Section 1.

1.Reference data types: They include objects and arrays, which are more complex data structures that can hold multiple values. These data types do not directly store the value; instead, they store a reference or pointer to the memory location where the data is stored.

Primitive data types: They include integers (e.g., int, long), floating-point numbers (e.g., float, double), characters (e.g., char), booleans (e.g., boolean), and others. They have a fixed size in memory and are typically used to represent simple values.

2.Global Variable: A global variable is declared outside of any specific block or function, usually at the beginning of a program. It can be accessed and modified from anywhere within the program, including inside functions. The global variable retains its value throughout the entire lifetime of the program.

Local Variable: A local variable is declared within a specific block of code, such as a function or loop. It can only be accessed within that block and is not visible outside of it. Once the block is exited, the local variable's memory is deallocated, and its value is lost.

3.it sets an initial value for a variable before it is used in a program. Without initialization, variables would contain unpredictable and possibly erroneous data, leading to unexpected behavior in the program. Initializing variables ensures that they start with a known and meaningful value, providing a consistent and reliable foundation for the program's execution.

4.Static variables: These are variables that belong to the class rather than an instance of the class. They are shared among all instances of the class and retain their value throughout the program's execution. Static variables are declared using the "static" keyword and are commonly used for values that are common to all instances of the class.

Local variables: These variables are declared inside a method, constructor, or block and are only accessible within that specific scope. They exist only for the duration of the method execution and are not visible outside of it. Local variables are used for temporary storage or calculations within a method.

Instance variables: Also known as member variables, these variables belong to a specific instance of the class. Each instance of the class has its own copy of instance variables, and their values can vary from one instance to another. They are declared within the class but outside any methods or constructors.

5.Widening Casting (Implicit Casting):

It is also known as "upcasting."

This type of casting occurs automatically when you convert a smaller data type to a larger data type.

It does not result in any loss of data or precision because the destination type can hold all possible values of the source type.

Examples: converting an int to a long, a float to a double, or a char to an int.

Narrowing Casting (Explicit Casting):

It is also known as "downcasting."

This type of casting requires manual intervention because it involves converting a larger data type to a smaller data type.

It may lead to loss of data or precision if the destination type cannot accommodate all possible values of the source type.

To perform narrowing casting, you must explicitly cast the source type to the desired destination type.

Example: converting a double to an int, a long to a short, or an int to a byte.

6.TYPE SIZE (IN BYTES) DEFAULT RANGE

boolean 1 bit false true, false

Char 2 '\0000' '\0000' to '\ffff'

Byte 1 0 -128 to 127

Short 2 0 -32,768 to 32,767

Int 4 0 -2,147,483,648 to 2,147,483,647

Long 8 0L -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Float 4 0.0f 1.4E-45 to 3.4028235E38

Double 8 0.0 4.9E-324 to 1.7976931348623157E308

7.In Java programming, a package is a way to organize and group related classes, interfaces, and sub-packages together. It serves as a namespace, preventing naming conflicts and providing a hierarchical structure for organizing code.

By defining classes within a package, you can control the visibility of the classes and their members, using access modifiers like public, private, protected, or package-private (default access).

Packages are declared using the "package" keyword at the beginning of the Java source file, and the directory structure on the file system reflects the package hierarchy. This allows for better code organization and modularity, making it easier to manage and maintain larger projects.

8. a.API Design and Documentation:

Organizing classes into packages helps in defining clear API boundaries. When designing libraries or frameworks, packages serve as a high-level overview of the functionality they provide, making it easier for developers to understand and use the API effectively.

b.Organization and Modularity:

Packages help in organizing code into logical units. By grouping related classes together, it becomes easier to manage large projects and maintain a clear code structure. This promotes modularity, making it simpler to locate, understand, and modify specific components of the application.

c.Naming Conflicts:

When working with multiple libraries or modules, naming conflicts may arise if two classes have the same name. Packages help in resolving such conflicts by providing a unique namespace for classes. As a result, classes with identical names can coexist peacefully as long as they are in different packages.

d.Reusability and Sharing:

By creating reusable code within packages, developers can easily share and distribute functionality across different projects. This promotes code sharing, collaboration, and the creation of libraries that can be used by other developers.

e.Access Control and Visibility:

Packages allow you to control the visibility of classes and members. Using access modifiers like public, protected, private, or package-private, you can define which classes or members can be accessed from outside the package. This helps in enforcing access restrictions and maintaining the integrity of the codebase.

Section2.

1.import java.util.Scanner;

public class SurnameAndAge {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Ask the user to enter their surname

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

String surname = scanner.nextLine();

// Ask the user to enter their current age

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

int age = scanner.nextInt();

// Calculate and print the number of characters in the surname

int numCharacters = surname.length();

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

// Check if the age is even or odd and print the result

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

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

scanner.close();

2.import java.util.Scanner;

public class AverageMarksCalculator {

public static void main(String[] args) {

int numberOfUnits = 5;

double totalMarks = 0;

Scanner scanner = new Scanner(System.in);System.out.println("Enter the marks for each unit:");

// Input marks for each unit

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

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

double marks = scanner.nextDouble();

totalMarks += marks;

}

// Calculate the average

double average = totalMarks / numberOfUnits;

// Display the average with two decimal places

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

scanner.close();

}

}

3.def check\_divisibility(number):

divisors = [2, 3, 4, 5, 6, 7, 8, 9]

for divisor in divisors:

if number % divisor == 0:

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

if divisor == 2 and number % 10 == 0:

print("The number is also divisible by 10 since it ends with a 0 (zero).")

elif divisor == 5 and number % 10 == 0:

print("The number is also divisible by 10 since it ends with a 5.")if number % 10 == 0:

print("The number is divisible by 10 since it ends with a 0 (zero).")

elif number % 5 == 0:

print("The number is divisible by 5 since it ends with a 5.")

else:

print("The number is not divisible by any of the integers in the range of 0-9.")

# Example usage:

number = 955

check\_divisibility(number)

4.public class MultiplesInRange {

public static void main(String[] args) {

int startRange = 71;

int endRange = 150;

System.out.println("Multiples of 2, 3, and 7 within the range 71 to 150 are:");

for (int i = startRange; i <= endRange; i++) {

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

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

}

}

}

}

5.import java.util.Scanner;

public class BasicCalculator {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

double result = 0;

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

double num1 = scanner.nextDouble();

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

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

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

double num2 = scanner.nextDouble();

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: Cannot divide by zero.");

return;

}

break;

default:

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

return;

}

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

}

}