ICS1312 - Java Programming
Lab Exercise 6: Generic Types

Simiyon Vincent Samuel L.

Reg no: 312224001062

August 7, 2025

SSN College of Engineering
Department of Computer Science and Engineering
II Year M.Tech Integrated CSE
Academic Year 2025-26

1 Aim

2 Learning Objectives

- Understand and implement generic types in Java for flexible and reusable code.
- Develop a generic stack class with operations like push, pop, and peek, using ArrayList.
- Create a generic class to manage shapes with type constraints and calculate total area.
- Practice throwing and catching exceptions for error scenarios in generic classes.
- Demonstrate type safety and modular programming in Java applications using generics.

3 Program Details

3.1 Program 1: Generic Stack

3.1.1 Objective

Design and implement a GenericStack<T> class using an ArrayList to store elements of any reference type, supporting operations: push, pop, peek, isEmpty, and size. Handle empty stack operations with a custom EmptyStackException.

3.1.2 Test Cases

1. Integer Stack Test: Create a GenericStack<Integer>, push values (10, 20), check size, peek, and pop elements.
2. String Stack Test: Create a GenericStack<String>, push values ("Apple", "Banana"), check size, peek, and pop elements.
3. Empty Stack Test: Attempt to pop and peek from an empty stack, verify that an EmptyStackException is thrown.

3.1.3 Java Code

```
import java.util.ArrayList;

public class GenericStack {
    // GenericStack class
    static class GenericStack<T> {
        private ArrayList<T> stack;
        private int top;

        public GenericStack() {
            stack = new ArrayList<>();
            top = -1;
        }
```

```
    public void push(T element) {
        stack.add(element);
        top++;
    }

    public T pop() {
        if (isEmpty()) {
            throw new IllegalStateException("Stack is empty");
        }
        T element = stack.remove(top);
        top--;
        return element;
    }

    public T peek() {
        if (isEmpty()) {
            throw new IllegalStateException("Stack is empty");
        }
        return stack.get(top);
    }

    public boolean isEmpty() {
        return top == -1;
    }

    public int size() {
        return top + 1;
    }
}

// Student class for testing
static class Student {
    private int id;
    private String name;

    public Student(int id, String name) {
        this.id = id;
        this.name = name;
    }

    @Override
    public String toString() {
        return "Student[ID=" + id + ", Name=" + name + "]";
    }
}

// Main method to test GenericStack
public static void main(String[] args) {
```

```
        // Integer Stack Test
        System.out.println("Integer Stack Operations:");
        GenericStack<Integer> intStack = new GenericStack<>();
        intStack.push(10);
        intStack.push(20);
        intStack.push(30);
        System.out.println("Pushing: 10, 20, 30");
        System.out.println("Size: " + intStack.size());
        System.out.println("Is Empty: " + intStack.isEmpty());
        System.out.println("Top element: " + intStack.peek());
        System.out.println("Popped: " + intStack.pop());
        System.out.println("Size after pop: " + intStack.size());

        // String Stack Test
        System.out.println("\nString Stack Operations:");
        GenericStack<String> strStack = new GenericStack<>();
        strStack.push("Hello");
        strStack.push("World");
        System.out.println("Pushing: Hello, World");
        System.out.println("Size: " + strStack.size());
        System.out.println("Top element: " + strStack.peek());
        System.out.println("Popped: " + strStack.pop());
        System.out.println("Size after pop: " + strStack.size());

        // Student Stack Test
        System.out.println("\nStudent Stack Operations:");
        GenericStack<Student> studentStack = new GenericStack<>();
        studentStack.push(new Student(1, "John"));
        studentStack.push(new Student(2, "Jane"));
        System.out.println("Pushing: Student[ID=1, Name=John], Student[ID=2, Name=Jane]");
        System.out.println("Size: " + studentStack.size());
        System.out.println("Top element: " + studentStack.peek());
        System.out.println("Popped: " + studentStack.pop());
        System.out.println("Size after pop: " + studentStack.size());

        // Empty Stack Test
        System.out.println("\nEmpty Stack Test:");
        GenericStack<Integer> emptyStack = new GenericStack<>();
        try {
            emptyStack.pop();
        } catch (IllegalStateException e) {
            System.out.println("Exception: " + e.getMessage());
        }
    }
}
```

3.1.4 Expected Output

Testing Integer Stack:
Pushed: 10
Pushed: 20
Size: 2
Peek: 20
Pop: 20
Pop: 10
Testing String Stack:
Pushed: Apple
Pushed: Banana
Size: 2
Peek: Banana
Pop: Banana
Pop: Apple

Testing Empty Stack:
Exception: Stack is empty, cannot pop.
Exception: Stack is empty, cannot peek.

3.2 Program 2: ShapeBox

3.2.1 Objective

Write a Java program that defines a generic ShapeBox<T extends Shape> class to hold shapes (e.g., Circle, Rectangle) and calculate the total area of all shapes, with an abstract Shape class and concrete subclasses.

3.2.2 Test Cases

1. Circle Box Test: Create a ShapeBox<Circle>, add circles (radii 5.0, 3.0), display shapes, and calculate total area.
2. Rectangle Box Test: Create a ShapeBox<Rectangle>, add rectangles (4x6, 2x3), display shapes, and calculate total area.
3. Mixed Shape Box Test: Create a ShapeBox<Shape>, add a circle and a rectangle, display shapes, and calculate total area.

3.2.3 Java Code

```
import java.util.ArrayList;

public class ShapeBox {
    // Abstract Shape class
    abstract static class Shape {
        abstract double getArea();
```

```
    }

    // Circle class
    static class Circle extends Shape {
        private double radius;

        public Circle(double radius) {
            this.radius = radius;
        }

        @Override
        double getArea() {
            return Math.PI * radius * radius;
        }
    }

    // Rectangle class
    static class Rectangle extends Shape {
        private double length;
        private double width;

        public Rectangle(double length, double width) {
            this.length = length;
            this.width = width;
        }

        @Override
        double getArea() {
            return length * width;
        }
    }

    // ShapeBox class
    static class ShapeBox<T extends Shape> {
        private ArrayList<T> shapes;

        public ShapeBox() {
            shapes = new ArrayList<>();
        }

        public void addShape(T shape) {
            shapes.add(shape);
        }

        public double calculateTotalArea() {
            double total = 0.0;
            for (T shape : shapes) {
                total += shape.getArea();
            }
```

```
            return total;
        }
    }

    // Main method to test ShapeBox
    public static void main(String[] args) {
        // Valid Shapes Test
        System.out.println("ShapeBox Operations:");
        ShapeBox<Shape> shapeBox = new ShapeBox<>();
        shapeBox.addShape(new Circle(5.0));
        System.out.println("Adding Circle with radius 5.0");
        shapeBox.addShape(new Rectangle(4.0, 6.0));
        System.out.println("Adding Rectangle with length 4.0, width 6.0");
        shapeBox.addShape(new Rectangle(16.0, 22.0));
        System.out.println("Adding Rectangle with length 16.0, width 22.0");
        double totalArea = shapeBox.calculateTotalArea();
        System.out.printf("Total area of shapes in the box: %.2f\n", totalArea);

        // Empty ShapeBox Test
        System.out.println("\nEmpty ShapeBox Test:");
        ShapeBox<Shape> emptyBox = new ShapeBox<>();
        System.out.println("Total area: " + emptyBox.calculateTotalArea());
    }
}
```

3.2.4 Expected Output

Testing ShapeBox with Circles:
Added: Circle{radius=5.0}
Added: Circle{radius=3.0}
Shapes in box: [Circle{radius=5.0}, Circle{radius=3.0}]
Total Area: 103.67

Testing ShapeBox with Rectangles:
Added: Rectangle{length=4.0, width=6.0}
Added: Rectangle{length=2.0, width=3.0}
Shapes in box: [Rectangle{length=4.0, width=6.0}, Rectangle{length=2.0, width=3.0}] Total Area: 30.00

Testing ShapeBox with Mixed Shapes:
Added: Circle{radius=2.0}
Added: Rectangle{length=5.0, width=5.0}
Shapes in box: [Circle{radius=2.0}, Rectangle{length=5.0, width=5.0}]
Total Area: 37.57

4 Learning Outcomes

- Mastered the implementation of generic types in Java for stack and shape management.
- Gained proficiency in designing generic classes with type constraints and custom exceptions.
- Developed skills in type-safe programming, input validation, and error handling.
- Learned to create robust Java applications that handle various data types and error scenarios gracefully using generics.