CO3408 – Week 2 Lab: Formal Specification & Contracts

Lab Objectives

By the end of this lab, students should be able to:

1. Write preconditions and postconditions for simple methods.
2. Define class invariants to ensure object consistency.
3. Apply Design by Contract principles to small classes.
4. Gather system requirements and design a simple state transition diagram.

Task 1: BankAccount – Deposit and Withdraw Methods

Scenario: Implement a BankAccount class with deposit and withdraw methods.

Requirements:

- deposit(amount) increases balance by amount.
- withdraw(amount) decreases balance by amount if sufficient funds exist.

Instructions:

1. Write preconditions and postconditions for both methods.
2. Use Design by Contract style (JML-style comments or pseudo-code).

Example:

```
/*@ requires amount > 0;
  @ ensures balance == \old(balance) + amount;
  @*/
public void deposit(int amount) { ... }
```

Task 2: Class Invariants

Scenario: Use the BankAccount from Task 1.

Instructions:

1. Define a class invariant to ensure the balance is never negative.
2. Explain why maintaining this invariant is important.

Hint:

//@ invariant balance >= 0;

Task 3: MinMax Class Specification

Scenario: Implement a MinMax class to find the minimum and maximum of an array.

Instructions:

1. Specify preconditions: array must not be empty.
2. Specify postconditions:
    - min returns the smallest element.
    - max returns the largest element.
3. Optionally, define a class invariant if applicable.

Scenario:

You are modelling a simple Bank ATM system that allows customers to perform standard banking operations. The system's behaviour depends on its current state and user actions (events).

Instructions

Step 1 – Gather System Requirements

List the main operations the ATM can perform:

- Insert Card
- Enter PIN
- Withdraw
- Deposit
- Eject Card

(Refer to the system requirements document for operation descriptions.)

Step 2 – Identify Main System States

Determine the key system states involved in ATM operation:

- Idle – waiting for card
- Card Inserted – card detected, awaiting PIN
- PIN Entered – authenticated user session active
- Transaction in Progress – performing a withdrawal or deposit
- Out of Service – ATM unavailable due to fault or maintenance

Step 3 – Draw the State Transition Diagram

Create a UML state diagram that shows:

- States as rounded rectangles (nodes)
- Transitions as labelled arrows representing events or operations

Your diagram should include (at minimum):

- InsertCard → moves Idle → Card Inserted
- EnterPIN → moves Card Inserted → PIN Entered
- Withdraw or Deposit → moves PIN Entered → Transaction in Progress
- EjectCard → returns system to Idle
- Error → moves Transaction in Progress → Out of Service
- MaintenanceDone → moves Out of Service → Idle

(Hint: use start/end symbols to mark system start in "Idle" and termination if required.)

Step 4 – Specify Preconditions and Postconditions

Choose one operation and formally express its precondition and postcondition to show how it affects system state.

Example:

Operation: WithdrawCash

Precondition:

balance >= amount && currentState == PINEntered

Postcondition:

balance = old(balance) - amount && currentState = Idle