1. Explain the differences between primitive and reference data types.

1. **Storage Mechanism**:

   - **Primitive Data Types**: Primitive data types are stored directly in the computer's memory. They hold the actual value of the data.

   - **Reference Data Types**: Reference data types are stored as a reference or pointer to the location in memory where the actual data is stored.

2. **Size and Allocation**:

   - **Primitive Data Types**: Primitive data types have a fixed size in memory, which is determined by the programming language. They are allocated on the stack.

   - **Reference Data Types**: Reference data types can have variable sizes, and the memory they occupy is allocated on the heap.

3. **Operations**:

   - **Primitive Data Types**: Operations on primitive data types involve manipulating the actual values stored in memory.

   - **Reference Data Types**: Operations on reference data types involve manipulating the references or pointers to the data stored in memory.

4. **Copying and Passing**:

   - **Primitive Data Types**: When you copy or pass a primitive data type, a new copy of the value is created.

   - **Reference Data Types**: When you copy or pass a reference data type, a new reference or pointer to the same data is created, but the underlying data remains the same.

5. **Examples**:

   - **Primitive Data Types**: Integer, float, char, etc.

   - **Reference Data Types**: String, array, object, class instances, etc.

2. Define the scope of a variable (hint: local and global variable)

1. **Local Scope**:

- A variable with local scope is defined within a specific block of code, such as a function, a loop, or a conditional statement.

- Local variables are only accessible within the block of code where they are defined.

- When the block of code is executed, the local variable is created and allocated memory. When the block of code is no longer being executed, the local variable is destroyed, and its memory is released.

- Local variables have the highest level of scope and are the most commonly used type of variables.

- Examples of local variables include variables declared inside a function, a loop, or an if-else statement.

2. **Global Scope**:

- A variable with global scope is defined outside of any specific block of code, usually at the top or the beginning of a program.

- Global variables are accessible throughout the entire program, including within any local scopes.

- Global variables are allocated memory for the entire duration of the program's execution.

- Global variables can be accessed and modified from anywhere in the program, which can lead to potential issues, such as unintended side effects or data corruption.

- It is generally recommended to minimize the use of global variables and instead use local variables as much as possible, as this can help maintain better code organization, readability, and maintainability.

- Examples of global variables include variables declared at the top of a file or outside of any function or block of code.

3Why is initialization of variables required.

1. **Memory Allocation**: When a variable is declared, the program needs to allocate memory for it. However, this memory may contain random or undefined values. Initializing the variable ensures that the memory location starts with a known and valid value.

2. **Avoiding Unintended Behavior**: If a variable is not initialized, it may contain a random or undefined value. Using this variable without initializing it can lead to unintended behavior, bugs, and unexpected program outcomes. Initializing variables helps ensure that the program starts with a known and expected state.

3. **Readability and Maintainability**: Initializing variables makes the code more readable and easier to understand. It's clearer to the developer and future maintainers of the code what the initial value of the variable is supposed to be.

4. **Preventing Errors**: Uninitialized variables can lead to runtime errors, such as null pointer

exceptions or unexpected results. Initializing variables helps catch these errors early in the development process and makes the code more robust.

5. **Consistent Behavior**: Initializing variables ensures that the program behaves consistently, regardless of the initial state of the memory. This is important for ensuring the correctness and reliability of the program.

6. **Security Considerations**: In some cases, uninitialized variables can contain sensitive or confidential data, which can lead to security vulnerabilities if not properly handled. Initializing variables helps mitigate such risks.

4.Differentiate between static, instance and local variables.

1. **Static Variables**:

- Static variables are associated with the class itself, not with individual instances (objects) of the class.

- They are typically declared using the `static` keyword.

- Static variables are initialized when the class is first loaded into memory and are accessible throughout the entire program.

- Static variables are shared among all instances of the class, and changes to their values affect all instances.

- Static variables are commonly used for global configurations, counters, or values that need to be accessible across multiple instances of a class.

2. **Instance Variables**:

- Instance variables are associated with individual instances (objects) of a class.

- They are typically declared without any special keywords (e.g., `int myVariable;`).

- Instance variables are created when an instance of the class is created and are unique to that instance.

- Instance variables are accessible within the class methods and constructors, and they can have different values for each instance of the class.

- Instance variables are used to store data that is specific to each instance of the class.

3. **Local Variables**:

- Local variables are declared within a specific block of code, such as a method, a loop, or a conditional statement.

- They are only accessible within the block of code where they are defined.

- Local variables are created when the block of code is executed and are destroyed when the block of code is no longer being executed.

- Local variables have the highest level of scope and are the most commonly used type of variables.

- Local variables are used to store temporary or intermediate values within a specific block of code.

5.Differentiate between widening and narrowing casting in java.

1. Widening Casting (Implicit Casting):

- Widening casting is the process of converting a smaller data type to a larger data type.

- This type of casting is done automatically by the Java compiler and is considered safe because the smaller data type can always be represented by the larger data type.

2. Narrowing Casting (Explicit Casting):

- Narrowing casting is the process of converting a larger data type to a smaller data type.

- This type of casting is not done automatically by the Java compiler and requires explicit casting by the programmer.

- Narrowing casting can lead to a loss of precision or data, and it is the programmer's responsibility to ensure that the conversion is safe.

6.the following table shows data type, its size, default value and the range. Filling in the missing values.

TYPE	SIZE (IN BYTES)	DEFAULT	RANGE
boolean	1 bit	false	true, false
Char	2	`\0`	'\0000' to '\ffff'
Byte	1	0	-128 to127
Short	2	0	-2^{15} to $+2^{15}-1$
Int	4	0	-2^31 to 2^31-1
Long	8	0L	- -2^63 to 2^63 - 1
Float	4	00.0f	-3.4E+38 to3.4E+38

Double	8	0.o	-1.8E+308 to +1.8E+308

7.Define class as used in OOP

_Is a set of techniques that aim at representing an information system in terms of object using object-oriented concept

8.Explain the importance of classes in Java programming.

1. **Encapsulation**: Classes allow you to encapsulate data (attributes) and behavior (methods) into a single unit. This helps to hide the implementation details of an object and provides a well-defined interface for interacting with it.

2. **Abstraction**: Classes enable abstraction, which means you can create a simplified model of a real-world object or concept. This abstraction helps to manage complexity and focus on the essential features of an object.

3. **Reusability**: Classes promote code reuse. Once you have defined a class, you can create multiple instances (objects) of that class, each with its own state but sharing the same behavior. This helps to avoid duplicating code and promotes modularity.

4. **Data Hiding**: Classes allow you to control the visibility and accessibility of class members (attributes and methods) using access modifiers such as `public`, `private`, and `protected`. This enables data hiding, which is a fundamental principle of OOP.

5. **Inheritance**: Classes support inheritance, which allows you to create new classes based on existing ones. This promotes code reuse and enables the creation of hierarchical relationships between classes.

6. **Polymorphism**: Classes, in combination with inheritance, enable polymorphism, which allows objects of different classes to be treated as objects of a common superclass. This allows for flexible and dynamic behavior in your applications.

7. **Modularity**: Classes help to organize your code into modular, self-contained units, making it easier to maintain, test, and extend your applications.

8. **Scalability**: As your application grows in complexity, classes provide a structured way to

manage and scale the codebase. They help you to break down the problem into smaller, more manageable parts.

9. **Object-Oriented Design**: Classes are the fundamental building blocks of object-oriented design (OOD) and object-oriented analysis (OOA), which are essential techniques for designing and developing complex software systems.

Section 2:

1. Write a Java program that asks the user to enter their sur name and current age then print the number of characters of their sir name and even or odd depending on their age number.

    Example of Expected result:

    If sir name is Saruni and age is 29, output will be;

    then the number of characters is 6.

    Your current age is an odd number

```
import java.util.Scanner;


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


      // Prompt the user to enter their surname
      System.out.print("Enter your surname: ");
      String surname = scanner.nextLine();


      // Prompt the user to enter their current age
      System.out.print("Enter your current age: ");
      int age = scanner.nextInt();
```

```
        // Calculate the number of characters in the surname
        int numCharacters = surname.length();

        // Determine if the age is even or odd
        String ageDescription = (age % 2 == 0) ? "even" : "odd";

        // Print the results
        System.out.println("The number of characters in your surname is: " +
numCharacters);
        System.out.println("Your current age is an " + ageDescription + " number.");
    }
}
```

2. Write Java program to ask student to enter the marks of the five units they did last semester, compute the average and display it on the screen. (Average should be given in two decimal places).

```
import java.util.Scanner;

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

        // Prompt the user to enter the marks for five units
        System.out.print("Enter the marks for Unit 1: ");
        double unit1 = scanner.nextDouble();

        System.out.print("Enter the marks for Unit 2: ");
        double unit2 = scanner.nextDouble();

        System.out.print("Enter the marks for Unit 3: ");
```

```
        double unit3 = scanner.nextDouble();

        System.out.print("Enter the marks for Unit 4: ");
        double unit4 = scanner.nextDouble();

        System.out.print("Enter the marks for Unit 5: ");
        double unit5 = scanner.nextDouble();

        // Calculate the average
        double average = (unit1 + unit2 + unit3 + unit4 + unit5) / 5;

        // Display the average with two decimal places
        System.out.printf("The average of the five units is: %.2f", average);
    }
}
```

3. Write a program that will help kids learn divisibly test of numbers of integers. The program should check whether the given integer is divisible by integers in the range of 0-9. For example, if a number (955) is divisible by five, the program should print, the number is divisible by 5 because it ends with a 5, and 900 is divisible by 5 because it ends with a 0(zero).

```
import java.util.Scanner;

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

        // Prompt the user to enter an integer
        System.out.print("Enter an integer: ");
        int number = scanner.nextInt();

        // Check divisibility by 0-9
```

```
        for (int i = 0; i <= 9; i++) {
            if (number % i == 0) {
                if (i == 0) {
                    System.out.println("The number is divisible by 0.");
                } else {
                    if (number % 10 == i) {
                        System.out.println("The number is divisible by " + i + " because it ends with " + i +
".");
                    } else if (number / 100 * 100 == number && i == 5) {
                        System.out.println("The number is divisible by " + i + " because it ends with 0.");
                    }
                }
            }
        }
    }
}
```

4. Write a Java program to display all the multiples of 2, 3 and 7 within the range 71 to 150.

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

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

5. Create a calculator using java to help user perform the basic operations (+, -, * and /).

a. User should be asked to enter a number, then an operation, the program computes the operation and display the output to the computer screen.

```
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 operation (+, -, *, /): ");
        char operation = scanner.next().charAt(0);


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


        double result;
        switch (operation) {
            case '+':
                result = num1 + num2;
                System.out.printf("%.2f + %.2f = %.2f", num1, num2, result);
                break;
```

```
            case '-':
                result = num1 - num2;
                System.out.printf("%.2f - %.2f = %.2f", num1, num2, result);
                break;
            case '*':
                result = num1 * num2;
                System.out.printf("%.2f * %.2f = %.2f", num1, num2, result);
                break;
            case '/':
                if (num2 == 0) {
                    System.out.println("Error: Cannot divide by zero.");
                } else {
                    result = num1 / num2;
                    System.out.printf("%.2f / %.2f = %.2f", num1, num2, result);
                }
                break;
            default:
                System.out.println("Error: Invalid operation.");
        }
    }
}
```