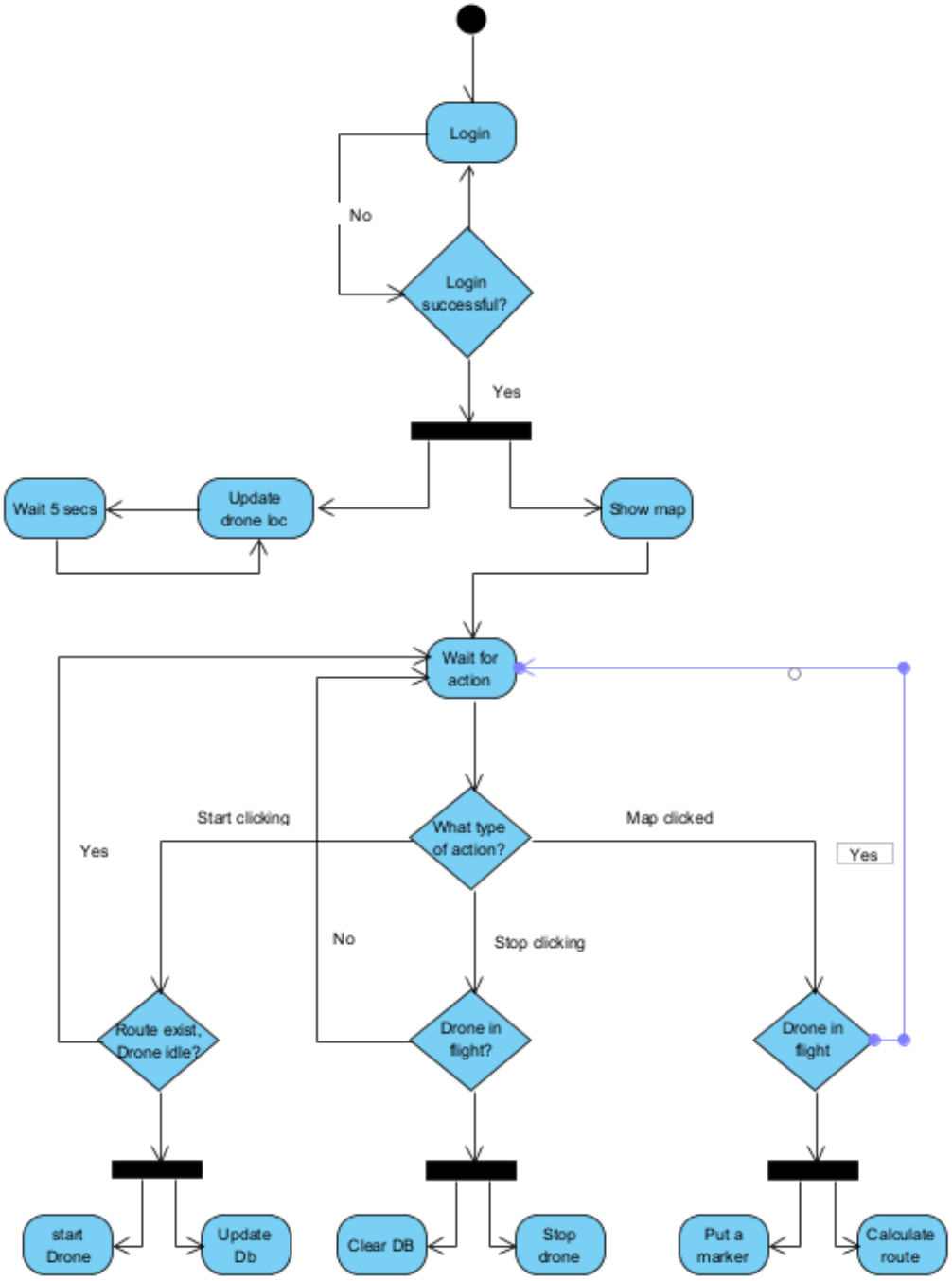
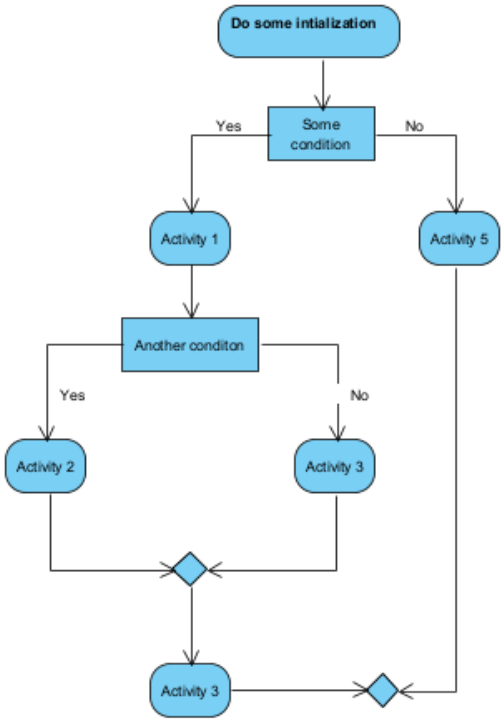
WEEK-3(Set-1)

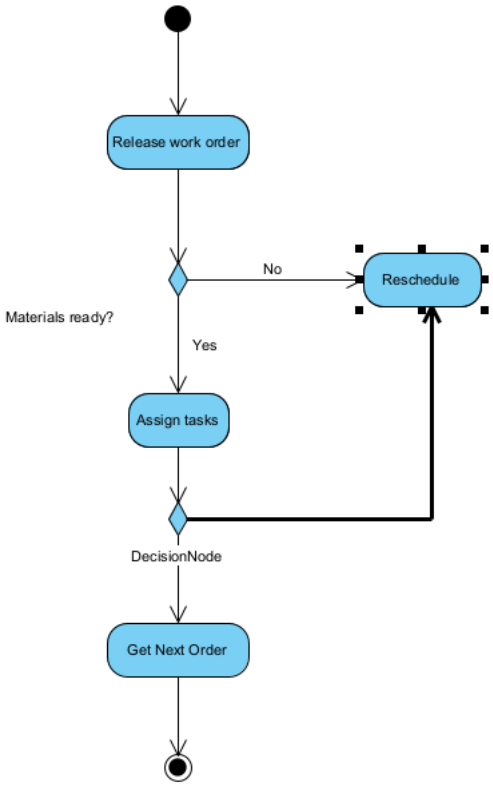
1) To implement uml activity diagram for android application



2) To implement uml activity start and end diagram.

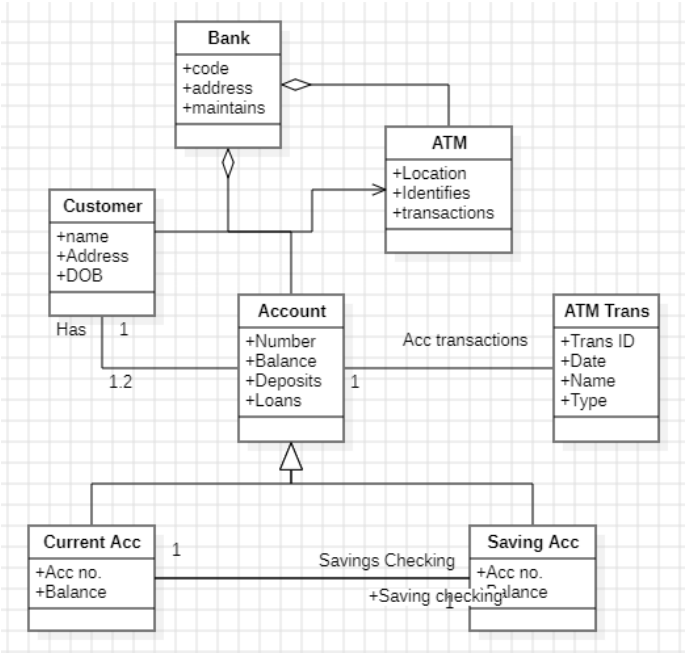


3) To implement uml activity branching diagram.



WEEK-4(Set-1)

A) To generate a java code from ATM class diagram.

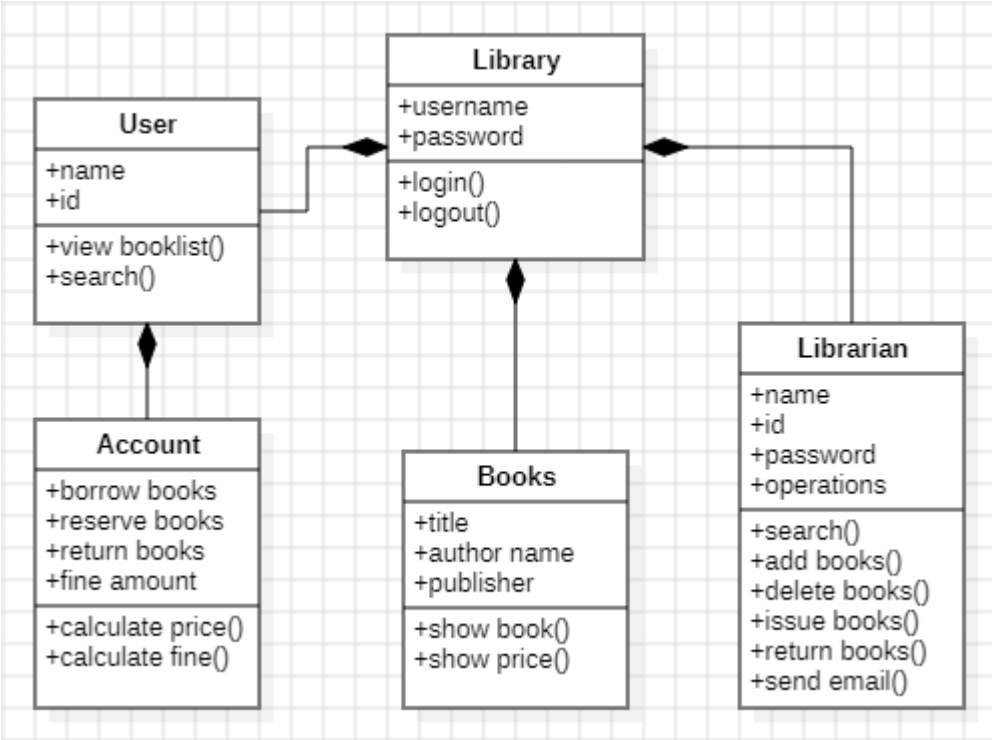


Code:

```java
import java.io.*;
import java.util.*;
public class Account {
    public Account() {
    }
    public void Number;
    public void Balance;
    public void Deposits;
    public void Loans;
}
```

```java
import java.io.*;
import java.util.*;
public class ATM {
    public ATM() {
    }
    public void Location;
    public void Identifies;
    public void transactions;
}
```

B) To generate a java code from Library info class diagram.

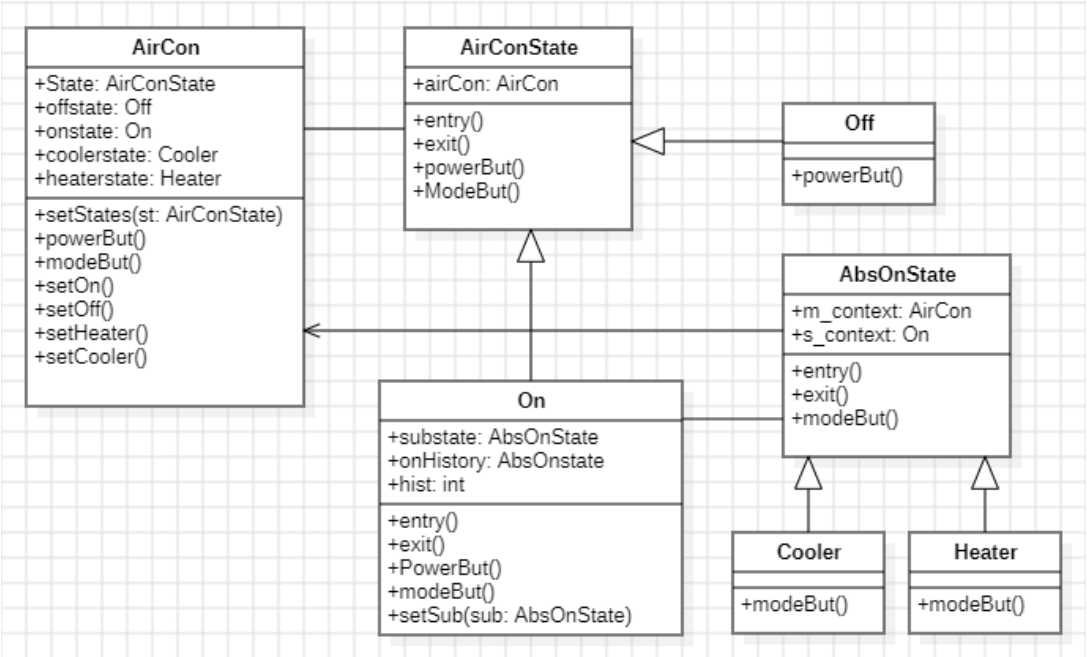


Code:

```java
import java.io.*;
import java.util.*;
public class Library {
    public Library() {}
    public void username;
    public void password;
    public void login() {}
    public void logout() {}
}
```

```java
import java.io.*;
import java.util.*;
public class Books {
    public Books() {}
    public void title;
    public void author name;
    public void publisher;
    public void show book() {}
    public void show price() {}
}
```

C) To generate a java code from Air Conditioner class diagram.



Code:

```java
import java.io.*;
import java.util.*;
public class AbsOnState {
    public AbsOnState() {}
    public AirCon m_context;
    public On s_context;
    public void entry() {}
    public void exit() {}
    public void modeBut() {}
}
```

```java
import java.io.*;
import java.util.*;
public class AirConState {
    public AirConState() {}
    public AirCon airCon;
    public void entry() {}
    public void exit() {}
    public void powerBut() {}
    public void ModeBut() {}
}
```

WEEK-5(Set-1)

1) Reverse the code into diagram for online grading system

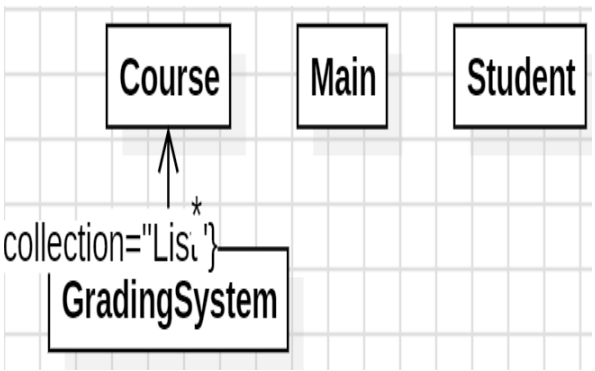
Steps:

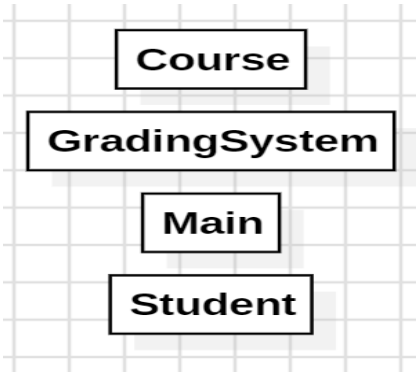
1. Install the Java tool form extensions in StarUML.
2. Save your Java code as .java files.
3. Import the .java files into StarUML (File > Import).
4. Go to **Tools** and select **Java Reverse Code Tool**.
5. Choose the imported Java file(s) to reverse-engineer.
6. Select diagram options like **Overview Attribute** and **Type Hierarchy**.
7. Generate and review the UML diagrams.

CODE:

```
public class Main {
    public static void main(String[] args) {
        // Create courses
        Course course1 = new Course("CS101", "Computer Science 101", 0.4f);
        Course course2 = new Course("MATH101", "Mathematics 101", 0.6f);
        // Create grading system
        GradingSystem gradingSystem = new GradingSystem();
        gradingSystem.addCourse(course1);
        gradingSystem.addCourse(course2);
        // Create students
        Student student1 = new Student("S1001", "Alice");
        Student student2 = new Student("S1002", "Bob");
        // Assign grades
        gradingSystem.assignGrade("S1001", "CS101", 85);
        gradingSystem.assignGrade("S1001", "MATH101", 90);
        gradingSystem.printStudentReport("S1001");
        gradingSystem.printStudentReport("S1002");
        System.out.println("Final grade for Alice: " + gradingSystem.calculateFinalGrade("S1001"));
        System.out.println("Final grade for Bob: " + gradingSystem.calculateFinalGrade("S1002"));
    }
}
```

Output:





2) Reverse the code into diagram for net banking system

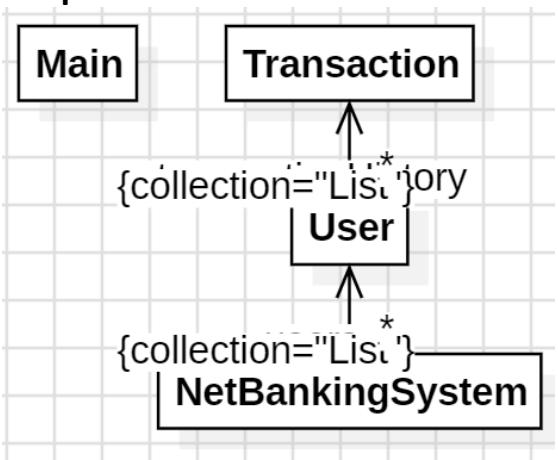
Steps:

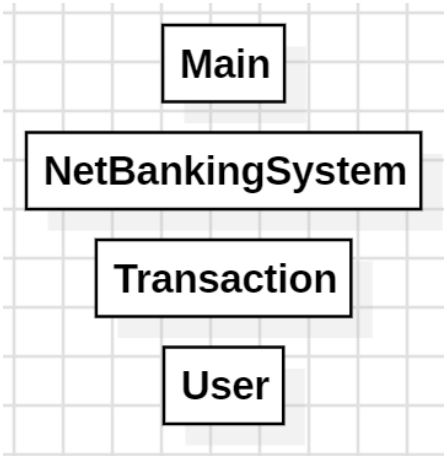
1. Install the Java tool form extensions in StarUML.
2. Save your Java code as .java files.
3. Import the .java files into StarUML (File > Import).
4. Go to **Tools** and select **Java Reverse Code Tool**.
5. Choose the imported Java file(s) to reverse-engineer.
6. Select diagram options like **Overview Attribute** and **Type Hierarchy**.
7. Generate and review the UML diagrams.

CODE:

```
public class Main {
    public static void main(String[] args) {
        NetBankingSystem bankingSystem = new NetBankingSystem();
        // Register users
        User alice = new User("A001", "Alice", "alice@example.com", "password123");
        User bob = new User("B001", "Bob", "bob@example.com", "password456");
        bankingSystem.registerUser(alice);
        bankingSystem.registerUser(bob);
        // Login user
        if (alice.login("alice@example.com", "password123")) {
            System.out.println("Login successful for Alice.");
        }        // Deposit money
        alice.deposit(1000);
        System.out.println("Alice's Balance: " + alice.getBalance());
        // Withdraw money
        alice.withdraw(500);
        System.out.println("Alice's Balance after withdrawal: " + alice.getBalance());
        // View transaction history
        System.out.println("\nAlice's Transaction History:");
        for (Transaction txn : alice.viewTransactionHistory()) {
            System.out.println(txn.getTransactionDetails());
        }
    }
}
```

Output:





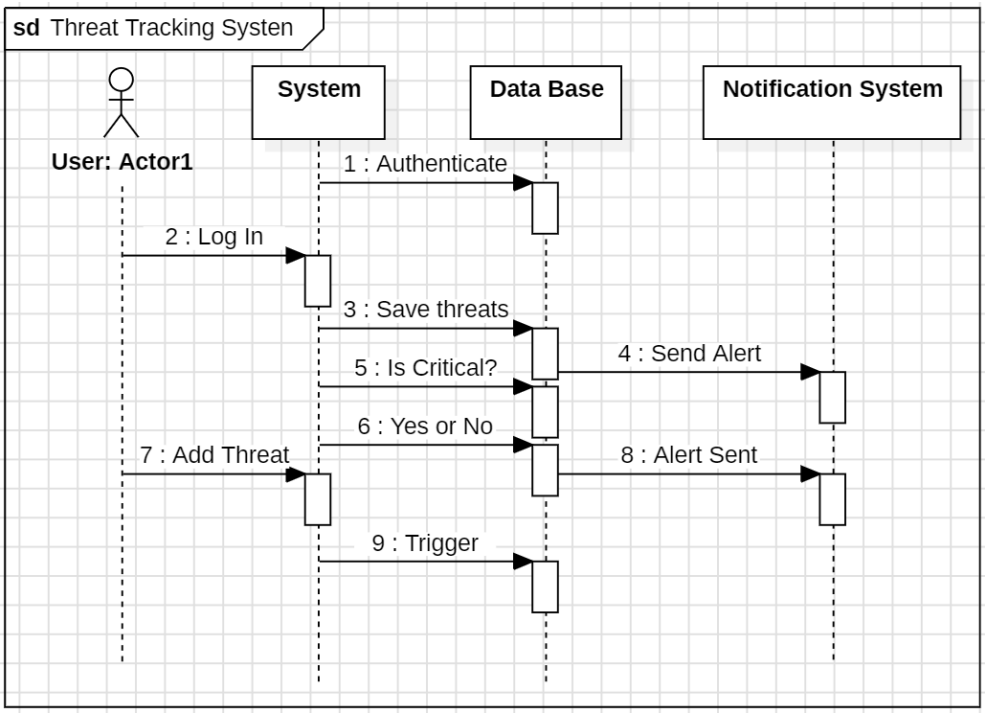
WEEK-6(Set-1)

1)Draw sequence diagram for Threat Tracking System.

Steps:

1. **Created the Class** – Named it **Threat Tracking System** to represent the process.
2. **Added an Actor** – Introduced **User (Actor1)** as the system's main user.
3. **Added Lifelines** – Created lifelines for **System, Database, and Notification System** to show interactions.
4. **User Logs In** – The system authenticates the user via the database.
5. **Threat is Added & Saved** – The user adds a threat, which the system stores in the database.
6. **Checks Criticality & Sends Alert** – If the threat is critical, the system alerts the **Notification System**.
7. **Triggers Actions** – The system takes further actions based on the threat status.

Output:



1) Draw sequence diagram Auto Spare Part Management System.

Steps:

1. **Created the Class** – Named it **Auto Spare Part Management System** to represent the process.
2. **Added an Actor** – Introduced **Customer(Actor1)** as the system's main user.
3. **Added Lifelines** – Created lifelines for **System, Database, and Notification System** to show
4. **User Logs In** – The system authenticates the user via the database.
5. **Threat is Added & Saved** – The user adds a threat, which the system stores in the database.
6. **Checks Criticality & Sends Alert** – If the threat is critical, the system alerts the **Notification System**.
7. **Triggers Actions** – The system takes further actions based on the threat status.

Output: