DEFAULT** | **RANGE** |
| --- | --- | --- | --- |
| boolean | 1 bit | false | true, false |
| char | 2 | '\0000' | '\u0000' to '\uffff' |
| byte | 1 | 0 | -128 to 127 |
| short | 2 | 0 | -32,768 to 32,767 |
| int | 4 | 0 | -2^31 to 2^31-1 |
| long | 8 | 0L | -2^63 to 2^63-1 |
| float | 4 | 0.0f | ±1.4E-45 to ±3.4E38 |
| double | 8 | 0.0 | ±4.9E-324 to ±1.8E308 |

**7. Define Class in OOP**

* **Class:**
  + **Definition:** A blueprint for creating objects that defines a datatype by bundling data and methods that work on the data.
  + **Components:** Fields (variables), methods, constructors, and access modifiers.

**8. Importance of Classes in Java Programming**

* **Encapsulation:** Classes bundle data (fields) and methods (functions) together, encapsulating the state and behavior of objects.
* **Reusability:** Classes allow for the creation of reusable code through inheritance and composition.
* **Abstraction:** Classes provide a way to model real-world entities and hide implementation details.
* **Maintainability:** Classes organize code, making it easier to manage and understand.

**Section 2**

**1. Java Program: Sur Name and Age**

import java.util.Scanner;

public class SurNameAge {

public static void main(String[] args) {

Scanner Scan = new Scanner(System.in);

System.out.print("Enter your surname: ");

String surname = Scan.nextLine();

System.out.print("Enter your current age: ");

int age = Scan.nextInt();

int surnameLength = surname.length();

String ageParity = (age % 2 == 0) ? "even" : "odd";

System.out.println("The number of characters in your surname is: " + surnameLength);

System.out.println("Your current age is an " + ageParity + " number.");

Scan.close();

}

}

**2. Java Program: Marks Average**

import java.util.Scanner;

public class MarksAverage {

public static void main(String[] args) {

Scanner Scan = new Scanner(System.in);

System.out.println("Enter the marks for five units:");

double[] marks = new double[5];

double sum = 0;

for (int i = 0; i < 5; i++) {

System.out.print("Unit " + (i + 1) + ": ");

marks[i] = Scan.nextDouble();

sum += marks[i];

}

double average = sum / 5;

System.out.printf("Average marks: %.2f%n", average);

Scan.close();

}

}

**3. Java Program: Divisibility Test**

import java.util.Scanner;

public class DivisibilityTest {

public static void main(String[] args) {

Scanner Scan = new Scanner(System.in);

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

int number = Scan.nextInt();

for (int divisor = 1; divisor <= 9; divisor++) {

if (number % divisor == 0) {

System.out.println(number + " is divisible by " + divisor + " because it ends with a " + (divisor == 0 ? "0" : divisor) + ".");

}

}

Scan.close();

}

}

**4. Java Program: Multiples of 2, 3, and 7**

public class MultiplesFinder {

public static void main(String[] args) {

System.out.println("Multiples of 2, 3, and 7 between 71 and 150:");

for (int num = 71; num <= 150; num++) {

if (num % 2 == 0 || num % 3 == 0 || num % 7 == 0) {

System.out.println(num);

}

}

}

}

**5. Java Program: Basic Calculator**

import java.util.Scanner;

public class BasicCalculator {

public static void main(String[] args) {

Scanner Scan = new Scanner(System.in);

System.out.print("Enter first number: ");

double num1 = Scan.nextDouble();

System.out.print("Enter an operator (+, -, \*, /): ");

char operator = Scan.next().charAt(0);

System.out.print("Enter second number: ");

double num2 = Scan.nextDouble();

double result = 0;

boolean validOperation = true;

switch (operator) {

case '+':

result = num1 + num2;

break;

case '-':

result = num1 - num2;

break;

case '\*':

result = num1 \* num2;

break;

case '/':

if (num2 != 0) {

result = num1 / num2;

} else {

System.out.println("Error: Division by zero.");

validOperation = false;

}

break;

default:

System.out.println("Error: Invalid operator.");

validOperation = false;

break;

}

if (validOperation) {

System.out.println("Result: " + result);

}

Scan.close();

}

}