**DENNIS MUNYOKI**

**22/00372**

**SECTION 1 ANSWERS**

**NO 1: -** **Explain the differences between primitive and reference data types.**

**Definition**:

* **Primitive Data Types:** Primitive data types are basic data types provided by the programming language itself. They are predefined and built-in. Examples include integers, floating-point numbers, characters, booleans, and so on.
* **Reference Data Types:** Reference data types are complex data types that are defined by the programmer or provided by libraries and frameworks. They are called reference types because they refer to objects stored in memory. Examples include arrays, strings, classes, interfaces, and user-defined types.

**Storage:**

* **Primitive Data Types:** Primitive data types store the actual value of the variable. They are typically stored on the stack memory, which is a small and fast region of memory.
* **Reference Data Types:** Reference data types store a reference (i.e., memory address) to the location of the actual data. The data itself is stored on the heap memory, which is a larger and slower region of memory. The reference is stored on the stack memory.

**Memory Management**:

* **Primitive Data Types**: Memory allocation and deallocation for primitive types are automatically managed by the programming language. When a primitive variable goes out of scope, its memory is automatically released.
* **Reference Data Types:** Memory allocation and deallocation for reference types are typically managed manually by the programmer. The programmer must explicitly allocate memory using the "new" keyword and release it using the "delete" or garbage collection mechanisms. If the programmer fails to release the memory, it can lead to memory leaks.

**Assignment and Comparison:**

* **Primitive Data Types:** When you assign a primitive variable to another variable or compare two primitive variables, the actual value is copied from one variable to another.
* **Reference Data Types:** When you assign a reference variable to another variable or compare two reference variables, only the memory address (reference) is copied, not the actual data. This means that both variables will point to the same object in memory.

**Default Values:**

* **Primitive Data** Types: Primitive data types have default values assigned by the programming language. For example, the default value of an integer is typically 0, and the default value of a boolean is false.
* **Reference Data Types:** Reference data types have a default value of "null" or "undefined," indicating that they do not refer to any valid object.

**NO 2 : Define the scope of a variable (hint: local and global variable)**

* The scope of a variable refers to the region or portion of the code where the variable is accessible and can be referenced. In most programming languages, there are two common types of variable scope: local scope and global scope.

**Local Variables:**Local variables are declared within a specific block of code, such as a function, method, or a block enclosed by curly braces while **Global Variables -**are declared outside of any specific block of code, typically at the top of a program or in a separate global variables section.

**NO 3 : Why is initialization of variables required**.

Initialization of variables is required in programming because it assigns an initial value to a variable before it is used in any computations or operations.

**NO 4** **: Differentiate between static, instance and local variables**

**Static Variables** -Static variables have a global scope within their defining block or function.They are accessible throughout the entire program while **Instance Variables** are declared within a class but outside of any methods while **Local Variables** are declared within a block or method and are accessible only within that block or method.

**NO 5** : **Differentiate between widening and narrowing casting in java**.

**Widening Casting** -Widening casting occurs when you convert a value from a smaller data type to a larger data type. It happens automatically by the Java compiler, as it is considered safe and lossless while **Narrowing Casting**  occurs when you convert a value from a larger data type to a smaller data type. It needs to be done explicitly by the programmer because it may result in data loss or precision loss..

**NO 6 :**

|  |  |  |  |
| --- | --- | --- | --- |
| TYPE | SIZE (size in bytes) | DEFAULT | RANGE |
| Boolean | 1bit | false | True,False |
| Char | 2 | =\u0000 | “\0000” to “ \ffff” |
| Byte | 1 | 0 | -128 to 127 |
| Short | 2 | 0 | -2^15 to +2^15-1 |
| Int | 4 | = 0 | -2,147483,648 to 2,147,483647 |
| Long | 8 | 0L | -9,223,372,036,854,775,807 to 9,223,372,036,854,775,807 |
| Float | 4 | 00.0F | +-3.4022347E+38F |
| Double | 8 | = 0.0d | -1.8E+306 to + 1.8E +308 |

**NO 7** **: 7.Define package as used in java programming**

* **In Java programming, a package** is a mechanism used to organize related classes, interfaces, and sub-packages. It provides a way to encapsulate code, manage naming conflicts, and create a hierarchical structure within a project.

**NO 8 : Explain the importance of using Java packages**

* **Organization and Modularity**: Packages help organize code into logical units based on functionality, domain, or purpose. By grouping related classes and resources together, it becomes easier to navigate, locate, and understand code. Packages establish a modular structure that promotes code separation and encapsulation, making it easier to manage and maintain large codebases.
* **Namespace Manag**ement: Packages provide a mechanism for avoiding naming conflicts between classes. Each package creates a separate namespace, ensuring that class names within a package are unique. This allows developers to use meaningful and concise class names without worrying about naming clashes with classes from other packages.
* **Code Reusability:** Packages facilitate code reuse by providing a way to share classes and resources across different projects and modules. By placing reusable code in separate packages, it becomes easier to distribute and reuse them in other applications. Developers can create libraries or frameworks as packages, making it simpler for others to leverage the existing codebase.
* **Access Control and Encapsulation**: Java packages offer access control mechanisms through the use of access modifiers (e.g., public, protected, private). This allows developers to define the visibility and accessibility of classes, methods, and variables within a package. It promotes encapsulation and information hiding, as classes and members that are not explicitly marked as public are hidden from external code, reducing the risk of unintentional modifications or dependencies.
* **Dependency Management:** Packages help manage dependencies between components of an application. By organizing code into packages, developers can define and enforce dependencies between packages, ensuring that high-level modules rely only on the necessary lower-level modules. This promotes modular design, separation of concerns, and reduces coupling between different parts of the codebase.
* **Code Sharing and Collaboration:** Packages facilitate code sharing and collaboration among developers working on the same project. By organizing code into packages, team members can work on different packages simultaneously without conflicts, as long as the interfaces and contracts between packages are well-defined. It allows for parallel development, code versioning, and integration of code changes from multiple developers.

**SECTION 2 Answers**

**NO 1 :**

import java.util.Scanner;

public class SurnameAndAge {

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

// Ask the user to enter their surname

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

String surname = input.nextLine();

// Ask the user to enter their current age

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

int age = input.nextInt()

int surnameLength = surname.length();

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

// Print the results

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

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

}

}

* When you run the program, it will prompt the user to enter their surname and age. After the user provides the inputs, the program will calculate the number of characters in the surname using the length() method of the String class. Then, it will check if the age is even or odd using the modulo operator %. Finally, it will print the results accordingly.

**NO 2 :**

import java.util.Scanner;

public class AverageMarks {

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

// Ask the student to enter the marks of five units

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

double sum = 0;

for (int i = 1; i <= 5; i++) {

System.out.print("Enter marks for unit " + i + ": ");

double marks = input.nextDouble();

sum += marks;

}

// Compute the average

double average = sum / 5;

// Display the average with two decimal places

System.out.printf("The average marks is: %.2f", average);

}

}

* When you run the program, it will prompt the student to enter the marks of five units. It uses a loop to iterate five times, asking for the marks of each unit and accumulating the sum. Then, it calculates the average by dividing the sum by 5. Finally, it uses printf to display the average with two decimal places using the format specifier %.2f.

**NO 3 :**

def check\_divisibility(number):

if number % 2 == 0:

print(f"The number {number} is divisible by 2 because it is even.")

else:

print(f"The number {number} is not divisible by 2.")

if number % 3 == 0:

print(f"The number {number} is divisible by 3 because the sum of its digits is divisible by 3.")

else:

print(f"The number {number} is not divisible by 3.")

if number % 4 == 0:

print(f"The number {number} is divisible by 4 because the last two digits form a number divisible by 4.")

else:

print(f"The number {number} is not divisible by 4.")

if number % 5 == 0:

print(f"The number {number} is divisible by 5 because it ends with a 5 or 0.")

else:

print(f"The number {number} is not divisible by 5.")

if number % 6 == 0:

print(f"The number {number} is divisible by 6 because it is divisible by both 2 and 3.")

else:

print(f"The number {number} is not divisible by 6.")

if number % 7 == 0:

print(f"The number {number} is divisible by 7.")

else:

print(f"The number {number} is not divisible by 7.")

if number % 8 == 0:

print(f"The number {number} is divisible by 8 because the last three digits form a number divisible by 8.")

else:

print(f"The number {number} is not divisible by 8.")

if number % 9 == 0:

print(f"The number {number} is divisible by 9 because the sum of its digits is divisible by 9.")

else:

print(f"The number {number} is not divisible by 9.")

if number % 10 == 0:

print(f"The number {number} is divisible by 10 because it ends with a 0.")

else:

print(f"The number {number} is not divisible by 10.")

# Test the function

num = int(input("Enter a number: "))

check\_divisibility(num)

* In this program, we define the function check\_divisibility which takes an integer number as input. The function checks divisibility for each number in the range of 0-9 by using the modulus operator % to check if the remainder is zero.
* Afterwards, we call the function and input a number to test its divisibility. The program will print whether the given number is divisible by each number in the range of 0-9, providing the corresponding reasons.

**NO 4 :**

public class MultiplesDisplay {

public static void main(String[] args) {

int start = 71;

int end = 150;

System.out.println("Multiples of 2 within the range 71 to 150:");

displayMultiples(start, end, 2);

System.out.println("\nMultiples of 3 within the range 71 to 150:");

displayMultiples(start, end, 3);

System.out.println("\nMultiples of 7 within the range 71 to 150:");

displayMultiples(start, end, 7);

}

public static void displayMultiples(int start, int end, int divisor) {

for (int i = start; i <= end; i++) {

if (i % divisor == 0) {

System.out.print(i + " ");

}

}

System.out.println();

}

}

**No 5 :**

import java.util.Scanner;

public class Calculator {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

double num1 = scanner.nextDouble();

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

String operator = scanner.next();

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

double num2 = scanner.nextDouble();

double result = 0;

switch (operator) {

case "+":

result = num1 + num2;

break;

case "-":

result = num1 - num2;

break;

case "\*":

result = num1 \* num2;

break;

case "/":

result = num1 / num2;

break;

default:

System.out.println("Invalid operator!");

System.exit(0);

}

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

}

}

* The program starts by asking the user to enter the first number, followed by the operator (+, -, \*, /), and then the second number. It then performs the corresponding operation based on the operator entered and displays the result on the screen.