23/02076 AGNES LINDA

JAVA PROGRAMMING ASSIGNMENT.

1. Primitive vs. Reference Data Types:

- **Primitive Types:**
  - Represent basic building blocks of data.
  - Store the actual value directly in memory (stack).
  - Examples: int,double,Boolean,char,byte,short,long.
  - More memory efficient and faster to access.
  - Cannot be used to represent complex objects.
- **Reference Data Types:**
  - Store a memory address (reference) that points to an object in the heap.
  - Objects are created using the ‘new’ keyword and contain attributes (fields) and methods (behaviors).
  - Examples: String,Integer,ArrayList custom classes.
  - Offer greater flexibility for complex data structures.
  - Assigning a reference type variable creates a copy of the reference, not the object itself.

2. Scope of a Variable:

- **Local Variable:**
  - Declared within a method, block, or loop.
  - Accessible only within that specific scope.
  - Created when the method/block is entered and destroyed when it exits.
- **Global Variable (or Instance Variables):**
  - Declared outside any method, but within a class.
  - Accessible from any method within the class (instance).
  - Exist throughout the lifetime of an object.

3. Importance of Variable Initialization:

- Ensures a defined starting value for a variable.
- Prevents unexpected behavior caused by using uninitialized values.

- Improves code readability and maintainability.
- Certain data types (like references) may have default values (null for objects).

4.Differences between Static, Instance, and Local Variables:

- **Static Variables:**
  - Declared with the ‘static’ keyword within a class.
  - Shared by all instances of the class.
  - Used for class-level constants or values shared across all objects.
  - Accessed using the class name (ClassName.staticVariable).
- **Instance Variables (or Global Variables):**
  - Declared within a class but outside any method.
  - Belong to each individual object of the class.
  - Accessed through object references (objectName.instanceVariable).
- **Local Variables:**
  - Declared within a method, block, or loop.
  - Only accessible within that specific scope.

5.Differences between Widening and Narrowing Casting in Java:

- **Widening Casting (Implicit Casting):**
  - Converting a smaller data type to a larger one.
  - No data loss occurs, as the larger type can accommodate the smaller value.
  - Example: int I = 10; long l = I; (int to long)
- **Narrowing Casting (Explicit Casting):**
  - Converting a larger data type to a smaller one.
  - Potential data loss if the larger value doesn't fit within the range of the smaller type.
  - Requires explicit casting using the target data type in parentheses: int j = (int) l; (long to int, possible truncation)

6. Data Type Table:

TYPE	SIZE (IN BYTES)	DEFAULT	RANGE
boolean	1 bit		true,false
Char	2		'\u0000' to '\uffff'
Byte	1	0	
Short	2	0	-2^{15} to $+2^{15}-1$
Int	4	4	-2,147,483,648 to 2,147,483,647
Long	8	0l	-922,337,203,685,477,5808 to 922,337,203,685,477,5807
Float	4	00.0f	+-3.40282347E + 38F (6 -7 significant decimaldigits)
Double	8	0.00d	-1.8E+308 to +1.8E+308

7. Class in Java OOP:

In Java, a **class** acts as a blueprint or template for creating objects. It defines the attributes (properties) and behaviors (methods) that objects of that class will share.

8.Importance of Classes in Java Programming:

- **Object-Oriented Programming (OOP) Foundation:** Classes are the fundamental building blocks of OOP. They promote modularity, reusability, and code organization.
- **Data Abstraction:** Classes encapsulate data, hiding implementation details and allowing controlled access through methods. This promotes data integrity and security.
- **Code Reusability:** By defining a class, you can create multiple objects with the same functionality, reducing code duplication and promoting efficiency.
- **Maintainability:** Classes improve code maintainability by grouping related data and behavior together. Changes made to a class definition are reflected in all objects of that class.
- **Inheritance:** Classes can inherit properties and behaviors from parent classes, promoting code reuse and enabling the creation of more specialized classes.

Example:

Consider a ‘Car’ class. It might have attributes like colour,model and year and methods like startEngine(),accelerate () and brake () . This class defines

the blueprint for creating Car objects, each with its own specific values for attributes but sharing the same set of behaviors.
By using classes, you can effectively model real-world entities and their interactions in your Java programs, leading to well-structured, maintainable, and reusable code.

SECTION 2:

1.

```
public class UserInfo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Get user input
        System.out.print("Enter your surname: ");
        String surname = scanner.nextLine();

        System.out.print("Enter your age: ");
        int age = scanner.nextInt();

        // Output the number of characters in the surname
        int length = surname.length();
        System.out.println("The number of characters in your surname is: " + length);

        // Check if age is even or odd
        if (age % 2 == 0) {
            System.out.println("Your current age is an even number.");
        } else {
            System.out.println("Your current age is an odd number.");
        }
```

```
        scanner.close();
    }
}
```

2.

```
public class AverageMarks {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        double sum = 0;

        // Get marks for five units
        for (int i = 1; i <= 5; i++) {
            System.out.print("Enter marks for unit " + i + ": ");
            double marks = scanner.nextDouble();
            sum += marks;
        }

        // Compute average
        double average = sum / 5;
        System.out.printf("The average marks are: %.2f\n", average);

        scanner.close();
    }
}
```

3.

```
public class DivisibilityCheck {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
```

```
        // Get the number from user
        System.out.print("Enter an integer: ");
        int number = scanner.nextInt();

        // Check divisibility by integers 1-9
        for (int i = 1; i <= 9; i++) {
            if (number % i == 0) {
                if (i == 2 && number % 2 == 0) {
                    System.out.println(number + " is divisible by " + i + " because it is even.");
                } else if (i == 3 && sumOfDigits(number) % 3 == 0) {
                    System.out.println(number + " is divisible by " + i + " because the sum of its digits is divisible by 3.");
                } else if (i == 4 && (number % 100) % 4 == 0) {
                    System.out.println(number + " is divisible by " + i + " because the last two digits form a number divisible by 4.");
                } else if (i == 5 && (number % 10 == 0 || number % 10 == 5)) {
                    System.out.println(number + " is divisible by " + i + " because it ends with a " + (number % 10) + ".");
                } else if (i == 6 && number % 6 == 0) {
                    System.out.println(number + " is divisible by " + i + " because it is divisible by both 2 and 3.");
                } else if (i == 7 && (number * 2) % 7 == 0) {  // Simplified check for 7
                    System.out.println(number + " is divisible by " + i + ".");
```

4.

```
public class Multiples {
    public static void main(String[] args) {
        System.out.println("Multiples of 2, 3, and 7 between 71 and 150:");
```

```
        for (int i = 71; i <= 150; i++) {
            if (i % 2 == 0 || i % 3 == 0 || i % 7 == 0) {
                System.out.println(i);
            }
        }
    }
}
```

5.

```
public class Calculator {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Get the first number
        System.out.print("Enter the first number: ");
        double num1 = scanner.nextDouble();

        // Get the operation
        System.out.print("Enter an operation (+, -, *, /): ");
        char operation = scanner.next().charAt(0);

        // Get the second number
        System.out.print("Enter the second number: ");
        double num2 = scanner.nextDouble();

        // Perform the calculation based on the operation
        double result = 0;
        boolean validOperation = true;
```

```
        switch (operation) {
            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 is not allowed.");
                    validOperation = false;
                }
                break;
            default:
                System.out.println("Error: Invalid operation.");
                validOperation = false;
        }

        // Display the result if the operation was valid
        if (validOperation) {
            System.out.println("The result is: " + result);
        }

        scanner.close();
    }
```

23/02076 AGNES LINDA

}