Chapter 19. Generics

Objectives:

To define and use generic methods and bounded generic types (§19.4).
To develop a generic sort method to sort an array of Comparable objects (§19.5).
To design and implement generic matrix classes (§19.9).

Problem A

Define a GenericMethod class to calculate the sum of all elements of an ArrayList of different numeric data types.

1) Create an ArrayList of the Number type.

2) Pre-fill your array by inserting values to your arrayList (ex: int, double, long, float) in main()

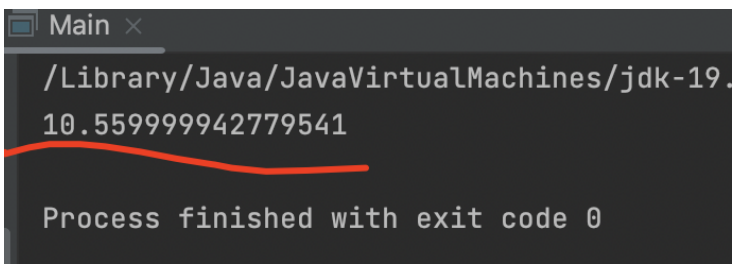
4) Add a method `sum(ArrayList<Number> array)` that calculates the sum.

3) Execute main() to see the sum of all elements in your array.

Ex: `Input:`

`ArrayList {1, 2L, 3.00, 4.56F}`

`Output:`



Problem B

Sort the areas of geometric objects.

Write a method GenericSort that sorts the areas of all the geometric objects in an array.

Write a test program that creates an array of four objects (two circles and two rectangles) and outputs a sorted array using the GenericSort method.

Ex. Input:

```
GeometricObject[] arr = {new Circle( radius: 5), new Circle( radius: 8),
        new Rectangle( width: 3,  height: 4), new Rectangle( width: 4,  height: 2)};
```

Output:

```
Sorted Geometric objects:
Rectangle: 8
Rectangle: 12
Circle: 78
Circle: 201

Process finished with exit code 0
```

Problem C

Modify LISTING 19.10 and LISTING 19.11 GenericMatrix.java and IntegerMartix.java

Add a method named minMatrix that finds the smallest value among two matrices.

Create a TestMatrix class to test your method with the right data.

Output operations + and * along with the minimum value.

IntegerMartix.java

```
public class IntegerMatrix extends GenericMatrix<Integer> {
  @Override /** Add two integers */
  protected Integer add(Integer o1, Integer o2) {
    return o1 + o2;
  }

  @Override /** Multiply two integers */
  protected Integer multiply(Integer o1, Integer o2) {
    return o1 * o2;
  }

  @Override /** Specify zero for an integer */
  protected Integer zero() {
    return 0;
  }
}
```

GenericMatrix.java

```
public abstract class GenericMatrix<E extends Number> {
  /** Abstract method for adding two elements of the matrices */
  protected abstract E add(E o1, E o2);
  /** Abstract method for multiplying two elements of the matrices */
  protected abstract E multiply(E o1, E o2);
  /** Abstract method for defining zero for the matrix element */
  protected abstract E zero();
  /** Add two matrices */
  public E[][] addMatrix(E[][] matrix1, E[][] matrix2) {
    // Check bounds of the two matrices
    if ((matrix1.length != matrix2.length) ||
        (matrix1[0].length != matrix2[0].length)) {
      throw new RuntimeException(
        "The matrices do not have the same size");
    }
    E[][] result =
      (E[][])new Number[matrix1.length][matrix1[0].length];
    // Perform addition
    for (int i = 0; i < result.length; i++)
      for (int j = 0; j < result[i].length; j++) {
        result[i][j] = add(matrix1[i][j], matrix2[i][j]);
      }
```

```
    return result;
  }
  /** Multiply two matrices */
  public E[][] multiplyMatrix(E[][] matrix1, E[][] matrix2) {
    // Check bounds
    if (matrix1[0].length != matrix2.length) {
      throw new RuntimeException(
        "The matrices do not have compatible size");
    }
    // Create result matrix
    E[][] result =
      (E[][])new Number[matrix1.length][matrix2[0].length];
    // Perform multiplication of two matrices
    for (int i = 0; i < result.length; i++) {
      for (int j = 0; j < result[0].length; j++) {
        result[i][j] = zero();

        for (int k = 0; k < matrix1[0].length; k++) {
          result[i][j] = add(result[i][j],
            multiply(matrix1[i][k], matrix2[k][j]));
        }
      }
    }
    return result;
  }
  /** Print matrices, the operator, and their operation result */
  public static void printResult(
      Number[][] m1, Number[][] m2, Number[][] m3, char op) {
    for (int i = 0; i < m1.length; i++) {
      for (int j = 0; j < m1[0].length; j++)
        System.out.print(" " + m1[i][j]);
      if (i == m1.length / 2)
        System.out.print("  " + op + "  ");
      else
        System.out.print("     ");

      for (int j = 0; j < m2.length; j++)
        System.out.print(" " + m2[i][j]);

      if (i == m1.length / 2)
        System.out.print("  =  ");
      else
        System.out.print("     ");
      for (int j = 0; j < m3.length; j++)
        System.out.print(m3[i][j] + " ");
      System.out.println();
    }
  }
}
```