Name: Muhammad Talha **Section: E** **Roll Number: 2023F-BSE-318**

SOFTWARE CONSTRUCTION AND DEVELOPMENT

LAB # 14

OPEN ENDED LAB

LAB TASK:

Generics:

Implement a generic class for representing a Loan in the Loan Management System. Utilize generics to allow flexibility in the type of loan amount (e.g., Integer, Double).

Mutability and Immutability:

- Create two versions of the Loan class:
- Mutable Loan: Allow modification of loan details after initialization.
- Immutable Loan: Design an immutable version of the Loan class where loan details cannot be modified once set.

Abstract Data Type (ADT):
Implement an abstract data type for managing a collection of loans. This ADT should provide methods for adding loans, retrieving loans, and calculating the total loan amount.

Exception Handling:
Implement exception classes for handling specific scenarios in the Loan Management System. For example, create a custom exception for cases where an attempt is made to modify an immutable loan.

Unit Testing:
Write comprehensive unit tests for the Loan Management System. Cover different scenarios, such as creating loans, adding them to the collection, attempting to modify immutable loans, and handling exceptions. Utilize a testing framework (e.g., JUnit) for efficient and organized testing.

1. **IMMUTABLE LOAN.JAVA:**

Name: Muhammad Talha **Section: E** **Roll Number: 2023F-BSE-318**

```java
package lms;
public final class ImmutableLoan<T extends Number> implements Loan<T> {

    private final String loanId;
    private final T amount;
    private final double interestRate;
    private final int durationMonths;
    private final String loanType;
    public ImmutableLoan(String loanId,
                         T amount,
                         double interestRate,
                         int durationMonths,
                         String loanType) {
        if (loanId == null || loanId.trim().isEmpty()) {
            throw new IllegalArgumentException("Loan ID cannot be null or empty");
        }
        if (amount == null || amount.doubleValue() <= 0) {
            throw new IllegalArgumentException("Amount must be positive");
        }
        if (interestRate < 0) {
            throw new IllegalArgumentException("Interest rate cannot be
negative");
        }
        if (durationMonths <= 0) {
            throw new IllegalArgumentException("Duration must be positive");
        }
        if (loanType == null || loanType.trim().isEmpty()) {
            throw new IllegalArgumentException("Loan type cannot be null or
empty");
        }
        this.loanId = loanId.trim();
        this.amount = amount;
        this.interestRate = interestRate;
        this.durationMonths = durationMonths;
        this.loanType = loanType.trim();
    }
    @Override
    public String getLoanId() {
        return loanId;
    }
    @Override
    public T getAmount() {
        return amount;
    }
    @Override
    public double getInterestRate() {
        return interestRate;
    }
    @Override
    public int getDurationMonths() {
        return durationMonths;
    }
    @Override
    public String getLoanType() {
        return loanType;
```

Name: Muhammad Talha **Section: E** **Roll Number: 2023F-BSE-318**

```
    }
    @Override
    public boolean isMutable() {
        return false;
    }
    @Override
    public double calculateTotalPayment() {
        double principal = amount.doubleValue();
        double monthlyRate = interestRate / 100.0 / 12;
        int numberOfPayments = durationMonths;
        if (monthlyRate == 0) {
            return principal;
        }
        double emi = principal * monthlyRate * Math.pow(1 + monthlyRate,
numberOfPayments)
                / (Math.pow(1 + monthlyRate, numberOfPayments) - 1);

        return emi * numberOfPayments;
    }
    @Override
    public String toString() {
        return String.format("ImmutableLoan[id=%s, type=%s, amount=%s,
rate=%.2f%%, months=%d]",
                loanId, loanType, amount, interestRate, durationMonths);
    }
}
```

2. IMMUTABLELOANMODIFICATIONEXCEPTION.JAVA:

```
package lms;
public class ImmutableLoanModificationException extends RuntimeException {
    public ImmutableLoanModificationException(String message) {
        super(message);
    }
    public ImmutableLoanModificationException(String message, Throwable cause) {
        super(message, cause);
    }
}
package lms;
public interface Loan<T extends Number> {
    String getLoanId();
    T getAmount();
    double getInterestRate();
    int getDurationMonths();
    String getLoanType();
    default boolean isMutable() {
        return false;
    }
    double calculateTotalPayment();
}
```

3. LOAN.JAVA(INTERFACE):

```
package lms;
public interface Loan<T extends Number> {
    String getLoanId();
    T getAmount();
    double getInterestRate();
    int getDurationMonths();
    String getLoanType();
    default boolean isMutable() {
        return false;
    }
    double calculateTotalPayment();
}
```

4. LOANREPSITORY.JAVA:

Name: Muhammad Talha **Section: E** **Roll Number: 2023F-BSE-318**

```
package lms;
import java.util.*;
public class LoanRepository<T extends Number> {
    private final Map<String, Loan<T>> loans = new HashMap<>();
    public void addLoan(Loan<T> loan) {
        if (loan == null) {
            throw new IllegalArgumentException("Loan cannot be null");
        }
        if (loans.containsKey(loan.getLoanId())) {
            throw new IllegalArgumentException("Loan with ID " + loan.getLoanId() + "
already exists");
        }
        loans.put(loan.getLoanId(), loan);
    }
    public Loan<T> getLoan(String loanId) {
        return loans.get(loanId);
    }
    public List<Loan<T>> getAllLoans() {
        return new ArrayList<>(loans.values());
    }
    public double getTotalLoanAmount() {
        return loans.values().stream()
                .mapToDouble(loan -> loan.getAmount().doubleValue())
                .sum();
    }
    public void updateLoanAmount(String loanId, T newAmount) {
        Loan<T> loan = getLoan(loanId);
        if (loan == null) {
            throw new IllegalArgumentException("Loan with ID " + loanId + " not
found");
        }
        if (!loan.isMutable()) {
            throw new ImmutableLoanModificationException(
                    "Cannot modify amount of immutable loan: " + loanId);
        }
        ((MutableLoan<T>) loan).setAmount(newAmount);
    }
    public int getLoanCount() {
        return loans.size();
    }
}
```

5. **LOANSYSTEMDEMO.JAVA:**

```
package lms;
public class LoanSystemDemo {
    public static void main(String[] args) {
        System.out.println("=== Loan Management System Demo ===\n");
        System.out.println("Using separate repositories for different amount
types:\n");
        LoanRepository<Double> doubleRepo = new LoanRepository<>();
        MutableLoan<Double> homeLoan = new MutableLoan<>(
                "H001", 3500000.0, 9.5, 240, "Home Loan");
        MutableLoan<Double> carLoan = new MutableLoan<>(
                "C001", 1800000.0, 11.75, 60, "Car Loan");
        doubleRepo.addLoan(homeLoan);
        doubleRepo.addLoan(carLoan);
        LoanRepository<Integer> intRepo = new LoanRepository<>();
        ImmutableLoan<Integer> personalLoan = new ImmutableLoan<>(
                "P001", 500000, 14.0, 36, "Personal Loan");
        intRepo.addLoan(personalLoan);
        System.out.println("Double-based loans:");
        printAllLoans(doubleRepo);
        System.out.printf("Total (Double): %,.2f\n\n",
  doubleRepo.getTotalLoanAmount());
        System.out.println("Integer-based loans:");
        printAllLoans(intRepo);
        System.out.printf("Total (Integer): %,.0f\n\n", intRepo.getTotalLoanAmount());
        System.out.println("=== Mutability Demonstration ===\n");
        System.out.println("Trying to increase Home loan amount (mutable)...");
        try {
            doubleRepo.updateLoanAmount("H001", 3800000.0);
            System.out.println("→ Success! Home loan updated.");
        } catch (Exception e) {
            System.out.println("→ Failed: " + e.getMessage());
        }
        System.out.println("\nTrying to change Personal loan amount (immutable)...");
        try {
            intRepo.updateLoanAmount("P001", 600000);
            System.out.println("→ This should NOT happen!");
        } catch (ImmutableLoanModificationException e) {
            System.out.println("→ Good! Protected: " + e.getMessage());
        }
```

Name: Muhammad Talha **Section: E** **Roll Number: 2023F-BSE-318**

```
        System.out.println("\nAfter modifications:");
        System.out.println("Double-based loans:");
        printAllLoans(doubleRepo);
        System.out.println("\nEstimated total payments (approximate):");
        System.out.printf("Home loan total:  %,.2f\n",
homeLoan.calculateTotalPayment());
        System.out.printf("Car loan total:   %,.2f\n",
carLoan.calculateTotalPayment());
        System.out.printf("Personal loan total: %,.0f\n",
personalLoan.calculateTotalPayment());
    }

    private static void printAllLoans(LoanRepository<?> repo) {
        for (Loan<?> loan : repo.getAllLoans()) {
            String mutability = loan.isMutable() ? "Mutable" : "Immutable";
            System.out.printf("  %-9s %-6s %-12s %,12.0f  %6.2f%%  %3d months
(type: %s)\n",
                    mutability,
                    loan.getLoanId(),
                    loan.getLoanType(),
                    loan.getAmount().doubleValue(),
                    loan.getInterestRate(),
                    loan.getDurationMonths(),
                    loan.getAmount().getClass().getSimpleName());

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

6. **MUTABLELOAN.JAVA:**

```
package lms;
public class MutableLoan<T extends Number> implements Loan<T> {

    private String loanId;
    private T amount;
    private double interestRate;      // annual interest rate in percent
    private int durationMonths;
    private String loanType;          // e.g. "Home", "Personal", "Car", "Education"
    public MutableLoan(String loanId,
                       T amount,
                       double interestRate,
                       int durationMonths,
                       String loanType) {

        if (loanId == null || loanId.trim().isEmpty()) {
            throw new IllegalArgumentException("Loan ID cannot be null or empty");
        }
        if (amount == null || amount.doubleValue() <= 0) {
            throw new IllegalArgumentException("Amount must be positive");
        }
        if (interestRate < 0) {
            throw new IllegalArgumentException("Interest rate cannot be negative");
        }
        if (durationMonths <= 0) {
            throw new IllegalArgumentException("Duration must be positive");
        }
        if (loanType == null || loanType.trim().isEmpty()) {
            throw new IllegalArgumentException("Loan type cannot be null or empty");
        }
        this.loanId = loanId.trim();
        this.amount = amount;
        this.interestRate = interestRate;
        this.durationMonths = durationMonths;
        this.loanType = loanType.trim();
    }

    @Override
    public String getLoanId() {
        return loanId;
    }

    @Override
    public T getAmount() {
        return amount;
    }
    @Override
    public double getInterestRate() {
        return interestRate;
    }
```

```
    @Override
    public int getDurationMonths() {
        return durationMonths;
    }
    @Override
    public String getLoanType() {
        return loanType;
    }
    @Override
    public boolean isMutable() {
        return true;
    }
    public void setLoanId(String loanId) {
        if (loanId == null || loanId.trim().isEmpty()) {
            throw new IllegalArgumentException("Loan ID cannot be null or empty");
        }
        this.loanId = loanId.trim();
    }

    public void setAmount(T amount) {
        if (amount == null || amount.doubleValue() <= 0) {
            throw new IllegalArgumentException("Amount must be positive");
        }
        this.amount = amount;
    }
    public void setInterestRate(double rate) {
        if (rate < 0) {
            throw new IllegalArgumentException("Interest rate cannot be negative");
        }
        this.interestRate = rate;
    }
    public void setDurationMonths(int months) {
        if (months <= 0) {
            throw new IllegalArgumentException("Duration must be positive");
        }
        this.durationMonths = months;
    }
    public void setLoanType(String loanType) {
        if (loanType == null || loanType.trim().isEmpty()) {
            throw new IllegalArgumentException("Loan type cannot be null or empty");
        }
        this.loanType = loanType.trim();
    }
    @Override
    public double calculateTotalPayment() {
        double principal = amount.doubleValue();
        double monthlyRate = interestRate / 100.0 / 12;
        int numberOfPayments = durationMonths;

        if (monthlyRate == 0) {
            return principal;
        }
        double emi = principal * monthlyRate * Math.pow(1 + monthlyRate,
umberOfPayments)
                / (Math.pow(1 + monthlyRate, numberOfPayments) - 1);


            return emi * numberOfPayments;
        }
        @Override
        public String toString() {
            return String.format("MutableLoan[id=%s, type=%s, amount=%s, rate=%.2f%%,
 months=%d]",
                    loanId, loanType, amount, interestRate, durationMonths);
        }
    }
```

7. **LOANMANAGEMENTSYSTEMTEST.JAVA(JUNIT)**

Name: Muhammad Talha **Section: E** **Roll Number: 2023F-BSE-318**

```java
package lms;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
class LoanManagementSystemTest {
    @Test
    void testMutableLoanModification() {
        MutableLoan<Double> loan = new MutableLoan<>("L001", 100000.0, 7.5, 60,
"Home");

        assertEquals(100000.0, loan.getAmount());
        assertTrue(loan.isMutable());
        loan.setAmount(150000.0);
        loan.setInterestRate(8.0);
        assertEquals(150000.0, loan.getAmount());
        assertEquals(8.0, loan.getInterestRate());
    }
    @Test
    void testImmutableLoanImmutability() {
        ImmutableLoan<Integer> loan = new ImmutableLoan<>("L002", 200000, 6.0, 36,
"Personal");
        assertEquals(200000, loan.getAmount());
        assertFalse(loan.isMutable());
        assertEquals(200000, loan.getAmount());
    }
    @Test
    void testLoanTotalPaymentCalculation() {
        Loan<Double> zeroInterest = new MutableLoan<>("L003", 1000.0, 0.0, 12,
"Education");
        assertEquals(1000.0, zeroInterest.calculateTotalPayment(), 0.001);

        Loan<Double> withInterest = new ImmutableLoan<>("L004", 1000.0, 12.0, 12,
"Personal");
        double expected = 1000 * (0.01) * Math.pow(1.01, 12) / (Math.pow(1.01, 12) -
1) * 12;
        assertEquals(expected, withInterest.calculateTotalPayment(), 0.01);
    }
    @Test
    void testLoanRepositoryAddAndRetrieve() {
        LoanRepository<Double> repo = new LoanRepository<>();

        MutableLoan<Double> loan1 = new MutableLoan<>("L005", 50000.0, 5.0, 24,
"Car");

        ImmutableLoan<Double> loan2 = new ImmutableLoan<>("L006", 75000.0, 6.5, 36,
"Personal");
        repo.addLoan(loan1);
        repo.addLoan(loan2);
        assertEquals(2, repo.getLoanCount());
        assertEquals(loan1, repo.getLoan("L005"));
        assertEquals(loan2, repo.getLoan("L006"));
    }
    @Test
    void testTotalLoanAmount() {
        LoanRepository<Double> repo = new LoanRepository<>();
        repo.addLoan(new MutableLoan<>("L007", 100000.0, 7.0, 60, "Home"));
        repo.addLoan(new ImmutableLoan<>("L008", 200000.0, 8.0, 36, "Personal"));
        repo.addLoan(new MutableLoan<>("L009", 150000.0, 6.0, 48, "Car"));
        assertEquals(450000.0, repo.getTotalLoanAmount(), 0.001);
    }
    @Test
    void testUpdateMutableLoanAmount() {
        LoanRepository<Double> repo = new LoanRepository<>();
        MutableLoan<Double> loan = new MutableLoan<>("L010", 100000.0, 7.0, 60,
"Home");
        repo.addLoan(loan);
        repo.updateLoanAmount("L010", 120000.0);
        assertEquals(120000.0, repo.getLoan("L010").getAmount());
    }
```

Name: Muhammad Talha **Section: E** **Roll Number: 2023F-BSE-318**

```
    @Test
    void testUpdateImmutableLoanThrowsException() {
        LoanRepository<Double> repo = new LoanRepository<>();
        ImmutableLoan<Double> immutable = new ImmutableLoan<>("L011", 150000.0, 6.5,
36, "Personal");
        repo.addLoan(immutable);
        assertThrows(ImmutableLoanModificationException.class, () -> {
            repo.updateLoanAmount("L011", 200000.0);
        });
    }
    @Test
    void testAddNullLoanThrowsException() {
        LoanRepository<Integer> repo = new LoanRepository<>();
        assertThrows(IllegalArgumentException.class, () -> repo.addLoan(null));
    }
    @Test
    void testGenericsSupportDifferentTypes() {
        LoanRepository<Integer> intRepo = new LoanRepository<>();
        intRepo.addLoan(new ImmutableLoan<>("L012", 500000, 7.0, 60, "Home"));
        LoanRepository<Double> doubleRepo = new LoanRepository<>();
        doubleRepo.addLoan(new MutableLoan<>("L013", 750000.5, 8.5, 48, "Car"));

        assertEquals(500000.0, intRepo.getTotalLoanAmount(), 0.001);
        assertEquals(750000.5, doubleRepo.getTotalLoanAmount(), 0.001);
    }
}
```

OUTPUT: