Ashlyn Goh 1002840 Cl02

50.003 – Problem Set 1

Cohort Exercise 1.2

Train API

```
Junction sourceJunction                  // field - Junction that Train started from

Junction destinationJunction             // field - Destination Junction

List<Junction> pathPlanner(Junction src, Junction dest)
         // function that takes in a source junction & destination junction and
         returns the route the train should take (in a List of Junctions)

boolean approachJunction(Junction j)     // function that is called when the train
                                         approaches a junction (takes in a junction
                                         as input). Checks if it needs to be held
                                         there to avoid collision, return True if
                                         so, else False

boolean approachDestination(Junction j)  // function that is called to check if
                                         Train is at its destination (& does not
                                         need to change track anymore)

void moveToDestination(Junction dest)    // function that is called to move train to
                                         destination, takes in the destination
                                         junction. Calls changeTrack when required
```

Junction API

```
void holdTrain(Train t)                  // function that is called to hold the
                                         train at a junction

void changeTrack(Train t, Junction j)    // function that is called to change a
                                         train’s track
```

Track API

```
Int isOccupied                           // field – 1 if occupied, 0 if not
```

Ashlyn Goh 1002840 Cl02

Cohort Exercise 1.3

Train API

```
Int typeOfTrain                          // field – 0:broad, 1:meter, 2:narrow

Junction sourceJunction                  // field - Junction that Train started from

Junction destinationJunction             // field - Destination Junction

void changeEngine()                      // function that changes engine if train is
                                         narrow gauge, else does not change engine


boolean approachJunction(Junction j)     // function that is called when the train
                                         approaches a junction (takes in a junction
                                         as input). Checks if it needs to be held
                                         there to avoid collision, return True if
                                         so, else False

boolean approachDestination(Junction j)  // function that is called to check if
                                         Train is at its destination (& does not
                                         need to change track anymore)

void moveToDestination(Junction dest)    // function that is called to move train to
                                         destination, takes in the destination
                                         junction

List<Junction> pathPlanner(Junction src, Junction dest)
         // function that takes in a source junction & destination junction and
         returns the route the train should take (in a List of Junctions)
```

Track API

```
Int typeOfTrack                          // field – 0:broad, 1:meter, 2:narrow

Int isOccupied                           // field – 1 if occupied, 0 if not

Void checkTrack(Train t, Junction j)     // function that checks train and track
                                         type, calls changeTrack(t,j) and
                                         changeEngine() accordingly
```

Junction API

```
void holdTrain(Train t)                  // function that is called to hold the
                                         train at a junction

void changeTrack(Train t, Junction j)    // function that is called to change a
                                         train’s track (considers type of train)
```

Ashlyn Goh 1002840 Cl02

Cohort Exercise 2

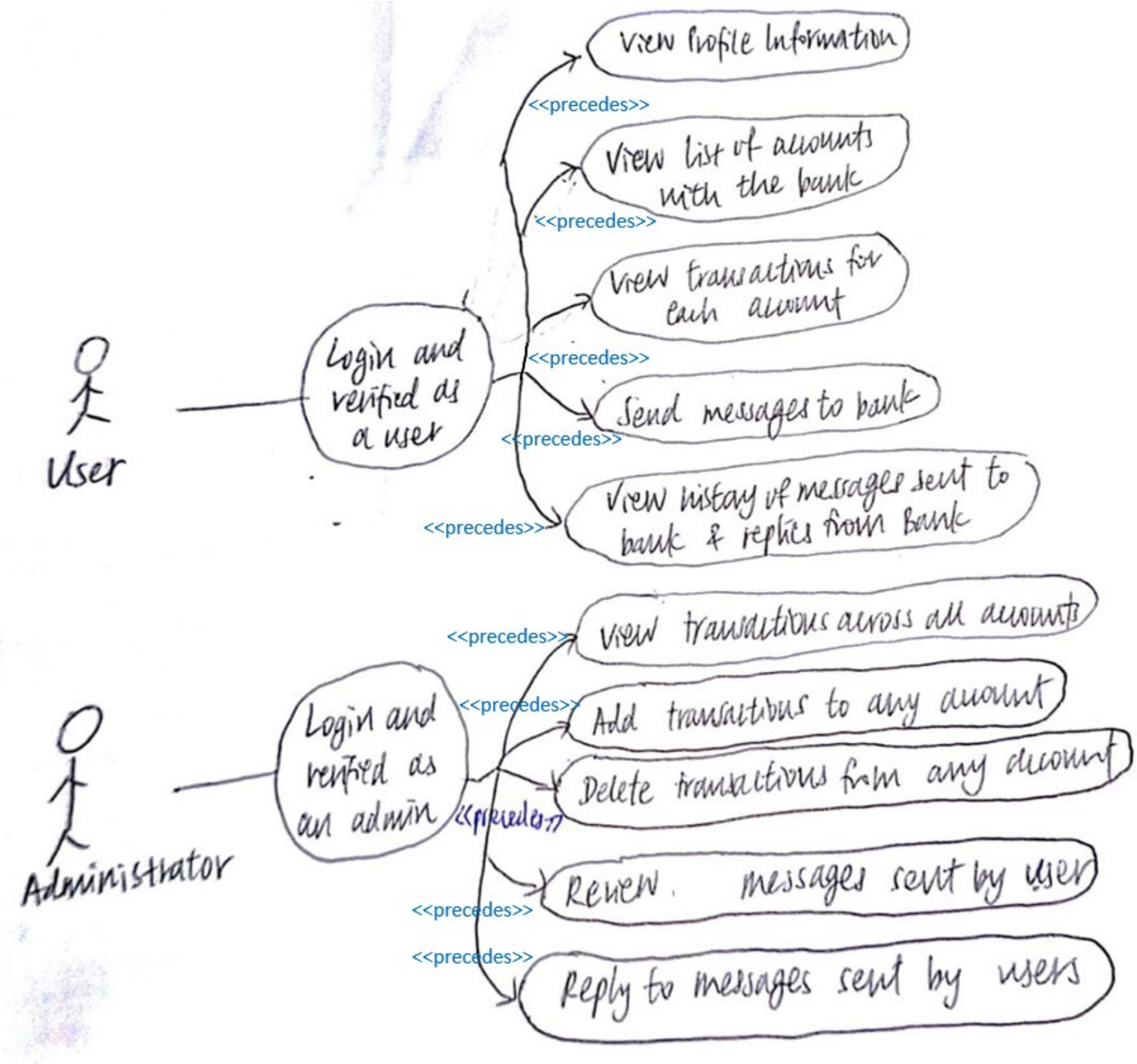
Design and implement a program that supports accepting two complex numbers from the user; adding, subtracting, multiplying, and/dividing them; and reporting each result to the user.

```
public class ComplexNumber
    double realVal                            // field
    double imagVal                            // field

public ComplexNumber(){                       // no-arg constructor
    this.realVal = 0;
    this.imagVal = 0;

public ComplexNumber(double r, double i){     // constructor
    this.realVal = r;
    this.imagVal = i; }

ComplexNumber addComplex(ComplexNumber c1, ComplexNumber c2){
    // function that adds given ComplexNumber c2 to c1
    return ComplexNumber(c1.realVal+c2.realVal, c1.imagVal+c2.imagVal);
    }


ComplexNumber subComplex(ComplexNumber c1, ComplexNumber c2){
    // function that subtracts given ComplexNumber c2 from c1
    return ComplexNumber(c1.realVal-c2.realVal, c1.imagVal-c2.imagVal);
    }

ComplexNumber multComplex(ComplexNumber c1, ComplexNumber c2){
    // function that multiplies given ComplexNumber c1 and c2
    double real = c1.realVal*c2.realVal - c1.imagVal*c2.imagVal;
    double imag = c1.realVal*c2.imagVal + c1.imagVal*c2.realVal;
    return ComplexNumber(real, imag);
    }

ComplexNumber divComplex(ComplexNumber c1, ComplexNumber c2){
    // function that divides given ComplexNumber c2 from c1
    double bottom = c2.realVal*c2.realVal + c2.imagVal*c2.imagVal;     //c2+d2
    double real = (c1.realVal*c2.realVal + c1.imagVal*c2.imagVal)/bottom;
    double imag = (c1.imagVal*c2.realVal - c1.realVal*c2.imagVal)/bottom;
    return ComplexNumber(real, imag);
    }

String toString(){          // used to report result to user
    if (this.imagVal == 0 && this.realVal == 0){
        return "0";
    }
    else if (this.imagVal == 0){
        // if complex number is in the form of bi
        return String.valueOf(this.realVal); }
    else if (this.realVal == 0){
        // if complex number is in the form of a
        return String.valueOf(this.imagVal) + "i"; }
    else {         // if complex number is in the form of a + bi
        return String.valueOf(this.realVal) + " " + String.valueOf(this.imagVal)+ "i";
    }
```

Ashlyn Goh 1002840 Cl02

Cohort Exercise 4

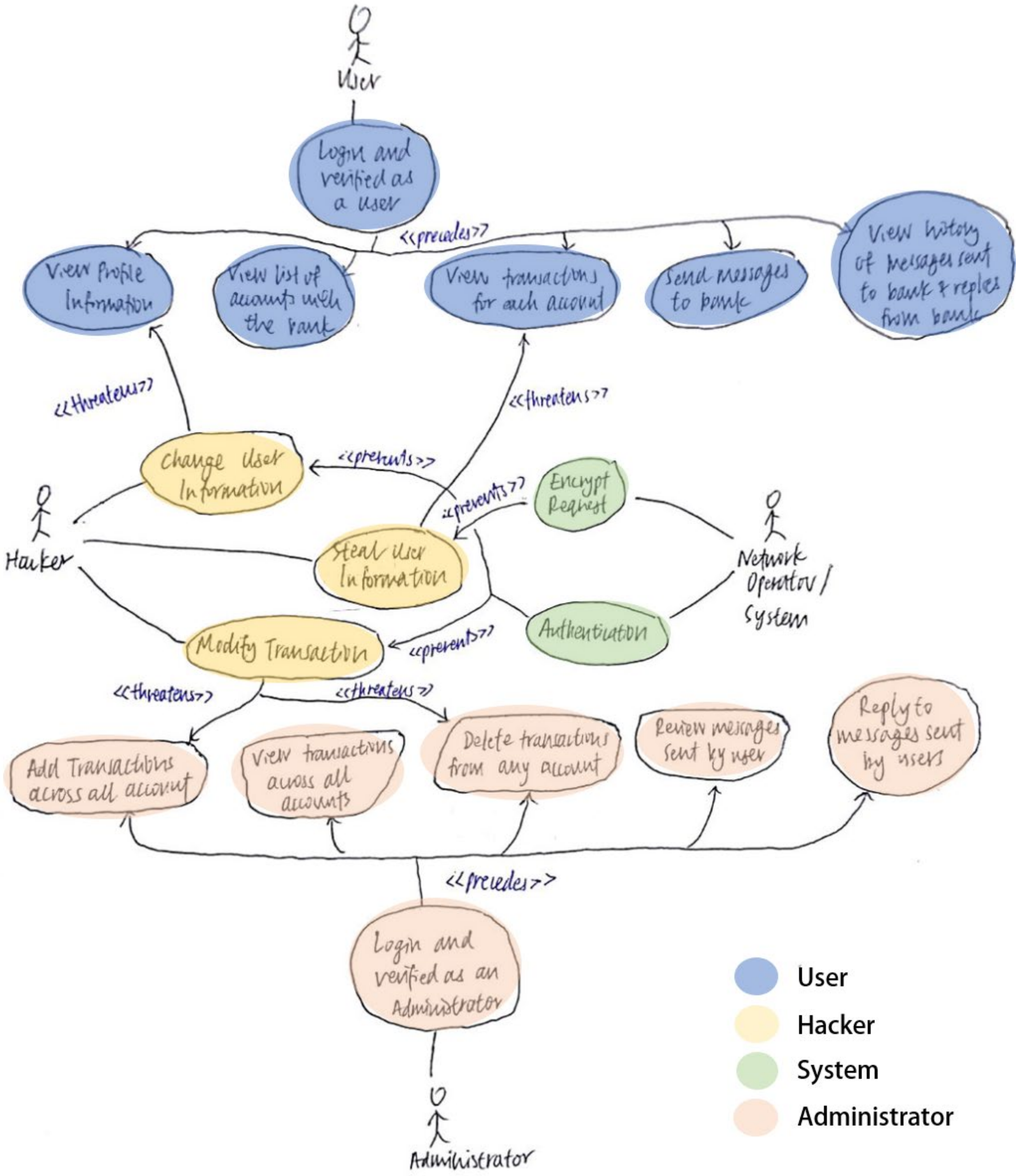
Draw a user case diagram for KBO (the online banking system discussed in the class).


User
Login and verified as a user
<<precedes>>
View Profile Information
<<precedes>>
View list of accounts with the bank
<<precedes>>
View transactions for each account
<<precedes>>
Send messages to bank
<<precedes>>
View history of messages sent to bank & replies from Bank
Administrator
Login and verified as an admin
<<precedes>>
View transactions across all accounts
<<precedes>>
Add transactions to any account
<<precedes>>
Delete transactions from any account
<<precedes>>
Review messages sent by user
<<precedes>>
Reply to messages sent by users

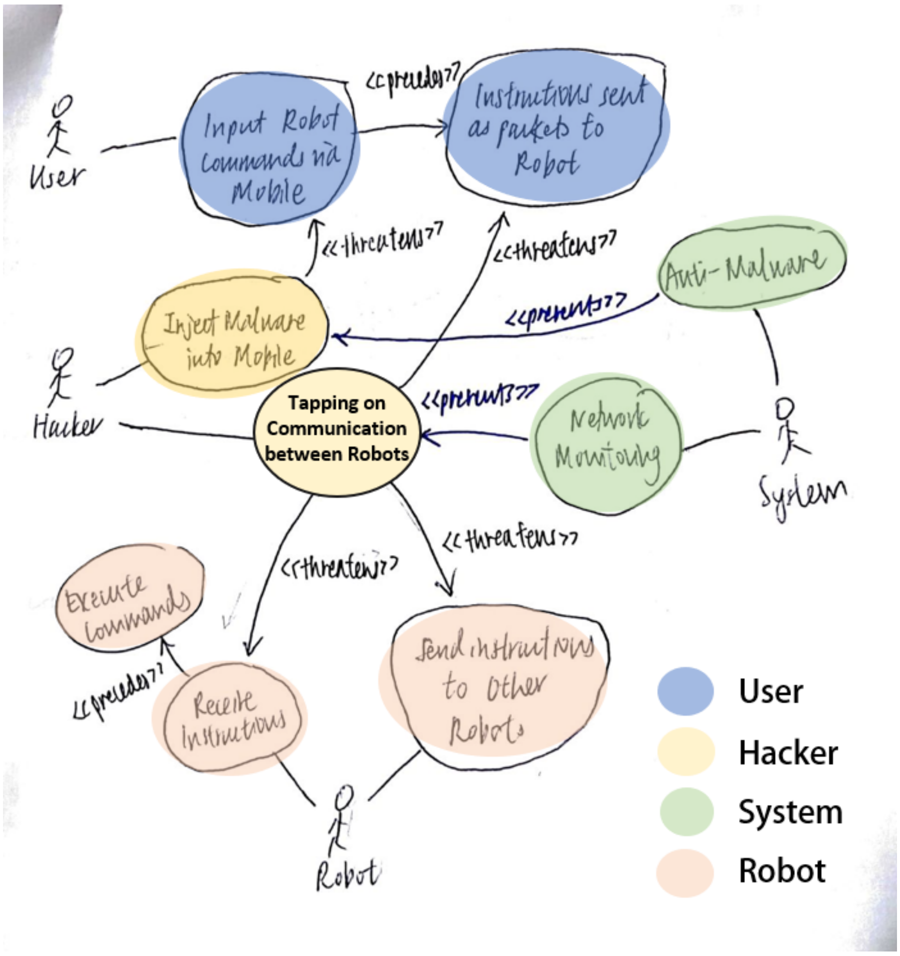
Ashlyn Goh 1002840 Cl02

Cohort Exercise 5

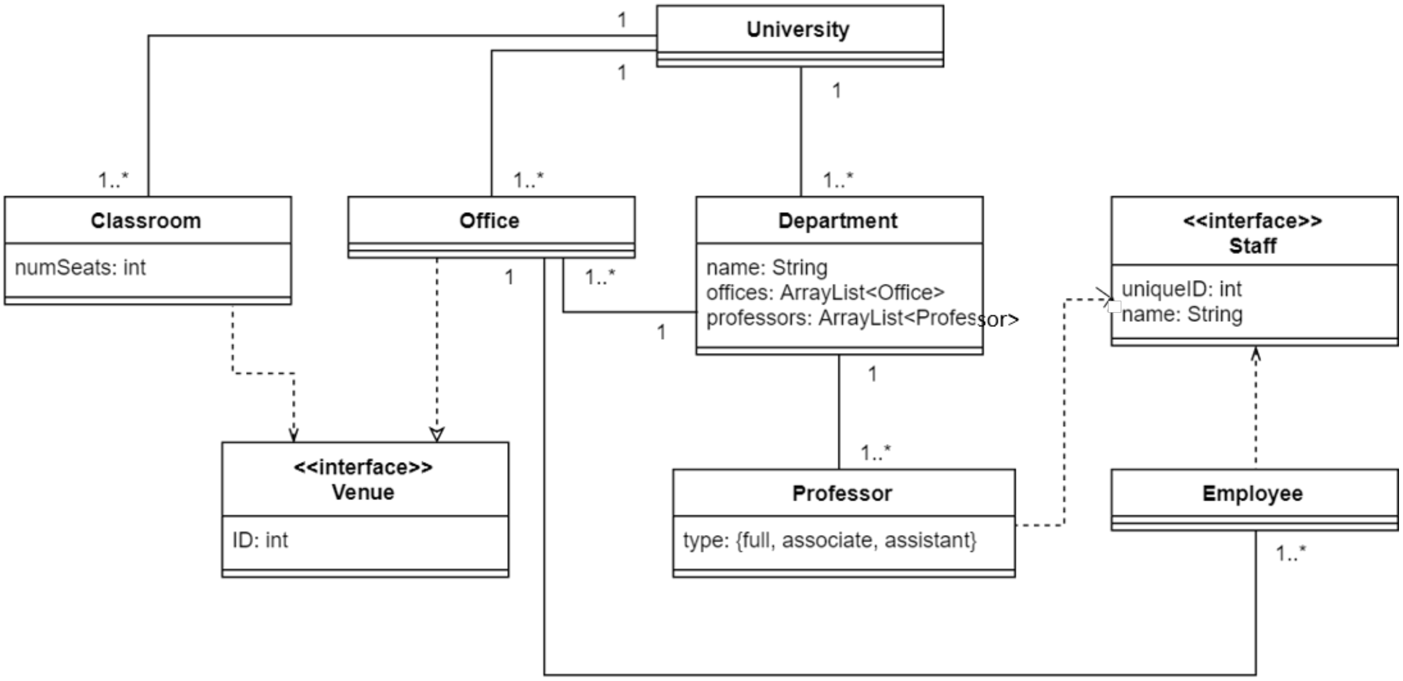
Augment the use case diagram for KBO with at least two misuse cases and additional use cases to prevent the misuse.


User
Login and verified as a user
<<precedes>>
View Profile Information
View list of accounts with the bank
View transactions for each account
Send messages to bank
View history of messages sent to bank & replies from bank
<<threatens>>
<<threatens>>
Change User Information
<<prevents>>
Hacker
<<prevents>>
Encrypt Request
Steal User Information
Network Operator / System
Authentication
Modify Transaction
<<prevents>>
<<threatens>>
<<threatens>>
Add Transactions across all account
View transactions across all accounts
Delete transactions from any account
Review messages sent by user
Reply to messages sent by users
<<precedes>>
Login and verified as an Administrator
Administrator
User
Hacker
System
Administrator

Cohort Exercise 6

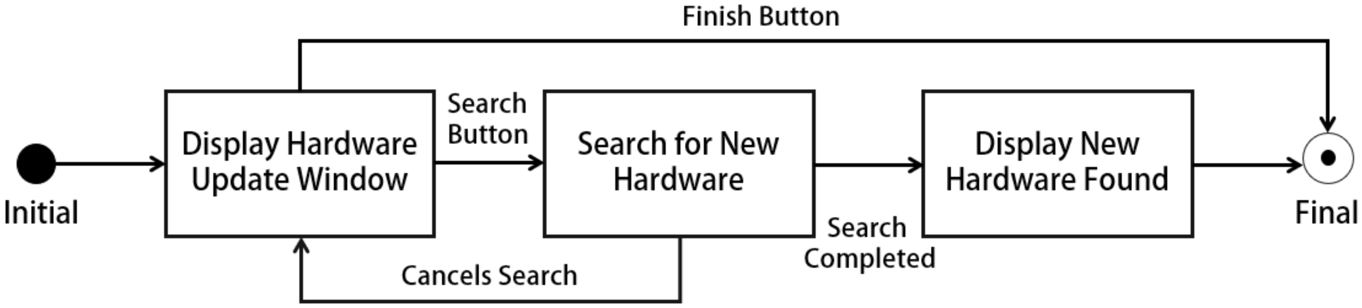


Cohort Exercise 7



Ashlyn Goh 1002840 Cl02

Cohort Exercise 8


Finish Button
Initial
Display Hardware Update Window
Search Button
Search for New Hardware
Cancels Search
Search Completed
Display New Hardware Found
Final