Assignment -2

Q-1) Working with `java.lang.Boolean`

a. Explore the Java API documentation for `java.lang.Boolean` and observe its modifiers and super types.

b. Declare a method-local variable `status` of type `boolean` with the value `true` and convert it to a `String` using the `toString` method. (Hint: Use `Boolean.toString(Boolean)`).

c. Declare a method-local variable `strStatus` of type `String` with the value `"true"` and convert it to a `boolean` using the `parseBoolean` method. (Hint: Use `Boolean.parseBoolean(String)`).

d. Declare a method-local variable `strStatus` of type `String` with the value `"1"` or `"0"` and attempt to convert it to a `boolean`. (Hint: `parseBoolean` method will not work as expected with `"1"` or `"0"`).

e. Declare a method-local variable `status` of type `boolean` with the value `true` and convert it to the corresponding wrapper class using `Boolean.valueOf()`. (Hint: Use `Boolean.valueOf(boolean)`).

f. Declare a method-local variable `strStatus` of type `String` with the value `"true"` and convert it to the corresponding wrapper class using `Boolean.valueOf()`. (Hint: Use `Boolean.valueOf(String)`).

g. Experiment with converting a `boolean` value into other primitive types or vice versa and observe the results.

```
Public class BooleanExample {
    Public static void main(String[] args) {
        // b. Declare a method-local variable status of type boolean with the value true
        // and convert it to a String using the toString method.
        Boolean status = true;
        String statusString = Boolean.toString(status);
        System.out.println(“Boolean to String: “ + statusString); // Output: true
```

```
        // c. Declare a method-local variable strStatus of type String with the value “true”
        // and convert it to a boolean using the parseBoolean method.
        String strStatus = “true”;
        Boolean parsedStatus = Boolean.parseBoolean(strStatus);
        System.out.println(“String to Boolean (true): “ + parsedStatus); // Output: true

        // d. Declare a method-local variable strStatus of type String with the value “1” or “0”
        // and attempt to convert it to a boolean.
        strStatus = “1”; // or “0”
        boolean numericParsedStatus = Boolean.parseBoolean(strStatus);
        System.out.println(“String to Boolean (1): “ + numericParsedStatus); // Output: false

        // e. Declare a method-local variable status of type boolean with the value true
        // and convert it to the corresponding wrapper class using Boolean.valueOf().
        Boolean wrapperStatus = Boolean.valueOf(status);
        System.out.println(“Boolean to Wrapper Class: “ + wrapperStatus); // Output: true

        // f. Declare a method-local variable strStatus of type String with the value “true”
        // and convert it to the corresponding wrapper class using Boolean.valueOf().
        strStatus = “true”;
        Boolean wrapperFromString = Boolean.valueOf(strStatus);
        System.out.println(“String to Wrapper Class: “ + wrapperFromString); // Output: true
```

```
        // g. Experiment with converting a boolean value into other primitive types or vice versa
        // and observe the results.
        Boolean boolValue = true;
        Int intValue = boolValue ? 1 : 0; // Convert boolean to int (1 for true, 0 for false)
        System.out.println("Boolean to int: " + intValue); // Output: 1

        // Convert int back to boolean
        Boolean fromInt = (intValue == 1); // Convert int back to boolean
        System.out.println("Int to Boolean: " + fromInt); // Output: true
    }
}
```

Q-2) Working with `java.lang.Byte`

a. Explore the Java API documentation for `java.lang.Byte` and observe its modifiers and super types.

b. Write a program to test how many bytes are used to represent a `byte` value using the `BYTES` field. (Hint: Use `Byte.BYTES`).

c. Write a program to find the minimum and maximum values of `byte` using the `MIN_VALUE` and `MAX_VALUE` fields. (Hint: Use `Byte.MIN_VALUE` and `Byte.MAX_VALUE`).

d. Declare a method-local variable `number` of type `byte` with some value and convert it to a `String` using the `toString` method. (Hint: Use `Byte.toString(byte)`).

e. Declare a method-local variable `strNumber` of type `String` with some value and convert it to a `byte` value using the `parseByte` method. (Hint: Use `Byte.parseByte(String)`).

f. Declare a method-local variable `strNumber` of type `String` with the value `"Ab12Cd3"` and attempt to convert it to a `byte` value. (Hint: `parseByte` method will throw a `NumberFormatException`).

g. Declare a method-local variable `number` of type `byte` with some value and convert it to the corresponding wrapper class using `Byte.valueOf()`. (Hint: Use `Byte.valueOf(byte)`).

h. Declare a method-local variable `strNumber` of type `String` with some `byte` value and convert it to the corresponding wrapper class using `Byte.valueOf()`. (Hint: Use `Byte.valueOf(String)`).

i. Experiment with converting a `byte` value into other primitive types or vice versa and observe the results.

```
Public class ByteExample {
   Public static void main(String[] args) {
      // b. Write a program to test how many bytes are used to represent a byte value using the BYTES field.
      System.out.println(“Bytes used to represent a byte value: “ + Byte.BYTES); // Output: 1

      // c. Write a program to find the minimum and maximum values of byte using the MIN_VALUE and MAX_VALUE fields.
      System.out.println(“Minimum byte value: “ + Byte.MIN_VALUE); // Output: -128
      System.out.println(“Maximum byte value: “ + Byte.MAX_VALUE); // Output: 127

      // d. Declare a method-local variable number of type byte with some value
      // and convert it to a String using the toString method.
      Byte number = 100;
      String numberString = Byte.toString(number);
      System.out.println(“Byte to String: “ + numberString); // Output: 100
```

```
// e. Declare a method-local variable strNumber of type String with some value
// and convert it to a byte value using the parseByte method.
String strNumber = “25”;
Byte parsedByte = Byte.parseByte(strNumber);
System.out.println(“String to Byte: “ + parsedByte); // Output: 25

// f. Declare a method-local variable strNumber of type String with the value “Ab12Cd3”
// and attempt to convert it to a byte value.
strNumber = “Ab12Cd3”;
try {
   byte invalidByte = Byte.parseByte(strNumber);
   System.out.println(“Parsed Byte: “ + invalidByte);
} catch (NumberFormatException e) {
   System.out.println(“NumberFormatException: Cannot convert ‘” + strNumber + “’ to byte.”); // Expected output
}

// g. Declare a method-local variable number of type byte with some value
// and convert it to the corresponding wrapper class using Byte.valueOf().
Byte wrapperByte = Byte.valueOf(number);
System.out.println(“Byte to Wrapper Class: “ + wrapperByte); // Output: 100

// h. Declare a method-local variable strNumber of type String with some byte value
// and convert it to the corresponding wrapper class using Byte.valueOf().
strNumber = “50”;
Byte wrapperFromString = Byte.valueOf(strNumber);
```

```
        System.out.println("String to Wrapper Class: " + wrapperFromString); // Output: 50


        // i. Experiment with converting a byte value into other primitive types or vice versa.
        // Convert byte to int
        Int intValue = number; // Implicit conversion
        System.out.println("Byte to Int: " + intValue); // Output: 100


        // Convert int back to byte
        Byte byteFromInt = (byte) intValue; // Explicit conversion
        System.out.println("Int to Byte: " + byteFromInt); // Output: 100
    }
}
```

Q-3) Working with java.lang.Short

a. Explore the Java API documentation for java.lang.Short and observe its modifiers and super types.

b. Write a program to test how many bytes are used to represent a short value using the BYTES field. (Hint: Use Short.BYTES).

c. Write a program to find the minimum and maximum values of short using the MIN_VALUE and MAX_VALUE fields. (Hint: Use Short.MIN_VALUE and Short.MAX_VALUE).

D. Declare a method-local variable number of type short with some value and convert it to a String using the toString method. (Hint: Use Short.toString(short)).

e. Declare a method-local variable strNumber of type String with some value and convert it to a short value using the parseShort method. (Hint: Use Short.parseShort(String)).

f. Declare a method-local variable strNumber of type String with the value “Ab12Cd3” and attempt to convert it to a short value. (Hint: parseShort method will throw a NumberFormatException).

g. Declare a method-local variable number of type short with some value and convert it to the corresponding wrapper class using Short.valueOf(). (Hint: Use Short.valueOf(short)).

h. Declare a method-local variable strNumber of type String with some short value and convert it to the corresponding wrapper class using Short.valueOf(). (Hint: Use Short.valueOf(String)).

i. Experiment with converting a short value into other primitive types or vice versa and observe the results

```
public class ShortExample {
    public static void main(String[] args) {
        // b. Write a program to test how many bytes are used to represent a short value using the BYTES field.
        System.out.println(“Bytes used to represent a short value: “ + Short.BYTES); // Output: 2

        // c. Write a program to find the minimum and maximum values of short using the MIN_VALUE and MAX_VALUE fields.
        System.out.println(“Minimum short value: “ + Short.MIN_VALUE); // Output: -32768
        System.out.println(“Maximum short value: “ + Short.MAX_VALUE); // Output: 32767

        // d. Declare a method-local variable number of type short with some value
        // and convert it to a String using the toString method.
        Short number = 12345;
        String numberString = Short.toString(number);
        System.out.println(“Short to String: “ + numberString); // Output: 12345
```

```
// e. Declare a method-local variable strNumber of type String with some value
// and convert it to a short value using the parseShort method.
String strNumber = “200”;
Short parsedShort = Short.parseShort(strNumber);
System.out.println(“String to Short: “ + parsedShort); // Output: 200

// f. Declare a method-local variable strNumber of type String with the value “Ab12Cd3”
// and attempt to convert it to a short value.
strNumber = “Ab12Cd3”;
try {
    short invalidShort = Short.parseShort(strNumber);
    System.out.println(“Parsed Short: “ + invalidShort);
} catch (NumberFormatException e) {
    System.out.println(“NumberFormatException: Cannot convert ‘” + strNumber + “’ to short.”); // Expected output
}

// g. Declare a method-local variable number of type short with some value
// and convert it to the corresponding wrapper class using Short.valueOf().
Short wrapperShort = Short.valueOf(number);
System.out.println(“Short to Wrapper Class: “ + wrapperShort); // Output: 12345

// h. Declare a method-local variable strNumber of type String with some short value
// and convert it to the corresponding wrapper class using Short.valueOf().
```

```
        strNumber = “1500”;
        Short wrapperFromString = Short.valueOf(strNumber);
        System.out.println(“String to Wrapper Class: “ + wrapperFromString); // Output:
1500


        // i. Experiment with converting a short value into other primitive types or vice
versa.
        // Convert short to int
        Int intValue = number; // Implicit conversion
        System.out.println(“Short to Int: “ + intValue); // Output: 12345


        // Convert int back to short
        Short shortFromInt = (short) intValue; // Explicit conversion
        System.out.println(“Int to Short: “ + shortFromInt); // Output: 12345
    }
}
```

Q-4) Working with `java.lang.Integer`

a. Explore the Java API documentation for `java.lang.Integer` and observe its modifiers and super types.

b. Write a program to test how many bytes are used to represent an `int` value using the `BYTES` field. (Hint: Use `Integer.BYTES`).

c. Write a program to find the minimum and maximum values of `int` using the `MIN_VALUE` and `MAX_VALUE` fields. (Hint: Use `Integer.MIN_VALUE` and `Integer.MAX_VALUE`).

d. Declare a method-local variable `number` of type `int` with some value and convert it to a `String` using the `toString` method. (Hint: Use `Integer.toString(int)`).

e. Declare a method-local variable `strNumber` of type `String` with some value and convert it to an `int` value using the `parseInt` method. (Hint: Use `Integer.parseInt(String)`).

f. Declare a method-local variable `strNumber` of type `String` with the value `"Ab12Cd3"` and attempt to convert it to an `int` value. (Hint: `parseInt` method will throw a `NumberFormatException`).

g. Declare a method-local variable `number` of type `int` with some value and convert it to the corresponding wrapper class using `Integer.valueOf()`. (Hint: Use `Integer.valueOf(int)`).

h. Declare a method-local variable `strNumber` of type `String` with some integer value and convert it to the corresponding wrapper class using `Integer.valueOf()`. (Hint: Use `Integer.valueOf(String)`).

i. Declare two integer variables with values `10` and `20`, and add them using a method from the `Integer` class. (Hint: Use `Integer.sum(int, int)`).

j. Declare two integer variables with values `10` and `20`, and find the minimum and maximum values using the `Integer` class. (Hint: Use `Integer.min(int, int)` and `Integer.max(int, int)`).

k. Declare an integer variable with the value `7`. Convert it to binary, octal, and hexadecimal strings using methods from the `Integer` class. (Hint: Use `Integer.toBinaryString(int)`, `Integer.toOctalString(int)`, and `Integer.toHexString(int)`).

l. Experiment with converting an `int` value into other primitive types or vice versa and observe the results.

```
Public class IntegerExample {
    Public static void main(String[] args) {
        // a. Explore the Java API documentation for java.lang.Integer
        // (Refer to the official Java documentation at https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Integer.html)

        // b. Write a program to test how many bytes are used to represent an int value using the BYTES field.
```

```
System.out.println(“Bytes used to represent an int value: “ + Integer.BYTES); // Output: 4


// c. Write a program to find the minimum and maximum values of int using the MIN_VALUE and MAX_VALUE fields.

System.out.println(“Minimum int value: “ + Integer.MIN_VALUE); // Output: -2147483648

System.out.println(“Maximum int value: “ + Integer.MAX_VALUE); // Output: 2147483647


// d. Declare a method-local variable number of type int with some value

// and convert it to a String using the toString method.

Int number = 123456;

String numberString = Integer.toString(number);

System.out.println(“Int to String: “ + numberString); // Output: 123456


// e. Declare a method-local variable strNumber of type String with some value

// and convert it to an int value using the parseInt method.

String strNumber = “7890”;

Int parsedInt = Integer.parseInt(strNumber);

System.out.println(“String to Int: “ + parsedInt); // Output: 7890


// f. Declare a method-local variable strNumber of type String with the value “Ab12Cd3”

// and attempt to convert it to an int value.

strNumber = “Ab12Cd3”;

try {

    int invalidInt = Integer.parseInt(strNumber);
```

```
        System.out.println(“Parsed Int: “ + invalidInt);
    } catch (NumberFormatException e) {
        System.out.println(“NumberFormatException: Cannot convert ‘” + strNumber + “’ to int.”); // Expected output
    }

    // g. Declare a method-local variable number of type int with some value
    // and convert it to the corresponding wrapper class using Integer.valueOf().
    Integer wrapperInt = Integer.valueOf(number);
    System.out.println(“Int to Wrapper Class: “ + wrapperInt); // Output: 123456

    // h. Declare a method-local variable strNumber of type String with some integer value
    // and convert it to the corresponding wrapper class using Integer.valueOf().
    strNumber = “150”;
    Integer wrapperFromString = Integer.valueOf(strNumber);
    System.out.println(“String to Wrapper Class: “ + wrapperFromString); // Output: 150

    // i. Declare two integer variables with values 10 and 20, and add them using a method from the Integer class.
    Int num1 = 10;
    Int num2 = 20;
    Int sum = Integer.sum(num1, num2);
    System.out.println(“Sum of 10 and 20: “ + sum); // Output: 30

    // j. Declare two integer variables with values 10 and 20, and find the minimum and maximum values using the Integer class.
```

```
        Int min = Integer.min(num1, num2);
        Int max = Integer.max(num1, num2);
        System.out.println(“Minimum of 10 and 20: “ + min); // Output: 10
        System.out.println(“Maximum of 10 and 20: “ + max); // Output: 20

        // k. Declare an integer variable with the value 7.
        // Convert it to binary, octal, and hexadecimal strings using methods from the Integer class.
        Int value = 7;
        String binaryString = Integer.toBinaryString(value);
        String octalString = Integer.toOctalString(value);
        String hexString = Integer.toHexString(value);
        System.out.println(“Binary representation of 7: “ + binaryString); // Output: 111
        System.out.println(“Octal representation of 7: “ + octalString); // Output: 7
        System.out.println(“Hexadecimal representation of 7: “ + hexString); // Output: 7

        // l. Experiment with converting an int value into other primitive types or vice versa.
        // Convert int to long
        Long longValue = number; // Implicit conversion
        System.out.println(“Int to Long: “ + longValue); // Output: 123456

        // Convert long back to int
        Int intFromLong = (int) longValue; // Explicit conversion
        System.out.println(“Long to Int: “ + intFromLong); // Output: 123456
    }
}
```

Q-5) Working with `java.lang.Long`

a. Explore the Java API documentation for `java.lang.Long` and observe its modifiers and super types.

b. Write a program to test how many bytes are used to represent a `long` value using the `BYTES` field. (Hint: Use `Long.BYTES`).

c. Write a program to find the minimum and maximum values of `long` using the `MIN_VALUE` and `MAX_VALUE` fields. (Hint: Use `Long.MIN_VALUE` and `Long.MAX_VALUE`).

d. Declare a method-local variable `number` of type `long` with some value and convert it to a `String` using the `toString` method. (Hint: Use `Long.toString(long)`).

e. Declare a method-local variable `strNumber` of type `String` with some value and convert it to a `long` value using the `parseLong` method. (Hint: Use `Long.parseLong(String)`).

f. Declare a method-local variable `strNumber` of type `String` with the value `"Ab12Cd3"` and attempt to convert it to a `long` value. (Hint: `parseLong` method will throw a `NumberFormatException`).

g. Declare a method-local variable `number` of type `long` with some value and convert it to the corresponding wrapper class using `Long.valueOf()`. (Hint: Use `Long.valueOf(long)`).

h. Declare a method-local variable `strNumber` of type `String` with some `long` value and convert it to the corresponding wrapper class using `Long.valueOf()`. (Hint: Use `Long.valueOf(String)`).

i. Declare two long variables with values `1123` and `9845`, and add them using a method from the `Long` class. (Hint: Use `Long.sum(long, long)`).

j. Declare two long variables with values `1122` and `5566`, and find the minimum and maximum values using the `Long` class. (Hint: Use `Long.min(long, long)` and `Long.max(long, long)`).

k. Declare a long variable with the value `7`. Convert it to binary, octal, and hexadecimal strings using methods from the `Long` class. (Hint: Use `Long.toBinaryString(long)`, `Long.toOctalString(long)`, and `Long.toHexString(long)`).

I. Experiment with converting a `long` value into other primitive types or vice versa and observe the results.

```
Public class LongExample {

   Public static void main(String[] args) {

      // a. Explore the Java API documentation for java.lang.Long

      // (Refer to the official Java documentation at https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Long.html)


      // b. Write a program to test how many bytes are used to represent a long value using the BYTES field.

      System.out.println("Bytes used to represent a long value: " + Long.BYTES); // Output: 8


      // c. Write a program to find the minimum and maximum values of long using the MIN_VALUE and MAX_VALUE fields.

      System.out.println("Minimum long value: " + Long.MIN_VALUE); // Output: -9223372036854775808

      System.out.println("Maximum long value: " + Long.MAX_VALUE); // Output: 9223372036854775807


      // d. Declare a method-local variable number of type long with some value

      // and convert it to a String using the toString method.

      Long number = 123456789L;

      String numberString = Long.toString(number);
```

```
System.out.println(“Long to String: “ + numberString); // Output: 123456789


// e. Declare a method-local variable strNumber of type String with some value
// and convert it to a long value using the parseLong method.
String strNumber = “7890123456”;
Long parsedLong = Long.parseLong(strNumber);
System.out.println(“String to Long: “ + parsedLong); // Output: 7890123456


// f. Declare a method-local variable strNumber of type String with the value “Ab12Cd3”
// and attempt to convert it to a long value.
strNumber = “Ab12Cd3”;
try {
    long invalidLong = Long.parseLong(strNumber);
    System.out.println(“Parsed Long: “ + invalidLong);
} catch (NumberFormatException e) {
    System.out.println(“NumberFormatException: Cannot convert ‘” + strNumber + “’ to long.”); // Expected output
}


// g. Declare a method-local variable number of type long with some value
// and convert it to the corresponding wrapper class using Long.valueOf().
Long wrapperLong = Long.valueOf(number);
System.out.println(“Long to Wrapper Class: “ + wrapperLong); // Output: 123456789
```

```
// h. Declare a method-local variable strNumber of type String with some long value
// and convert it to the corresponding wrapper class using Long.valueOf().
strNumber = “15098765432”;
Long wrapperFromString = Long.valueOf(strNumber);
System.out.println(“String to Wrapper Class: “ + wrapperFromString); // Output: 15098765432

// i. Declare two long variables with values 1123 and 9845,
// and add them using a method from the Long class.
Long num1 = 1123L;
Long num2 = 9845L;
Long sum = Long.sum(num1, num2);
System.out.println(“Sum of 1123 and 9845: “ + sum); // Output: 10968

// j. Declare two long variables with values 1122 and 5566,
// and find the minimum and maximum values using the Long class.
Long num3 = 1122L;
Long num4 = 5566L;
Long min = Long.min(num3, num4);
Long max = Long.max(num3, num4);
System.out.println(“Minimum of 1122 and 5566: “ + min); // Output: 1122
System.out.println(“Maximum of 1122 and 5566: “ + max); // Output: 5566
```

```
        // k. Declare a long variable with the value 7.

        // Convert it to binary, octal, and hexadecimal strings using methods from the Long class.

        Long value = 7L;

        String binaryString = Long.toBinaryString(value);

        String octalString = Long.toOctalString(value);

        String hexString = Long.toHexString(value);

        System.out.println("Binary representation of 7: " + binaryString); // Output: 111

        System.out.println("Octal representation of 7: " + octalString); // Output: 7

        System.out.println("Hexadecimal representation of 7: " + hexString); // Output: 7


        // l. Experiment with converting a long value into other primitive types or vice versa.

        // Convert long to int

        Int intValue = (int) number; // Explicit conversion

        System.out.println("Long to Int: " + intValue); // Output: 123456789 (may truncate if long is larger than int range)


        // Convert long to double

        Double doubleValue = number; // Implicit conversion

        System.out.println("Long to Double: " + doubleValue); // Output: 123456789.0

    }

}
```

Q-6) Working with `java.lang.Float`

a. Explore the Java API documentation for `java.lang.Float` and observe its modifiers and super types.

b. Write a program to test how many bytes are used to represent a `float` value using the `BYTES` field. (Hint: Use `Float.BYTES`).

c. Write a program to find the minimum and maximum values of `float` using the `MIN_VALUE` and `MAX_VALUE` fields. (Hint: Use `Float.MIN_VALUE` and `Float.MAX_VALUE`).

d. Declare a method-local variable `number` of type `float` with some value and convert it to a `String` using the `toString` method. (Hint: Use `Float.toString(float)`).

e. Declare a method-local variable `strNumber` of type `String` with some value and convert it to a `float` value using the `parseFloat` method. (Hint: Use `Float.parseFloat(String)`).

f. Declare a method-local variable `strNumber` of type `String` with the value `"Ab12Cd3"` and attempt to convert it to a `float` value. (Hint: `parseFloat` method will throw a `NumberFormatException`).

g. Declare a method-local variable `number` of type `float` with some value and convert it to the corresponding wrapper class using `Float.valueOf()`. (Hint: Use `Float.valueOf(float)`).

h. Declare a method-local variable `strNumber` of type `String` with some `float` value and convert it to the corresponding wrapper class using `Float.valueOf()`. (Hint: Use `Float.valueOf(String)`).

i. Declare two float variables with values `112.3` and `984.5`, and add them using a method from the `Float` class. (Hint: Use `Float.sum(float, float)`).

j. Declare two float variables with values `112.2` and `556.6`, and find the minimum and maximum values using the `Float` class. (Hint: Use `Float.min(float, float)` and `Float.max(float, float)`).

k. Declare a float variable with the value `-25.0f`. Find the square root of this value. (Hint: Use `Math.sqrt()` method).

l. Declare two float variables with the same value, `0.0f`, and divide them. (Hint: Observe the result and any special floating-point behavior).

m. Experiment with converting a `float` value into other primitive types or vice versa and observe the results.

```
Public class FloatExample {

   Public static void main(String[] args) {

      // a. Explore the Java API documentation for java.lang.Float

      // (Refer to the official Java documentation at https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Float.html)


      // b. Write a program to test how many bytes are used to represent a float value using the BYTES field.

      System.out.println("Bytes used to represent a float value: " + Float.BYTES); // Output: 4


      // c. Write a program to find the minimum and maximum values of float using the MIN_VALUE and MAX_VALUE fields.

      System.out.println("Minimum float value: " + Float.MIN_VALUE); // Output: 1.4E-45

      System.out.println("Maximum float value: " + Float.MAX_VALUE); // Output: 3.4028235E38


      // d. Declare a method-local variable number of type float with some value

      // and convert it to a String using the toString method.

      Float number = 123.45f;

      String numberString = Float.toString(number);

      System.out.println("Float to String: " + numberString); // Output: 123.45
```

```
        // e. Declare a method-local variable strNumber of type String with some value
        // and convert it to a float value using the parseFloat method.
        String strNumber = “678.9”;
        Float parsedFloat = Float.parseFloat(strNumber);
        System.out.println(“String to Float: “ + parsedFloat); // Output: 678.9

        // f. Declare a method-local variable strNumber of type String with the value “Ab12Cd3”
        // and attempt to convert it to a float value.
        strNumber = “Ab12Cd3”;
        try {
            float invalidFloat = Float.parseFloat(strNumber);
            System.out.println(“Parsed Float: “ + invalidFloat);
        } catch (NumberFormatException e) {
            System.out.println(“NumberFormatException: Cannot convert ‘” + strNumber + “’ to float.”); // Expected output
        }

        // g. Declare a method-local variable number of type float with some value
        // and convert it to the corresponding wrapper class using Float.valueOf().
        Float wrapperFloat = Float.valueOf(number);
        System.out.println(“Float to Wrapper Class: “ + wrapperFloat); // Output: 123.45

        // h. Declare a method-local variable strNumber of type String with some float value
```

```
        // and convert it to the corresponding wrapper class using Float.valueOf().
        strNumber = “150.75”;
        Float wrapperFromString = Float.valueOf(strNumber);
        System.out.println(“String to Wrapper Class: “ + wrapperFromString); // Output:
150.75


        // i. Declare two float variables with values 112.3 and 984.5,
        // and add them using a method from the Float class.
        Float num1 = 112.3f;
        Float num2 = 984.5f;
        Float sum = Float.sum(num1, num2);
        System.out.println(“Sum of 112.3 and 984.5: “ + sum); // Output: 1096.8


        // j. Declare two float variables with values 112.2 and 556.6,
        // and find the minimum and maximum values using the Float class.
        Float num3 = 112.2f;
        Float num4 = 556.6f;
        Float min = Float.min(num3, num4);
        Float max = Float.max(num3, num4);
        System.out.println(“Minimum of 112.2 and 556.6: “ + min); // Output: 112.2
        System.out.println(“Maximum of 112.2 and 556.6: “ + max); // Output: 556.6


        // k. Declare a float variable with the value -25.0f.
        // Find the square root of this value.
```

```
        Float negativeValue = -25.0f;

        Double sqrtValue = Math.sqrt(negativeValue); // Will return NaN for negative
numbers

        System.out.println(“Square root of -25.0: “ + sqrtValue); // Output: NaN



        // l. Declare two float variables with the same value, 0.0f, and divide them.

        Float zero1 = 0.0f;

        Float zero2 = 0.0f;

        Float divisionResult = zero1 / zero2; // This will result in NaN

        System.out.println(“Result of dividing 0.0f by 0.0f: “ + divisionResult); // Output:
NaN



        // m. Experiment with converting a float value into other primitive types or vice
versa.

        // Convert float to int

        Int intValue = (int) number; // Explicit conversion

        System.out.println(“Float to Int: “ + intValue); // Output: 123



        // Convert float to double

        Double doubleValue = number; // Implicit conversion

        System.out.println(“Float to Double: “ + doubleValue); // Output: 123.45

    }

}
```

Q-7) Working with `java.lang.Double`

a. Explore the Java API documentation for `java.lang.Double` and observe its modifiers and super types.

b. Write a program to test how many bytes are used to represent a `double` value using the `BYTES` field. (Hint: Use `Double.BYTES`).

c. Write a program to find the minimum and maximum values of `double` using the `MIN_VALUE` and `MAX_VALUE` fields. (Hint: Use `Double.MIN_VALUE` and `Double.MAX_VALUE`).

d. Declare a method-local variable `number` of type `double` with some value and convert it to a `String` using the `toString` method. (Hint: Use `Double.toString(double)`).

e. Declare a method-local variable `strNumber` of type `String` with some value and convert it to a `double` value using the `parseDouble` method. (Hint: Use `Double.parseDouble(String)`).

f. Declare a method-local variable `strNumber` of type `String` with the value `"Ab12Cd3"` and attempt to convert it to a `double` value. (Hint: `parseDouble` method will throw a `NumberFormatException`).

g. Declare a method-local variable `number` of type `double` with some value and convert it to the corresponding wrapper class using `Double.valueOf()`. (Hint: Use `Double.valueOf(double)`).

h. Declare a method-local variable `strNumber` of type `String` with some `double` value and convert it to the corresponding wrapper class using `Double.valueOf()`. (Hint: Use `Double.valueOf(String)`).

i. Declare two double variables with values `112.3` and `984.5`, and add them using a method from the `Double` class. (Hint: Use `Double.sum(double, double)`).

j. Declare two double variables with values `112.2` and `556.6`, and find the minimum and maximum values using the `Double` class. (Hint: Use `Double.min(double, double)` and `Double.max(double, double)`).

k. Declare a double variable with the value `-25.0`. Find the square root of this value. (Hint: Use `Math.sqrt()` method).

l. Declare two double variables with the same value, `0.0`, and divide them. (Hint: Observe the result and any special floating-point behavior).

m. Experiment with converting a `double` value into other primitive types or vice versa and observe the results.

```
Public class DoubleExample {

   Public static void main(String[] args) {

      // a. Explore the Java API documentation for java.lang.Double

      // (Refer to the official Java documentation at https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Double.html)


      // b. Write a program to test how many bytes are used to represent a double value using the BYTES field.

      System.out.println("Bytes used to represent a double value: " + Double.BYTES); // Output: 8


      // c. Write a program to find the minimum and maximum values of double using the MIN_VALUE and MAX_VALUE fields.

      System.out.println("Minimum double value: " + Double.MIN_VALUE); // Output: 4.9E-324

      System.out.println("Maximum double value: " + Double.MAX_VALUE); // Output: 1.7976931348623157E308


      // d. Declare a method-local variable number of type double with some value

      // and convert it to a String using the toString method.

      Double number = 123.456;

      String numberString = Double.toString(number);

      System.out.println("Double to String: " + numberString); // Output: 123.456
```

```
        // e. Declare a method-local variable strNumber of type String with some value
        // and convert it to a double value using the parseDouble method.
        String strNumber = “678.9”;
        Double parsedDouble = Double.parseDouble(strNumber);
        System.out.println(“String to Double: “ + parsedDouble); // Output: 678.9

        // f. Declare a method-local variable strNumber of type String with the value “Ab12Cd3”
        // and attempt to convert it to a double value.
        strNumber = “Ab12Cd3”;
        try {
            double invalidDouble = Double.parseDouble(strNumber);
            System.out.println(“Parsed Double: “ + invalidDouble);
        } catch (NumberFormatException e) {
            System.out.println(“NumberFormatException: Cannot convert ‘” + strNumber + “’ to double.”); // Expected output
        }

        // g. Declare a method-local variable number of type double with some value
        // and convert it to the corresponding wrapper class using Double.valueOf().
        Double wrapperDouble = Double.valueOf(number);
        System.out.println(“Double to Wrapper Class: “ + wrapperDouble); // Output: 123.456
```

```
// h. Declare a method-local variable strNumber of type String with some double value

// and convert it to the corresponding wrapper class using Double.valueOf().

strNumber = “150.75”;

Double wrapperFromString = Double.valueOf(strNumber);

System.out.println(“String to Wrapper Class: “ + wrapperFromString); // Output: 150.75


// i. Declare two double variables with values 112.3 and 984.5,

// and add them using a method from the Double class.

Double num1 = 112.3;

Double num2 = 984.5;

Double sum = Double.sum(num1, num2);

System.out.println(“Sum of 112.3 and 984.5: “ + sum); // Output: 1096.8


// j. Declare two double variables with values 112.2 and 556.6,

// and find the minimum and maximum values using the Double class.

Double num3 = 112.2;

Double num4 = 556.6;

Double min = Double.min(num3, num4);

Double max = Double.max(num3, num4);

System.out.println(“Minimum of 112.2 and 556.6: “ + min); // Output: 112.2

System.out.println(“Maximum of 112.2 and 556.6: “ + max); // Output: 556.6
```

```
// k. Declare a double variable with the value -25.0.

// Find the square root of this value.

Double negativeValue = -25.0;

Double sqrtValue = Math.sqrt(negativeValue); // Will return NaN for negative numbers

System.out.println(“Square root of -25.0: “ + sqrtValue); // Output: NaN



// l. Declare two double variables with the same value, 0.0, and divide them.

Double zero1 = 0.0;

Double zero2 = 0.0;

Double divisionResult = zero1 / zero2; // This will result in NaN

System.out.println(“Result of dividing 0.0 by 0.0: “ + divisionResult); // Output: NaN



// m. Experiment with converting a double value into other primitive types or vice versa.

// Convert double to float

Float floatValue = (float) number; // Explicit conversion

System.out.println(“Double to Float: “ + floatValue); // Output: 123.456



// Convert double to int

Int intValue = (int) number; // Explicit conversion

System.out.println(“Double to Int: “ + intValue); // Output: 123



// Convert double to long
```

```
        Long longValue = (long) number; // Explicit conversion

        System.out.println("Double to Long: " + longValue); // Output: 123

    }

}
```

Q-8) Conversion between Primitive Types and Strings

Initialize a variable of each primitive type with a user-defined value and convert it into String:

First, use the toString method of the corresponding wrapper class. (e.g., Integer.toString()).

Then, use the valueOf method of the String class. (e.g., String.valueOf()).

```
Public class PrimitiveToStringConversion {

    Public static void main(String[] args) {

        // Initialize variables of each primitive type with user-defined values

        Int intValue = 42;

        Double doubleValue = 3.14;

        Boolean booleanValue = true;

        Char charValue = 'A';

        Float floatValue = 5.67f;

        Long longValue = 123456789L;

        Short shortValue = 12345;
```

```
        // Convert int to String using Integer.toString() and String.valueOf()
        String intString1 = Integer.toString(intValue);
        String intString2 = String.valueOf(intValue);
        System.out.println("Integer to String using Integer.toString(): " + intString1);
        System.out.println("Integer to String using String.valueOf(): " + intString2);

        // Convert double to String using Double.toString() and String.valueOf()
        String doubleString1 = Double.toString(doubleValue);
        String doubleString2 = String.valueOf(doubleValue);
        System.out.println("Double to String using Double.toString(): " + doubleString1);
        System.out.println("Double to String using String.valueOf(): " + doubleString2);

        // Convert boolean to String using Boolean.toString() and String.valueOf()
        String booleanString1 = Boolean.toString(booleanValue);
        String booleanString2 = String.valueOf(booleanValue);
        System.out.println("Boolean to String using Boolean.toString(): " +
booleanString1);
        System.out.println("Boolean to String using String.valueOf(): " + booleanString2);

        // Convert char to String using Character.toString() and String.valueOf()
        String charString1 = Character.toString(charValue);
        String charString2 = String.valueOf(charValue);
        System.out.println("Character to String using Character.toString(): " + charString1);
        System.out.println("Character to String using String.valueOf(): " + charString2);
```

```
        // Convert float to String using Float.toString() and String.valueOf()
        String floatString1 = Float.toString(floatValue);
        String floatString2 = String.valueOf(floatValue);
        System.out.println("Float to String using Float.toString(): " + floatString1);
        System.out.println("Float to String using String.valueOf(): " + floatString2);

        // Convert long to String using Long.toString() and String.valueOf()
        String longString1 = Long.toString(longValue);
        String longString2 = String.valueOf(longValue);
        System.out.println("Long to String using Long.toString(): " + longString1);
        System.out.println("Long to String using String.valueOf(): " + longString2);

        // Convert short to String using Short.toString() and String.valueOf()
        String shortString1 = Short.toString(shortValue);
        String shortString2 = String.valueOf(shortValue);
        System.out.println("Short to String using Short.toString(): " + shortString1);
        System.out.println("Short to String using String.valueOf(): " + shortString2);
    }
}
```

Q-9) Default Values of Primitive Types

Declare variables of each primitive type as fields of a class and check their default values. (Note: Default values depend on whether the variables are instance variables or static variables)

```
Public class DefaultValues {

    // Instance variables (fields)

    Byte instanceByte;

    Short instanceShort;

    Int instanceInt;

    Long instanceLong;

    Float instanceFloat;

    Double instanceDouble;

    Char instanceChar;

    Boolean instanceBoolean;


    // Static variables

    Static byte staticByte;

    Static short staticShort;

    Static int staticInt;

    Static long staticLong;

    Static float staticFloat;

    Static double staticDouble;

    Static char staticChar;
```

```
Static boolean staticBoolean;


Public void displayInstanceDefaults() {
    System.out.println(“Default values of instance variables:”);
    System.out.println(“byte: “ + instanceByte);
    System.out.println(“short: “ + instanceShort);
    System.out.println(“int: “ + instanceInt);
    System.out.println(“long: “ + instanceLong);
    System.out.println(“float: “ + instanceFloat);
    System.out.println(“double: “ + instanceDouble);
    System.out.println(“char: “ + instanceChar);
    System.out.println(“boolean: “ + instanceBoolean);
}


Public static void displayStaticDefaults() {
    System.out.println(“Default values of static variables:”);
    System.out.println(“byte: “ + staticByte);
    System.out.println(“short: “ + staticShort);
    System.out.println(“int: “ + staticInt);
    System.out.println(“long: “ + staticLong);
    System.out.println(“float: “ + staticFloat);
    System.out.println(“double: “ + staticDouble);
    System.out.println(“char: “ + staticChar);
```

```
        System.out.println("boolean: " + staticBoolean);
    }

    Public static void main(String[] args) {
        // Create an instance of DefaultValues to access instance variables
        DefaultValues defaultValues = new DefaultValues();

        // Display instance defaults
        defaultValues.displayInstanceDefaults();

        // Display static defaults
        displayStaticDefaults();
    }
}
```

Q-10) Arithmetic Operations with Command Line Input

Write a program that accepts two integers and an arithmetic operator (`+`, `-`, `*`, `/`) from the command line. Perform the specified arithmetic operation based on the operator provided. (Hint: Use `switch-case` for operations).

```
Public class ArithmeticOperations {
    Public static void main(String[] args) {
        // Check if there are enough arguments
        If (args.length != 3) {
            System.out.println("Usage: java ArithmeticOperations <num1> <operator> <num2>");
```

```
    System.exit(1);
}

// Parse the first and third arguments as integers
Int num1 = Integer.parseInt(args[0]);
String operator = args[1];
Int num2 = Integer.parseInt(args[2]);
Int result;

// Perform the specified arithmetic operation using switch-case
Switch (operator) {
    Case “+”:
        Result = num1 + num2;
        System.out.println(“Result: “ + num1 + “ + “ + num2 + “ = “ + result);
        Break;
    Case “-“:
        Result = num1 – num2;
        System.out.println(“Result: “ + num1 + “ – “ + num2 + “ = “ + result);
        Break;
    Case “*”:
        Result = num1 * num2;
        System.out.println(“Result: “ + num1 + “ * “ + num2 + “ = “ + result);
        Break;
```

```
            Case "/":
                // Handle division by zero
                If (num2 != 0) {
                    Result = num1 / num2;
                    System.out.println("Result: " + num1 + " / " + num2 + " = " + result);
                } else {
                    System.out.println("Error: Division by zero is not allowed.");
                }
                Break;
            Default:
                System.out.println("Error: Invalid operator. Please use +, -, *, or /.");
                Break;
        }
    }
}
```

Assignment -4

Q-1) Loan Amortization Calculator

Implement a system to calculate and display the monthly payments for a mortgage loan. The system should:

Accept the principal amount (loan amount), annual interest rate, and loan term (in years) from the user.

Calculate the monthly payment using the standard mortgage formula:

Monthly Payment Calculation:

monthlyPayment = principal * (monthlyInterestRate * (1 + monthlyInterestRate)^(numberOfMonths)) / ((1 + monthlyInterestRate)^(numberOfMonths) – 1)

Where monthlyInterestRate = annualInterestRate / 12 / 100 and numberOfMonths = loanTerm * 12

Note: Here ^ means power and to find it you can use Math.pow( ) method

Display the monthly payment and the total amount paid over the life of the loan, in Indian Rupees (₹).

Define the class LoanAmortizationCalculator with fields, an appropriate constructor, getter and setter methods, a toString method and business logic methods. Define the class LoanAmortizationCalculatorUtil with methods acceptRecord, printRecord, and menuList. Define the class Program with a main method and test the functionality of the utility class.

```
Public class LoanAmortizationCalculator {
    Private double principal;
    Private double annualInterestRate;
    Private int loanTerm;
```

```
    Public LoanAmortizationCalculator(double principal, double annualInterestRate, int
loanTerm) {
        This.principal = principal;
        This.annualInterestRate = annualInterestRate;
        This.loanTerm = loanTerm;
    }

    Public double getPrincipal() { return principal; }
    Public void setPrincipal(double principal) { this.principal = principal; }

    Public double getAnnualInterestRate() { return annualInterestRate; }
    Public void setAnnualInterestRate(double annualInterestRate) {
this.annualInterestRate = annualInterestRate; }

    Public int getLoanTerm() { return loanTerm; }
    Public void setLoanTerm(int loanTerm) { this.loanTerm = loanTerm; }

    Public double calculateMonthlyPayment() {
        Double monthlyInterestRate = annualInterestRate / 12 / 100;
        Int numberOfMonths = loanTerm * 12;
        Return principal * (monthlyInterestRate * Math.pow(1 + monthlyInterestRate,
numberOfMonths)) /
            (Math.pow(1 + monthlyInterestRate, numberOfMonths) – 1);
    }

    Public double calculateTotalPayment() {
        Return calculateMonthlyPayment() * loanTerm * 12;
```

```
    }

    @Override
    Public String toString() {
        Return “Principal: ₹” + principal + “, Annual Interest Rate: “ + annualInterestRate + “%, Loan Term: “ + loanTerm + “ years”;
    }
}

Import java.util.Scanner;

Public class LoanAmortizationCalculatorUtil {
    Private LoanAmortizationCalculator loan;

    Public void acceptRecord() {
        Scanner scanner = new Scanner(System.in);
        System.out.print(“Enter Principal Amount: “);
        Double principal = scanner.nextDouble();
        System.out.print(“Enter Annual Interest Rate (%): “);
        Double interestRate = scanner.nextDouble();
        System.out.print(“Enter Loan Term (years): “);
        Int loanTerm = scanner.nextInt();
        Loan = new LoanAmortizationCalculator(principal, interestRate, loanTerm);
    }

    Public void printRecord() {
```

```
        System.out.println(loan);
        System.out.printf("Monthly Payment: ₹%.2f%n", loan.calculateMonthlyPayment());
        System.out.printf("Total Payment: ₹%.2f%n", loan.calculateTotalPayment());
    }

    Public void menuList() {
        Scanner scanner = new Scanner(System.in);
        Int choice;
        Do {
            System.out.println("1. Input Loan Details");
            System.out.println("2. Display Loan Details and Payments");
            System.out.println("3. Exit");
            Choice = scanner.nextInt();
            Switch (choice) {
                Case 1 -> acceptRecord();
                Case 2 -> printRecord();
                Case 3 -> System.out.println("Exiting…");
                Default -> System.out.println("Invalid choice.");
            }
        } while (choice != 3);
    }
}
```

```
Public class Program {
    Public static void main(String[] args) {
        LoanAmortizationCalculatorUtil util = new LoanAmortizationCalculatorUtil();
        Util.menuList();
    }
}
```

Q-2) Compound Interest Calculator for Investment

Develop a system to compute the future value of an investment with compound interest. The system should:

Accept the initial investment amount, annual interest rate, number of times the interest is compounded per year, and investment duration (in years) from the user.

Calculate the future value of the investment using the formula:

Future Value Calculation:

futureValue = principal * (1 + annualInterestRate / numberOfCompounds)^(numberOfCompounds * years)

Total Interest Earned: totalInterest = futureValue – principal

Display the future value and the total interest earned, in Indian Rupees (₹).

Define the class CompoundInterestCalculator with fields, an appropriate constructor, getter and setter methods, a toString method and business logic methods. Define the class CompoundInterestCalculatorUtil with methods acceptRecord, printRecord, and menuList. Define the class Program with a main method to test the functionality of the utility class.

```
Public class CompoundInterestCalculator {
    Private double principal;
    Private double annualInterestRate;
    Private int numberOfCompounds;
    Private int years;

    Public CompoundInterestCalculator(double principal, double annualInterestRate, int
numberOfCompounds, int years) {
        This.principal = principal;
        This.annualInterestRate = annualInterestRate;
        This.numberOfCompounds = numberOfCompounds;
        This.years = years;
    }

    Public double getPrincipal() { return principal; }
    Public void setPrincipal(double principal) { this.principal = principal; }

    Public double getAnnualInterestRate() { return annualInterestRate; }
    Public void setAnnualInterestRate(double annualInterestRate) {
this.annualInterestRate = annualInterestRate; }

    Public int getNumberOfCompounds() { return numberOfCompounds; }
    Public void setNumberOfCompounds(int numberOfCompounds) {
this.numberOfCompounds = numberOfCompounds; }
```

```
    Public int getYears() { return years; }
    Public void setYears(int years) { this.years = years; }

    Public double calculateFutureValue() {
        Return principal * Math.pow(1 + annualInterestRate / numberOfCompounds,
numberOfCompounds * years);
    }

    Public double calculateTotalInterest() {
        Return calculateFutureValue() – principal;
    }

    @Override
    Public String toString() {
        Return “Principal: ₹” + principal + “, Annual Interest Rate: “ + annualInterestRate +
“%, Duration: “ + years + “ years”;
    }
}



Import java.util.Scanner;

Public class CompoundInterestCalculatorUtil {
    Private CompoundInterestCalculator calculator;

    Public void acceptRecord() {
        Scanner scanner = new Scanner(System.in);
```

```
        System.out.print(“Enter Initial Investment: “);
        Double principal = scanner.nextDouble();
        System.out.print(“Enter Annual Interest Rate (%): “);
        Double interestRate = scanner.nextDouble();
        System.out.print(“Enter Number of Times Interest is Compounded per Year: “);
        Int compounds = scanner.nextInt();
        System.out.print(“Enter Investment Duration (years): “);
        Int years = scanner.nextInt();
        Calculator = new CompoundInterestCalculator(principal, interestRate, compounds, years);
    }

    Public void printRecord() {
        System.out.println(calculator);
        System.out.printf(“Future Value: ₹%.2f%n”, calculator.calculateFutureValue());
        System.out.printf(“Total Interest Earned: ₹%.2f%n”, calculator.calculateTotalInterest());
    }

    Public void menuList() {
        Scanner scanner = new Scanner(System.in);
        Int choice;
        Do {
            System.out.println(“1. Input Investment Details”);
            System.out.println(“2. Display Future Value and Interest”);
            System.out.println(“3. Exit”);
            Choice = scanner.nextInt();
```

```
            Switch (choice) {
                Case 1 -> acceptRecord();
                Case 2 -> printRecord();
                Case 3 -> System.out.println("Exiting…");
                Default -> System.out.println("Invalid choice.");
            }
        } while (choice != 3);
    }
}

Public class Program {
    Public static void main(String[] args) {
        CompoundInterestCalculatorUtil util = new CompoundInterestCalculatorUtil();
        Util.menuList();
    }
}
```

Q-3) BMI (Body Mass Index) Tracker

Create a system to calculate and classify Body Mass Index (BMI). The system should:

Accept weight (in kilograms) and height (in meters) from the user.

Calculate the BMI using the formula:

BMI Calculation: BMI = weight / (height * height)

Classify the BMI into one of the following categories:

Underweight: BMI < 18.5

Normal weight: 18.5 ≤ BMI < 24.9

Overweight: 25 ≤ BMI < 29.9

Obese: BMI ≥ 30

Display the BMI value and its classification.

Define the class BMITracker with fields, an appropriate constructor, getter and setter methods, a toString method, and business logic methods. Define the class BMITrackerUtil with methods acceptRecord, printRecord, and menuList. Define the class Program with a main method to test the functionality of the utility class.

```
Public class BMITracker {
    Private double weight;
    Private double height;
    Private double bmi;

    Public BMITracker(double weight, double height) {
        This.weight = weight;
        This.height = height;
        This.bmi = calculateBMI();  // Automatically calculate BMI when the object is created
    }

    // Getter and Setter methods
    Public double getWeight() { return weight; }
    Public void setWeight(double weight) {
        This.weight = weight;
        This.bmi = calculateBMI();  // Recalculate BMI when weight is updated
```

```
}

Public double getHeight() { return height; }
Public void setHeight(double height) {
    This.height = height;
    This.bmi = calculateBMI();  // Recalculate BMI when height is updated
}

Public double getBMI() { return bmi; }

// Method to calculate BMI
Private double calculateBMI() {
    Return weight / (height * height);
}

// Method to classify the BMI value
Public String classifyBMI() {
    If (bmi < 18.5) {
        Return “Underweight”;
    } else if (bmi >= 18.5 && bmi < 24.9) {
        Return “Normal weight”;
    } else if (bmi >= 25 && bmi < 29.9) {
        Return “Overweight”;
    } else {
        Return “Obese”;
    }
```

```
    }

    @Override
    Public String toString() {
        Return String.format(“Weight: %.2f kg, Height: %.2f meters, BMI: %.2f (%s)”,
                        Weight, height, bmi, classifyBMI());
    }
}



Import java.util.Scanner;

Public class BMITrackerUtil {
    Private BMITracker bmiTracker;

    // Accepts weight and height from the user and creates a BMITracker object
    Public void acceptRecord() {
        Scanner scanner = new Scanner(System.in);
        System.out.print(“Enter weight (in kilograms): “);
        Double weight = scanner.nextDouble();
        System.out.print(“Enter height (in meters): “);
        Double height = scanner.nextDouble();
        bmiTracker = new BMITracker(weight, height);  // Create BMITracker object with
input values
    }
```

```
    // Prints the BMI details (value and classification)
    Public void printRecord() {
        If (bmiTracker != null) {
            System.out.println(bmiTracker.toString());
        } else {
            System.out.println("No BMI record available. Please enter the weight and height first.");
        }
    }

    // Menu to allow user interaction
    Public void menuList() {
        Scanner scanner = new Scanner(System.in);
        Int choice;
        Do {
            System.out.println("\n--- BMI Tracker Menu ---");
            System.out.println("1. Input Weight and Height");
            System.out.println("2. Display BMI and Classification");
            System.out.println("3. Exit");
            System.out.print("Choose an option: ");
            Choice = scanner.nextInt();

            Switch (choice) {
                Case 1 -> acceptRecord();
                Case 2 -> printRecord();
                Case 3 -> System.out.println("Exiting BMI Tracker…");
```

```
            Default -> System.out.println("Invalid choice, please try again.");
        }
    } while (choice != 3);
  }
}
```

```
Public class Program {
  Public static void main(String[] args) {
    BMITrackerUtil util = new BMITrackerUtil();
    Util.menuList();  // Display the menu and let user interact with BMI tracker
  }
}
```

Sample output

```
--- BMI Tracker Menu ---
1. Input Weight and Height
2. Display BMI and Classification
3. Exit
Choose an option: 1
Enter weight (in kilograms): 75
Enter height (in meters): 1.75
```

--- BMI Tracker Menu ---

1. Input Weight and Height

2. Display BMI and Classification

3. Exit

Choose an option: 2

Weight: 75.00 kg, Height: 1.75 meters, BMI: 24.49 (Normal weight)

--- BMI Tracker Menu ---

1. Input Weight and Height

2. Display BMI and Classification

3. Exit

Choose an option: 3

Exiting BMI Tracker…

==========

Q-4. Discount Calculation for Retail Sales

Design a system to calculate the final price of an item after applying a discount. The system should:

Accept the original price of an item and the discount percentage from the user.

Calculate the discount amount and the final price using the following formulas:

Discount Amount Calculation: discountAmount = originalPrice * (discountRate / 100)

Final Price Calculation: finalPrice = originalPrice – discountAmount

Display the discount amount and the final price of the item, in Indian Rupees (₹).

Define the class DiscountCalculator with fields, an appropriate constructor, getter and setter methods, a toString method, and business logic methods. Define the class DiscountCalculatorUtil with methods acceptRecord, printRecord, and menuList. Define the class Program with a main method to test the functionality of the utility class.

```
Public class DiscountCalculator {
    Private double originalPrice;
    Private double discountRate;

    Public DiscountCalculator(double originalPrice, double discountRate) {
        This.originalPrice = originalPrice;
        This.discountRate = discountRate;
    }

    // Getter and Setter methods
    Public double getOriginalPrice() { return originalPrice; }
    Public void setOriginalPrice(double originalPrice) { this.originalPrice = originalPrice; }

    Public double getDiscountRate() { return discountRate; }
    Public void setDiscountRate(double discountRate) { this.discountRate = discountRate; }

    // Method to calculate the discount amount
```

```
    Public double calculateDiscountAmount() {
        Return originalPrice * (discountRate / 100);
    }

    // Method to calculate the final price after applying the discount
    Public double calculateFinalPrice() {
        Return originalPrice – calculateDiscountAmount();
    }

    @Override
    Public String toString() {
        Return String.format(“Original Price: ₹%.2f, Discount Rate: %.2f%%, Discount
Amount: ₹%.2f, Final Price: ₹%.2f”,
                originalPrice, discountRate, calculateDiscountAmount(),
calculateFinalPrice());
    }
}



Import java.util.Scanner;

Public class DiscountCalculatorUtil {
    Private DiscountCalculator discountCalculator;

    // Accepts original price and discount rate from the user and creates a
DiscountCalculator object
    Public void acceptRecord() {
```

```
        Scanner scanner = new Scanner(System.in);
        System.out.print(“Enter the original price of the item (₹): “);
        Double originalPrice = scanner.nextDouble();
        System.out.print(“Enter the discount rate (%): “);
        Double discountRate = scanner.nextDouble();
        discountCalculator = new DiscountCalculator(originalPrice, discountRate);  // Create DiscountCalculator object
    }

    // Prints the discount amount and the final price of the item
    Public void printRecord() {
        If (discountCalculator != null) {
            System.out.println(discountCalculator.toString());
        } else {
            System.out.println(“No record available. Please enter the original price and discount rate first.”);
        }
    }

    // Menu to allow user interaction
    Public void menuList() {
        Scanner scanner = new Scanner(System.in);
        Int choice;
        Do {
            System.out.println(“\n--- Discount Calculator Menu ---“);
            System.out.println(“1. Input Original Price and Discount Rate”);
            System.out.println(“2. Display Discount Amount and Final Price”);
```

```
            System.out.println(“3. Exit”);
            System.out.print(“Choose an option: “);
            Choice = scanner.nextInt();

            Switch (choice) {
                Case 1 -> acceptRecord();
                Case 2 -> printRecord();
                Case 3 -> System.out.println(“Exiting Discount Calculator…”);
                Default -> System.out.println(“Invalid choice, please try again.”);
            }
        } while (choice != 3);
    }
}


Public class Program {
    Public static void main(String[] args) {
        DiscountCalculatorUtil util = new DiscountCalculatorUtil();
        Util.menuList();  // Display the menu and let user interact with Discount Calculator
    }
}
```

==========

Q-5) **Toll Booth Revenue Management**

Develop a system to simulate a toll booth for collecting revenue. The system should:

1. Allow the user to set toll rates for different vehicle types: Car, Truck, and Motorcycle.
2. Accept the number of vehicles of each type passing through the toll booth.
3. Calculate the total revenue based on the toll rates and number of vehicles.
4. Display the total number of vehicles and the total revenue collected, in Indian Rupees (₹).

- **Toll Rate Examples:**
  - Car: ₹50.00
  - Truck: ₹100.00
  - Motorcycle: ₹30.00

Define the class `TollBoothRevenueManager` with fields, an appropriate constructor, getter and setter methods, a `toString` method, and business logic methods. Define the class `TollBoothRevenueManagerUtil` with methods `acceptRecord`, `printRecord`, and `menuList`. Define the class `Program` with a `main` method to test the functionality of the utility class.

```
Public class TollBoothRevenueManager {
    Private double carRate;
    Private double truckRate;
    Private double motorcycleRate;
    Private int carCount;
```

```
    Private int truckCount;
    Private int motorcycleCount;

    Public TollBoothRevenueManager() {
        // Setting default toll rates
        This.carRate = 50.00;
        This.truckRate = 100.00;
        This.motorcycleRate = 30.00;
        This.carCount = 0;
        This.truckCount = 0;
        This.motorcycleCount = 0;
    }

    // Getter and Setter methods
    Public double getCarRate() { return carRate; }
    Public void setCarRate(double carRate) { this.carRate = carRate; }

    Public double getTruckRate() { return truckRate; }
    Public void setTruckRate(double truckRate) { this.truckRate = truckRate; }

    Public double getMotorcycleRate() { return motorcycleRate; }
    Public void setMotorcycleRate(double motorcycleRate) { this.motorcycleRate =
motorcycleRate; }

    Public int getCarCount() { return carCount; }
    Public void setCarCount(int carCount) { this.carCount = carCount; }
```

```
    Public int getTruckCount() { return truckCount; }
    Public void setTruckCount(int truckCount) { this.truckCount = truckCount; }

    Public int getMotorcycleCount() { return motorcycleCount; }
    Public void setMotorcycleCount(int motorcycleCount) { this.motorcycleCount =
motorcycleCount; }

    // Method to calculate total revenue
    Public double calculateTotalRevenue() {
        Return (carCount * carRate) + (truckCount * truckRate) + (motorcycleCount *
motorcycleRate);
    }

    // Method to calculate total number of vehicles
    Public int calculateTotalVehicles() {
        Return carCount + truckCount + motorcycleCount;
    }

    @Override
    Public String toString() {
        Return String.format("Total Vehicles: %d, Total Revenue: ₹%.2f",
                        calculateTotalVehicles(), calculateTotalRevenue());
    }
}

Import java.util.Scanner;
```

```
Public class TollBoothRevenueManagerUtil {
    Private TollBoothRevenueManager tollBooth;

    // Constructor to initialize TollBoothRevenueManager
    Public TollBoothRevenueManagerUtil() {
        tollBooth = new TollBoothRevenueManager();  // Create an instance of
TollBoothRevenueManager
    }

    // Accepts the number of vehicles of each type from the user
    Public void acceptRecord() {
        Scanner scanner = new Scanner(System.in);

        System.out.print(“Enter number of Cars: “);
        tollBooth.setCarCount(scanner.nextInt());

        System.out.print(“Enter number of Trucks: “);
        tollBooth.setTruckCount(scanner.nextInt());

        System.out.print(“Enter number of Motorcycles: “);
        tollBooth.setMotorcycleCount(scanner.nextInt());
    }

    // Prints the total number of vehicles and the total revenue collected
    Public void printRecord() {
```

```
        System.out.println(tollBooth.toString());
    }

    // Menu to allow user interaction
    Public void menuList() {
        Scanner scanner = new Scanner(System.in);
        Int choice;
        Do {
            System.out.println("\n--- Toll Booth Revenue Management Menu ---");
            System.out.println("1. Input Number of Vehicles");
            System.out.println("2. Display Total Revenue and Vehicle Count");
            System.out.println("3. Exit");
            System.out.print("Choose an option: ");
            Choice = scanner.nextInt();

            Switch (choice) {
                Case 1 -> acceptRecord();
                Case 2 -> printRecord();
                Case 3 -> System.out.println("Exiting Toll Booth Revenue Management…");
                Default -> System.out.println("Invalid choice, please try again.");
            }
        } while (choice != 3);
    }
}

Public class Program {
```

```
    Public static void main(String[] args) {
        TollBoothRevenueManagerUtil util = new TollBoothRevenueManagerUtil();
        Util.menuList();  // Display the menu and let user interact with the Toll Booth Revenue Manager
    }
}
```

Sample output

```
--- Toll Booth Revenue Management Menu ---
1. Input Number of Vehicles
2. Display Total Revenue and Vehicle Count
3. Exit
Choose an option: 1
Enter number of Cars: 10
Enter number of Trucks: 5
Enter number of Motorcycles: 7

--- Toll Booth Revenue Management Menu ---
1. Input Number of Vehicles
2. Display Total Revenue and Vehicle Count
3. Exit
Choose an option: 2
Total Vehicles: 22, Total Revenue: ₹800.00

--- Toll Booth Revenue Management Menu ---
1. Input Number of Vehicles
```

2. Display Total Revenue and Vehicle Count

3. Exit

Choose an option: 3

Exiting Toll Booth Revenue Management…

Assignment -5

Q-1) Design and implement a class named InstanceCounter to track and count the number of instances created from this class.

```
Public class InstanceCounter {
    Private static int instanceCount = 0;

    // Constructor
    Public InstanceCounter() {
        instanceCount++;
    }
```

```
    // Static method to get instance count
    Public static int getInstanceCount() {
        Return instanceCount;
    }
}
```

Q-2) Design and implement a class named Logger to manage logging messages for an application. The class should be implemented as a singleton to ensure that only one instance of the Logger exists throughout the application.

The class should Include the following methods:

getInstance(): Returns the unique instance of the Logger class.

Log(String message): Adds a log message to the logger.

getLog(): Returns the current log messages as a String.

clearLog(): Clears all log messages.

```
Public class Logger {
    Private static Logger loggerInstance = null;
    Private StringBuilder logMessages = new StringBuilder();

    // Private constructor to prevent instantiation
```

```
    Private Logger() {}

    // Static method to get the single instance (Singleton pattern)
    Public static Logger getInstance() {
        If (loggerInstance == null) {
            loggerInstance = new Logger();
        }
        Return loggerInstance;
    }

    // Method to log a message
    Public void log(String message) {
        logMessages.append(message).append("\n");
    }

    // Method to get all log messages
    Public String getLog() {
        Return logMessages.toString();
    }

    // Method to clear log messages
    Public void clearLog() {
        logMessages.setLength(0);
    }
}
```

Q-3) Design and implement a class named Employee to manage employee data for a company. The class should include fields to keep track of the total number of employees and the total salary expense, as well as individual employee details such as their ID, name, and salary.

The class should have methods to:

Retrieve the total number of employees (getTotalEmployees())

Apply a percentage raise to the salary of all employees (applyRaise(double percentage))

Calculate the total salary expense, including any raises (calculateTotalSalaryExpense())

Update the salary of an individual employee (updateSalary(double newSalary))

Understand the problem statement and use static and non-static fields and methods appropriately. Implement static and non-static initializers, constructors, getter and setter methods, and a toString() method to handle the initialization and representation of employee data.

Write a menu-driven program in the main method to test the functionalities.

```
Public class Employee {
    Private static int totalEmployees = 0;
    Private static double totalSalaryExpense = 0.0;
```

```
Private int employeeID;
Private String employeeName;
Private double salary;

// Static initializer block to initialize static fields
Static {
    totalEmployees = 0;
    totalSalaryExpense = 0.0;
}

// Constructor to initialize employee details
Public Employee(int employeeID, String employeeName, double salary) {
    This.employeeID = employeeID;
    This.employeeName = employeeName;
    This.salary = salary;
    totalEmployees++;
    totalSalaryExpense += salary;
}

// Method to get the total number of employees
Public static int getTotalEmployees() {
    Return totalEmployees;
}

// Method to apply a raise to all employees' salaries
Public static void applyRaise(double percentage) {
```

```
    totalSalaryExpense += totalSalaryExpense * (percentage / 100);
}

// Method to calculate the total salary expense
Public static double calculateTotalSalaryExpense() {
    Return totalSalaryExpense;
}

// Method to update the salary of an individual employee
Public void updateSalary(double newSalary) {
    totalSalaryExpense -= this.salary;  // Subtract old salary
    this.salary = newSalary;
    totalSalaryExpense += newSalary;    // Add new salary
}

// Getter for Employee details
Public int getEmployeeID() {
    Return employeeID;
}

Public String getEmployeeName() {
    Return employeeName;
}

Public double getSalary() {
    Return salary;
```

```
    }

    // toString method for displaying employee details
    @Override
    Public String toString() {
        Return “Employee ID: “ + employeeID + “, Name: “ + employeeName + “, Salary: $” + salary;
    }
}
```

==========

Main.java (Menu-Driven Program)

```
Import java.util.Scanner;

Public class Main {
    Public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Logger logger = Logger.getInstance();

        // Array to store employees
```

```
Employee[] employees = new Employee[100];
Int employeeCount = 0;

While (true) {
    System.out.println(“\nMenu:”);
    System.out.println(“1. Create an Employee”);
    System.out.println(“2. View Total Employees”);
    System.out.println(“3. Apply Salary Raise”);
    System.out.println(“4. View Total Salary Expense”);
    System.out.println(“5. Update Employee Salary”);
    System.out.println(“6. View Logs”);
    System.out.println(“7. Clear Logs”);
    System.out.println(“8. Exit”);
    System.out.print(“Choose an option: “);
    Int choice = scanner.nextInt();

    Switch (choice) {
        Case 1:
            System.out.print(“Enter Employee ID: “);
            Int id = scanner.nextInt();
            Scanner.nextLine();  // Consume newline
            System.out.print(“Enter Employee Name: “);
            String name = scanner.nextLine();
            System.out.print(“Enter Employee Salary: “);
            Double salary = scanner.nextDouble();
```

```
            Employees[employeeCount++] = new Employee(id, name, salary);
            Logger.log(“Created employee with ID: “ + id);
            Break;

        Case 2:
            System.out.println(“Total Employees: “ + Employee.getTotalEmployees());
            Break;

        Case 3:
            System.out.print(“Enter raise percentage: “);
            Double percentage = scanner.nextDouble();
            Employee.applyRaise(percentage);
            Logger.log(“Applied raise of “ + percentage + “% to all employees.”);
            Break;

        Case 4:
            System.out.println(“Total Salary Expense: $” +
Employee.calculateTotalSalaryExpense());
            Break;

        Case 5:
            System.out.print(“Enter Employee ID to update salary: “);
            Int empID = scanner.nextInt();
            System.out.print(“Enter new salary: “);
            Double newSalary = scanner.nextDouble();
```

```
        For (Employee emp : employees) {
            If (emp != null && emp.getEmployeeID() == empID) {
                Emp.updateSalary(newSalary);
                Logger.log(“Updated salary for employee ID: “ + empID);
                Break;
            }
        }
        Break;

    Case 6:
        System.out.println(“Logs:”);
        System.out.println(logger.getLog());
        Break;

    Case 7:
        Logger.clearLog();
        System.out.println(“Logs cleared.”);
        Break;

    Case 8:
        System.out.println(“Exiting…”);
        Return;

    Default:
        System.out.println(“Invalid option! Try again.”);
}
```

```
        }
    }
}
```

Assignment (2 august)

Q-1) Write a program that checks if a given year is a leap year or not using both if-else and switch-case.

```
Import java.util.Scanner;

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

        // Prompt the user to enter a year
System.out.print(“Enter a year: “);
Int year = scanner.nextInt();
```

```
        // Check if the year is a leap year using if-else
        If (isLeapYearUsingIfElse(year)) {
System.out.println(year + " is a leap year (using if-else).");
        } else {
System.out.println(year + " is not a leap year (using if-else).");
        }

        // Check if the year is a leap year using switch-case
checkLeapYearUsingSwitch(year);

scanner.close();
          }

    // Method to check leap year using if-else
    Public static booleanisLeapYearUsingIfElse(int year) {
        If (year % 4 == 0) {
            If (year % 100 == 0) {
                    Return year % 400 == 0; // Divisible by 400
            }
            Return true; // Divisible by 4 but not by 100
        }
        Return false; // Not divisible by 4
          }
```

```
    // Method to check leap year using switch-case
    Public static void checkLeapYearUsingSwitch(int year) {
IntisLeap = 0; // 0: not leap, 1: leap


        // Determine if the year is a leap year and set isLeap accordingly
        Switch (year % 4) {
            Case 0:
                    Switch (year % 100) {
                    Case 0:
                    Switch (year % 400) {
                         Case 0:
isLeap = 1; // Leap year
                              break;
                              default:
isLeap = 0; // Not a leap year
                    }
                    Break;
                    Default:
isLeap = 1; // Leap year
                    }
                    Break;
            Default:
```

```
isLeap = 0; // Not a leap year
        }

        // Print the result
        If (isLeap == 1) {
System.out.println(year + " is a leap year (using switch-case).");
        } else {
System.out.println(year + " is not a leap year (using switch-case).");
        }
          }
}
```

Q-2)Implement a program that calculates the Body Mass Index (BMI) based on height and weight input using if-else to classify the BMI int categories (underweight, normal weight, overweight,etc).

```
Import java.util.Scanner;

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

```
        // Prompt the user for weight and height
System.out.print(“Enter your weight in kilograms: “);
        Double weight = scanner.nextDouble();

System.out.print(“Enter your height in meters: “);
        Double height = scanner.nextDouble();

        // Calculate BMI
        Double bmi = calculateBMI(weight, height);

        // Display the BMI result
System.out.printf(“Your BMI is: %.2f\n”, bmi);

        // Classify the BMI
classifyBMI(bmi);

scanner.close();
          }

    // Method to calculate BMI
    Public static double calculateBMI(double weight, double height) {
        Return weight / (height * height);
          }
```

```
    // Method to classify the BMI
    Public static void classifyBMI(double bmi) {
        If (bmi< 18.5) {
System.out.println(“You are classified as: Underweight”);
        } else if (bmi>= 18.5 &&bmi< 24.9) {
System.out.println(“You are classified as: Normal weight”);
        } else if (bmi>= 25 &&bmi< 29.9) {
System.out.println(“You are classified as: Overweight”);
        } else {
System.out.println(“You are classified as: Obesity”);
        }
          }
}
```

Q-3) Write a program that checks if a person is eligible to vote based on their age.

```
Import java.util.Scanner;

Public class VotingEligibility {
    Public static void main(String[] args) {
```

```
        // Create a Scanner object to read input from the user
        Scanner scanner = new Scanner(System.in);

        // Prompt the user to enter their age
System.out.print(“Enter your age: “);
Int age = scanner.nextInt();

        // Check if the user is eligible to vote
        If (age >= 18) {
System.out.println(“You are eligible to vote.”);
        } else {
System.out.println(“You are not eligible to vote.”);
        }

        // Close the scanner to prevent resource leaks
Scanner.close();
          }
}
```

Q-4) Write a program that takes a month (1-12) and prints the corresponding season (Winter, Spring, Summer, Autumn) using a switch case

```
Import java.util.Scanner;


Public class SeasonChecker {
    Public static void main(String[] args) {
        // Create a Scanner object to read input from the user
        Scanner scanner = new Scanner(System.in);


        // Prompt the user to enter a month number (1-12)
System.out.print(“Enter a month (1-12): “);
Int month = scanner.nextInt();


        // Determine the season using a switch statement
        String season;


        Switch (month) {
            Case 1: // January
            Case 2: // February
            Case 12: // December
                    Season = “Winter”;
                    Break;
            Case 3: // March
```

```
            Case 4: // April
            Case 5: // May
                Season = “Spring”;
                Break;
            Case 6: // June
            Case 7: // July
            Case 8: // August
                Season = “Summer”;
                Break;
            Case 9: // September
            Case 10: // October
            Case 11: // November
                Season = “Autumn”;
                Break;
            Default:
                Season = “Invalid month. Please enter a number between 1 and 12.”;
                Break;
        }

        // Print the corresponding season
System.out.println(“The corresponding season is: “ + season);

        // Close the scanner to prevent resource leaks
```

```
Scanner.close();
        }
}
```

Q-5) Write a program that allows the user to select a shape (Circle, Square, Rectangle, Triangle) and then calculates the area based on user-provided dimensions using a switch case

```
Import java.util.Scanner;

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

System.out.println("Select a shape to calculate the area:");
System.out.println("1. Circle");
System.out.println("2. Square");
System.out.println("3. Rectangle");
System.out.println("4. Triangle");
System.out.print("Enter your choice (1-4): ");
Int choice = scanner.nextInt();

        Double area;

        Switch (choice) {
        Case 1: // Circle
System.out.print("Enter the radius of the circle: ");
            Double radius = scanner.nextDouble();
                Area = Math.PI * radius * radius; // Area of circle = πr²
System.out.println("The area of the circle is: " + area);
            Break;
```

```
        Case 2: // Square
System.out.print("Enter the side length of the square: ");
            Double side = scanner.nextDouble();
                Area = side * side; // Area of square = side²
System.out.println("The area of the square is: " + area);
            Break;

        Case 3: // Rectangle
System.out.print("Enter the length of the rectangle: ");
            Double length = scanner.nextDouble();
System.out.print("Enter the width of the rectangle: ");
            Double width = scanner.nextDouble();
                Area = length * width; // Area of rectangle = length × width
System.out.println("The area of the rectangle is: " + area);
            Break;

        Case 4: // Triangle
System.out.print("Enter the base of the triangle: ");
            Double base = scanner.nextDouble();
System.out.print("Enter the height of the triangle: ");
            Double height = scanner.nextDouble();
                Area = 0.5 * base * height; // Area of triangle = 0.5 × base × height
System.out.println("The area of the triangle is: " + area);
            Break;

        Default:
System.out.println("Invalid choice. Please select a number between 1 and 4.");
            Break;
        }

        // Close the scanner
Scanner.close();
        }
}
.
```

Assignment- 6

1. Declare a single-dimensional array of 5 integers, print default values, accept records, and print updated values.

Flowchart

1. **Start**
2. Declare an array of size 5.
3. Initialize all elements to default values (0).
4. **For** each element in the array:
    - Print the element (default value).
5. Accept 5 integers from the user.
6. **For** each input:
    - Update the corresponding array element.
7. **For** each element in the array:
    - Print the updated element.
8. **End**

Algorithm:

1. Declare a single-dimensional array of size 5.
2. Loop through the array to print default values (all 0).
3. Use a `Scanner` object to input 5 integers from the user.
4. Update the array with these values and print the updated values.

```
import java.util.Scanner;

public class ArrayDefaultValues {
    public static void main(String[] args) {
        int[] arr = new int[5]; // Declare an array of 5 integers

        // Print default values
        System.out.println("Default values of the array:");
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + " ");
```

```
        }
        System.out.println();

        // Accept input from the user
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter 5 integers to update the array:");
        for (int i = 0; i < arr.length; i++) {
            arr[i] = scanner.nextInt();
        }

        // Print updated values
        System.out.println("Updated values of the array:");
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + " ");
        }
    }
}
```

---

2. Declare a single-dimensional array and use methods `acceptRecord` and `printRecord`.

Flowchart

1. **Start**
2. Declare an array of size 5.
3. Call `acceptRecord` method.
    - Accept 5 integers from the user and store them in the array.
4. Call `printRecord` method.
    - Print the array values.
5. **End**

Algorithm:

1. Declare a single-dimensional array of size 5.
2. Create `acceptRecord` method to input values from the user.
3. Create `printRecord` method to display the values of the array.

```
import java.util.Scanner;

public class ArrayWithMethods {
    public static void main(String[] args) {
        int[] arr = new int[5];
        acceptRecord(arr);
        printRecord(arr);
    }

    public static void acceptRecord(int[] arr) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter 5 integers:");
        for (int i = 0; i < arr.length; i++) {
            arr[i] = scanner.nextInt();
        }
    }

    public static void printRecord(int[] arr) {
        System.out.println("Array values:");
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + " ");
        }
    }
}
```

---

3. Find the maximum and minimum values in a single-dimensional array.

Flowchart

1. **Start**
2. Declare an array of size 5.
3. Accept 5 integers from the user.
4. Initialize `max` and `min` with the first element of the array.
5. **For** each element from the second to the last:
    - If the element is greater than `max`, update `max`.
    - If the element is less than `min`, update `min`.
6. Print `max` and `min`.

7. **End**

Algorithm:

1. Declare an array and accept values.
2. Traverse the array to find the maximum and minimum elements.
3. Print the result.

```
import java.util.Scanner;

public class MaxMinArray {
    public static void main(String[] args) {
        int[] arr = new int[5];
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter 5 integers:");

        for (int i = 0; i < arr.length; i++) {
            arr[i] = scanner.nextInt();
        }

        int max = arr[0];
        int min = arr[0];

        for (int i = 1; i < arr.length; i++) {
            if (arr[i] > max) {
                max = arr[i];
            }
            if (arr[i] < min) {
                min = arr[i];
            }
        }

        System.out.println("Maximum value: " + max);
        System.out.println("Minimum value: " + min);
    }
}
```

4. Remove duplicate elements from a single-dimensional array.

Flowchart

1. **Start**
2. Declare an array of size 5.
3. Accept 5 integers from the user.
4. Create a second array to store unique elements.
5. **For** each element in the original array:
    - **If** the element is not in the second array:
        - Add it to the second array.
6. Print the second array (which contains no duplicates).
7. **End**

Algorithm:

1. Create an array and accept elements from the user.
2. Use another array to store non-duplicate elements.
3. Traverse the original array to check for duplicates and add only unique elements to the second array.
4. Print the array without duplicates.

```
import java.util.Scanner;
import java.util.Arrays;

public class RemoveDuplicates {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      int[] arr = new int[5];
      System.out.println("Enter 5 integers (duplicates allowed):");

      for (int i = 0; i < arr.length; i++) {
         arr[i] = scanner.nextInt();
      }
```

```
        int[] result = removeDuplicates(arr);
        System.out.println("Array after removing duplicates:");
        for (int i : result) {
            System.out.print(i + " ");
        }
    }

    public static int[] removeDuplicates(int[] arr) {
        return Arrays.stream(arr).distinct().toArray();
    }
}
```

---

5. Find the intersection of two arrays.

Flowchart

1. **Start**
2. Declare two arrays of size 5.
3. Accept 5 integers for the first array.
4. Accept 5 integers for the second array.
5. Initialize an empty set for storing the intersection.
6. **For** each element in the first array:
    - **If** the element exists in the second array:
        - Add it to the intersection set.
7. Print the intersection set.
8. **End**

---

6. Find the missing number in an array from 1 to N.

Flowchart

1. **Start**
2. Declare an array of size `N-1`.
3. Accept `N-1` integers from the user.
4. Calculate the sum of numbers from 1 to N using the formula `N(N+1)/2`.
5. Calculate the sum of the elements in the array.
6. Subtract the array sum from the sum of numbers from 1 to N.
7. The result is the missing number.

8. Print the missing number.
9. **End**

---

7. Declare a single-dimensional array as a field inside a class and define `acceptRecord` and `printRecord` methods.

Flowchart

1. **Start**
2. Create a class with an integer array as a field.
3. Initialize the array in the constructor.
4. Call `acceptRecord` method:
    - Accept user input for all array elements.
5. Call `printRecord` method:
    - Print the array values.
6. **End**

---

8. Modify the previous assignment to use getter and setter methods.

Flowchart

1. **Start**
2. Create a class with an integer array as a field.
3. Initialize the array in the constructor.
4. **For** each index in the array:
    - Call `setElement` to assign a value to the array element at that index.
5. **For** each index in the array:
    - Call `getElement` to retrieve and print the element at that index.
6. **End**

---

9. Airplane Seat Assignment System.

Flowchart

1. **Start**
2. Initialize a 2D array to represent seats (rows and columns).
3. **Display seating chart**:

    - Print all seats as available (0).
4. **Book a seat**:
    - **If** the seat is available (0), mark it as booked (X).
    - **Else**, display "Seat already booked."
5. **Cancel a booking**:
    - **If** the seat is booked (X), mark it as available (0).
    - **Else**, display "Seat is not booked."
6. **Check seat availability**:
    - **If** seat is available (0), display "Seat is available."
    - **Else**, display "Seat is booked."
7. **End**

Assignment-5

1)Create a base class BankAccount with methods like deposit() and withdraw(). Derive a class SavingsAccount that overrides the withdraw() method to impose a limit on the withdrawal amount. Write a program that demonstrates the use of overridden methods and proper access modifiers & return the details.

```
/ Base class BankAccount
class BankAccount {
    private String accountHolderName;
    private double balance;

    // Constructor
    public BankAccount(String accountHolderName, double initialBalance) {
        this.accountHolderName = accountHolderName;
        if (initialBalance >= 0) {
            this.balance = initialBalance;
        } else {
            this.balance = 0.0;
        }
    }
```

```
    // Method to deposit money
    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
            System.out.println("Deposited: " + amount);
        } else {
            System.out.println("Deposit amount must be positive.");
        }
    }

    // Method to withdraw money
    public void withdraw(double amount) {
        if (amount > 0 && amount <= balance) {
            balance -= amount;
            System.out.println("Withdrawn: " + amount);
        } else {
            System.out.println("Insufficient balance or invalid amount.");
        }
    }

    // Getter to return balance
    public double getBalance() {
        return balance;
    }

    // Method to return account details
    public void accountDetails() {
        System.out.println("Account Holder: " + accountHolderName);
        System.out.println("Balance: " + balance);
    }
}

// Derived class SavingsAccount
class SavingsAccount extends BankAccount {
    private static final double WITHDRAWAL_LIMIT = 1000.0;

    // Constructor
    public SavingsAccount(String accountHolderName, double initialBalance) {
        super(accountHolderName, initialBalance);
    }

    // Overridden withdraw method with a limit
    @Override
    public void withdraw(double amount) {
```

```
        if (amount > WITHDRAWAL_LIMIT) {
            System.out.println("Withdrawal amount exceeds the limit of " + WITHDRAWAL_LIMIT);
        } else {
            super.withdraw(amount); // Calling the base class method
        }
    }
}

// Main class to demonstrate the functionality
public class Main {
    public static void main(String[] args) {
        // Creating a SavingsAccount object
        SavingsAccount mySavings = new SavingsAccount("John Doe", 2000.0);

        // Display initial account details
        mySavings.accountDetails();

        // Deposit money
        mySavings.deposit(500.0);

        // Attempt to withdraw an amount within the limit
        mySavings.withdraw(800.0);

        // Attempt to withdraw an amount exceeding the limit
        mySavings.withdraw(1500.0);

        // Display final account details
        mySavings.accountDetails();
    }
}
```

2) Create a base class Vehicle with attributes like make and year. Provide a constructor in Vehicle to initialize these attributes. Derive a class Car that has an additional attribute model and write a constructor that initializes make, year, and model. Write a program to create a Car object and display its details.

```
// Base class Vehicle
class Vehicle {
    private String make;
    private int year;
```

```
    // Constructor to initialize make and year
    public Vehicle(String make, int year) {
        this.make = make;
        this.year = year;
    }

    // Getter for make
    public String getMake() {
        return make;
    }

    // Getter for year
    public int getYear() {
        return year;
    }

    // Method to display vehicle details
    public void displayDetails() {
        System.out.println("Make: " + make);
        System.out.println("Year: " + year);
    }
}

// Derived class Car
class Car extends Vehicle {
    private String model;

    // Constructor to initialize make, year, and model
    public Car(String make, int year, String model) {
        // Call the constructor of the base class (Vehicle)
        super(make, year);
        this.model = model;
    }

    // Getter for model
    public String getModel() {
        return model;
    }

    // Overriding displayDetails method to include model
    @Override
    public void displayDetails() {
        // Call the base class displayDetails method
        super.displayDetails();
```

```
        System.out.println("Model: " + model);
    }
}

// Main class to demonstrate the functionality
public class Main {
    public static void main(String[] args) {
        // Create a Car object
        Car myCar = new Car("Toyota", 2020, "Camry");

        // Display the details of the car
        myCar.displayDetails();
    }
}
```

3) Create a base class Animal with attributes like name, and methods like eat() and sleep(). Create a subclass Dog that inherits from Animal and has an additional method bark(). Write a program to demonstrate the use of inheritance by creating objects of Animal and Dog and calling their methods

```
// Base class Animal
class Animal {
    private String name;

    // Constructor to initialize the name
    public Animal(String name) {
        this.name = name;
    }

    // Method to simulate eating
    public void eat() {
        System.out.println(name + " is eating.");
    }

    // Method to simulate sleeping
    public void sleep() {
        System.out.println(name + " is sleeping.");
    }

    // Getter for name
    public String getName() {
        return name;
    }
```

```
}

// Subclass Dog that inherits from Animal
class Dog extends Animal {

    // Constructor to initialize the name using the base class constructor
    public Dog(String name) {
        super(name);
    }

    // Additional method to simulate barking
    public void bark() {
        System.out.println(getName() + " is barking.");
    }
}

// Main class to demonstrate the functionality
public class Main {
    public static void main(String[] args) {
        // Create an object of the base class Animal
        Animal genericAnimal = new Animal("Generic Animal");
        genericAnimal.eat();
        genericAnimal.sleep();

        System.out.println();  // Adding a line break for clarity

        // Create an object of the derived class Dog
        Dog myDog = new Dog("Buddy");
        myDog.eat();       // Inherited method from Animal class
        myDog.sleep();     // Inherited method from Animal class
        myDog.bark();      // Method specific to Dog class
    }
}
```

4) Build a class Student which contains details about the Student and compile and run its instance

```
// Class Student to store details about the student
class Student {
    private String name;
    private int age;
    private String studentId;
```

```
    private String course;

    // Constructor to initialize the attributes
    public Student(String name, int age, String studentId, String course) {
        this.name = name;
        this.age = age;
        this.studentId = studentId;
        this.course = course;
    }

    // Getter for name
    public String getName() {
        return name;
    }

    // Getter for age
    public int getAge() {
        return age;
    }

    // Getter for studentId
    public String getStudentId() {
        return studentId;
    }

    // Getter for course
    public String getCourse() {
        return course;
    }

    // Method to display student details
    public void displayDetails() {
        System.out.println("Student Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Student ID: " + studentId);
        System.out.println("Course: " + course);
    }
}

// Main class to run the Student class
public class Main {
    public static void main(String[] args) {
        // Create an instance of the Student class
        Student student1 = new Student("Alice", 20, "S123456", "Computer Science");
```

```
        // Display the details of the student
        student1.displayDetails();
    }
}
```

5. Write a Java program to create a base class Vehicle with methods startEngine() and stopEngine(). Create two subclasses Car and Motorcycle. Override the startEngine() and stopEngine() methods in each subclass to start and stop the engines differently

```
// Base class Vehicle
class Vehicle {
    // Method to start the engine
    public void startEngine() {
        System.out.println("Starting the vehicle engine.");
    }

    // Method to stop the engine
    public void stopEngine() {
        System.out.println("Stopping the vehicle engine.");
    }
}

// Subclass Car that extends Vehicle
class Car extends Vehicle {
    // Override startEngine for Car
    @Override
    public void startEngine() {
        System.out.println("Car engine is starting with a roar!");
    }

    // Override stopEngine for Car
    @Override
    public void stopEngine() {
        System.out.println("Car engine is stopping smoothly.");
    }
}

// Subclass Motorcycle that extends Vehicle
class Motorcycle extends Vehicle {
    // Override startEngine for Motorcycle
    @Override
    public void startEngine() {
```

```
        System.out.println("Motorcycle engine is starting with a vroom!");
    }

    // Override stopEngine for Motorcycle
    @Override
    public void stopEngine() {
        System.out.println("Motorcycle engine is stopping quickly.");
    }
}

// Main class to demonstrate the functionality
public class Main {
    public static void main(String[] args) {
        // Create an instance of Car
        Vehicle myCar = new Car();
        myCar.startEngine();  // Calls the overridden method in Car
        myCar.stopEngine();   // Calls the overridden method in Car

        System.out.println();  // Adding a line break for clarity

        // Create an instance of Motorcycle
        Vehicle myMotorcycle = new Motorcycle();
        myMotorcycle.startEngine();  // Calls the overridden method in Motorcycle
        myMotorcycle.stopEngine();   // Calls the overridden method in Motorcycle
    }
}
```

Assignment-3

JDK Components: JDK includes the Java compiler, Java Runtime Environment (JRE), libraries, and development tools like debugger and documentation tools.
JDK, JVM, JRE: JDK is the development kit, JVM runs Java code, and JRE provides libraries and environment to run the code.
Role of JVM: The JVM runs Java bytecode by converting it into machine code, ensuring platform independence.
JVM Memory Management: JVM divides memory into Heap, Stack, Method Area, and others, and uses garbage collection for memory management.
JIT Compiler: JIT compiles bytecode to native code at runtime for performance. Bytecode is platform-independent intermediate code.
JVM Architecture: It consists of class loader, memory area, execution engine, and native interface for executing Java bytecode.

Platform Independence: JVM abstracts platform differences by running Java bytecode on any machine with the respective JVM.
Class Loader: It loads classes into memory. Garbage collection reclaims memory from objects no longer in use.
Access Modifiers: Public (accessible everywhere), private (accessible within the class), protected (accessible in the package and subclasses), default (accessible within the package).
Public, Protected, Default: Public is accessible everywhere, protected in the package and subclasses, default only within the package.
Override with Different Modifier: No, you cannot reduce the visibility; a protected method cannot be overridden with private.
Protected vs. Default: Protected allows subclass access, while default restricts access to the same package.
Private Class: Only inner classes can be private, limiting access to their enclosing class.
Protected/Private Top-Level Class: No, top-level classes cannot be private or protected, they can only be public or package-private.
Private Members: Private members are inaccessible from other classes, even within the same package.
Package-Private (Default) Access: Members are accessible only within the same package, restricting visibility outside of it.

Assignment-4

Static keyword and memory management: Static variables and methods belong to the class, not instances, reducing memory use as only one copy exists.
Overloading and Overriding static methods: Static methods can be overloaded but not overridden. Static variables are shared across all instances of a class.
Final keyword significance: Final variables cannot be reassigned, final methods cannot be overridden, and final classes cannot be subclassed.
Narrowing and widening conversions: Widening converts smaller to larger types, while narrowing converts larger types to smaller ones.
Examples: Widening: `int` to `double`, narrowing: `double` to `int`.
Narrowing conversion precision loss: Java requires explicit casting for narrowing conversions to signal potential precision loss.
Automatic widening conversion: Java automatically converts smaller types to larger compatible types without explicit casting.
Implications of conversions: Widening is safe without data loss; narrowing risks data loss and requires explicit casting.

1) Write a program that demonstrates widening conversion from int to double and prints the result.

2) Create a program that demonstrates narrowing conversion from double to int and prints the result.

3) Write a program that performs arithmetic operations involving different data types (int, double, float) and observes how Java handles widening conversions automatically.

4) Write a Program that demonstrates widening conversion from int to (double,float, boolean, string) and prints the result.

1. **Widening conversion from int to double**:

```
public class WideningExample {
    public static void main(String[] args) {
```

```
        int intValue = 10;
        double doubleValue = intValue;  // Widening conversion
        System.out.println("Widening from int to double: " +
doubleValue);
    }
}
```

2. **Narrowing conversion from double to int**:

```
public class NarrowingExample {
    public static void main(String[] args) {
        double doubleValue = 10.5;
        int intValue = (int) doubleValue;  // Narrowing conversion
(explicit cast)
        System.out.println("Narrowing from double to int: " +
intValue);
    }
}
```

3. **Arithmetic operations with different data types**:

```
public class MixedDataTypes {
    public static void main(String[] args) {
        int intValue = 5;
        double doubleValue = 3.14;
        float floatValue = 2.5f;

        // Automatic widening conversions during arithmetic
        double result = intValue + doubleValue + floatValue;

        System.out.println("Result of int + double + float: " +
result);
    }
}
```

4. **Widening conversion from int to double, float, boolean, and string**:

```
public class WideningMultiple {
    public static void main(String[] args) {
        int intValue = 42;

        // Widening to double
        double doubleValue = intValue;
        System.out.println("Widening from int to double: " +
doubleValue);

        // Widening to float
        float floatValue = intValue;
        System.out.println("Widening from int to float: " +
floatValue);

        // Widening to String
        String stringValue = Integer.toString(intValue);
        System.out.println("Widening from int to String: " +
stringValue);
    }
}
```