Lab 2: Role Play

1. Objectives

- To be able to apply the OOP inheritance concept.
- To be able to use the *extends*, *super*, and *instanceof* operators.
- To be able to perform upcasting and downcasting.
- To understand the concept of dynamic binding and apply it to make generic methods and data structures.
- To be able to use generic types.
- To understand the class UML diagram.
- To be able to use JUnit Test Framework to make a program without errors.

2. Problem Statement

In this lab, we will implement a program which simulates a role-playing game. The game consists of 4 roles: *person*, *merchant*, *thief*, and *police*. A person is the most basic role in the game, and anyone in the game is considered to be a person. A person might have a more specific role in the game and, thus, can also be either merchant, thief, or police.

Living in the city of capitalism, each person has a corresponding name and money. When any two persons meet, they interact with each other, i.e., when person A meets person B, person A will do person A's action to person B and vice versa. Each person might have different actions; the actions will be determined by the role they have, which is roughly specified below (the exact action behavior is described in the Implementation Details section):

- **Person**: A person will try to **buy** something from a merchant. Otherwise, he does nothing.
- **Merchant**: A merchant does not buy from anyone. He **sells** something to a buyer, but he can **remember** all thieves that stole his money. He can also **report** all thieves he remembered to the police.
- **Thief**: A thief **steals** money from everyone he meets (except police and another thief.)
- **Police**: A police can receive a theft report and then look for the thief involved in the theft. If the police meets the thief that has been reported, he will **catch** the thief, send him to jail, and return money to the victim. He does nothing if he meets a person who is not a thief.

In each day in the game, each person will meet some random person and do actions to each other. Assume that the city consists of 15 non-thief people, there are a total of 15 action pairs occurred each day. Note that because a thief can be sent to jail, the number of action pairs can be less than 15.

The program we are going to create simulates 20 days of the city and outputs all actions occurred within each day as well as the daily report. At the beginning of each day, a normal person that is not a merchant, thief, or police will get 100 units of money.

Your task is to complete the program using given template files. If implemented correctly, the resulting program should match the following output for during day 2 using given random seed (123456789):

```
Day : 2
Enter to pass a day...

thief1(Thief) steals 14 from merchant1(Merchant)
merchant1(Merchant) remembers thief1(Thief)
-----------------------------------
thief2(Thief) steals 6 from person2(Person)
person2(Person) meets thief2(Thief)
-----------------------------------
thief3(Thief) meets police2(Police)
police2(Police) catches thief3(Thief)
-----------------------------------
police1(Police) meets person2(Person)
person2(Person) meets police1(Police)
-----------------------------------
police2(Police) meets police1(Police)
police1(Police) meets police2(Police)
-----------------------------------
merchant1(Merchant) sells something to person1(Person)
person1(Person) buys something from merchant1(Merchant)
-----------------------------------
merchant2(Merchant) meets police2(Police)
police2(Police) meets merchant2(Merchant)
-----------------------------------
person1(Person) buys something from merchant1(Merchant)
merchant1(Merchant) sells something to person1(Person)
-----------------------------------
person2(Person) buys something from merchant1(Merchant)
merchant1(Merchant) sells something to person2(Person)
-----------------------------------
person3(Person) meets thief2(Thief)
thief2(Thief) steals 6 from person3(Person)
-----------------------------------
===================================
Thief| name : thief1, money : 56, stolenAmount : 21
Thief| name : thief2, money : 30, stolenAmount : 6
Police| name : police1, money : 0
Police| name : police2, money : 200
Merchant| name : merchant1, money : 119, price : 35
        Thief list : [thief1, thief1]
Merchant| name : merchant2, money : 42, price : 21
        Thief list : []
Person| name : person1, money : 182
Person| name : person2, money : 239
Person| name : person3, money : 232
===================================
Station
        Cases : [
                Victim : merchant1, Thief : thief1
        ]
        Jail : [thief3]
===================================
```

3. Instructions

1. Download and extract the provided ZIP file from myCourseVille.
2. Open Eclipse and then import existing project from the extracted folder into your Eclipse workspace.
3. Implement all classes and methods that are marked as TODO in the Implementation Details section using given template codes. Do not forget to check that your codes are correct.
4. Export your project into a JAR file named "`Lab2_2017_S2_{ID}.jar`", e.g. "`Lab2_2017_S2_5930000021.jar`". **Verify that your JAR file includes all your source codes and the JUnit tests.**
5. Submit the exported JAR file through myCourseVille.

4. Implementation Details

In this lab, you must create the program according to a UML diagram in Figure 1.

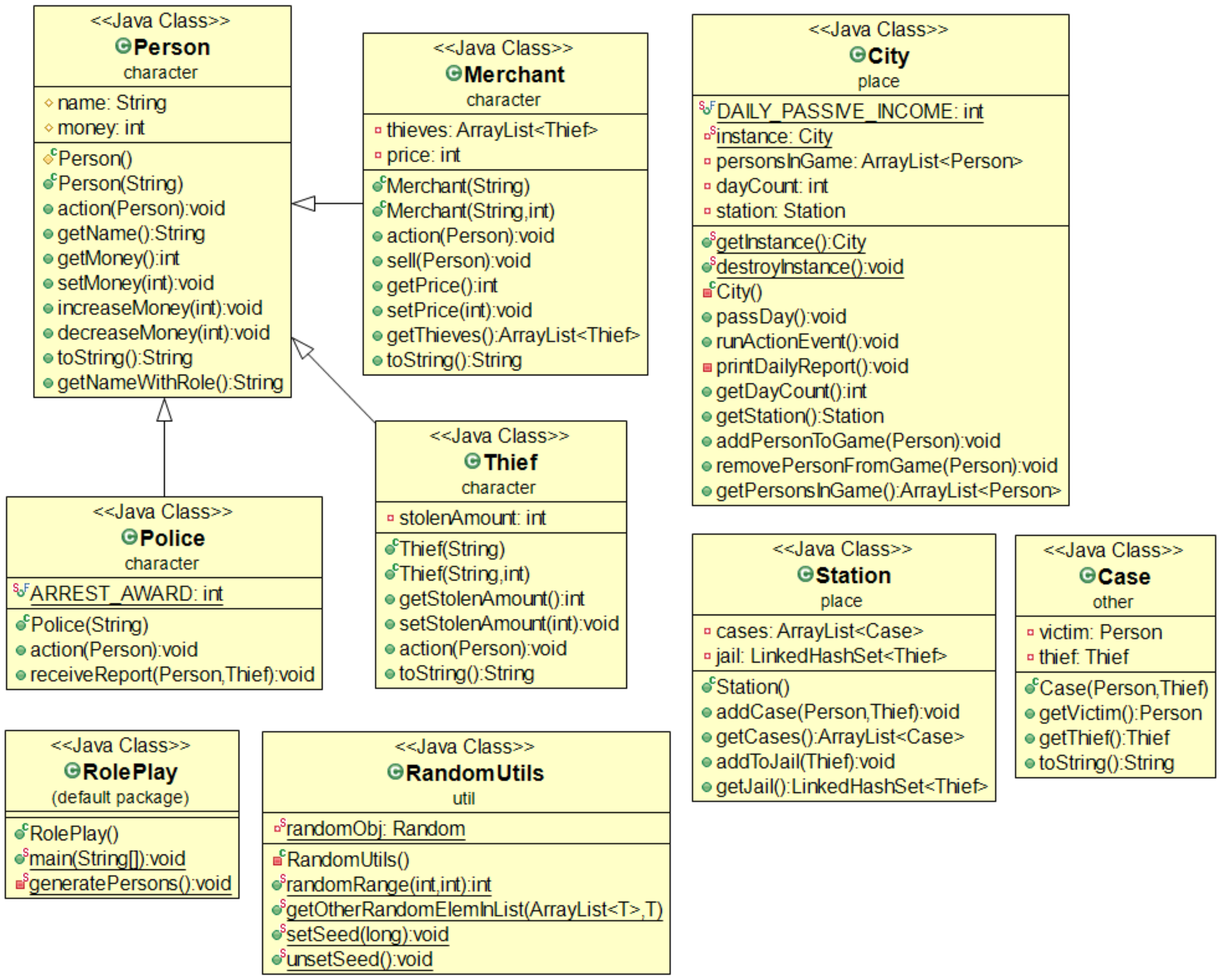

Figure 1: UML Diagram of the program

In the `action` methods within Person class and its subclasses, apart from doing the normal logic, you should print out the action using the code similar to the one below:

```
System.out.println(
    this.getNameWithRole() + " meets " + other.getNameWithRole());
```

For the example code above, if `this` is a person with name John and `other` is a thief with name Joe, it will output “`John(Person) meets Joe(Thief)`”. You should replace the middle string (“meets”) with the appropriate verb/phrase, for example:

- John(Person) meets Joe(Person)
- John(Person) buys something from Joe(Merchant)
- John(Merchant) sells something to Joe(Person)
- John(Merchant) remembers Joe(Thief)
- John(Merchant) reports thieves to Joe(Police)
- John(Thief) steals 50 from Joe(Person)
- John(Police) catches Joe(Thief)

You should always use the provided `RandomUtils` class to randomize a number. You should also test that your codes are correct. We provided some JUnit tests within the test package. The given JUnit tests should all pass.

4.1. Class character.Person

This class represents a person in the game.

4.1.1. Fields

`# name: String`	TODO. The name of the person.
`# money: int`	TODO. The amount of the money that this person currently has.

4.1.2. Constructors

`# Person()`	TODO. Creates a person with money equal to zero.
`+ Person(String name)`	TODO. Creates a person with specified name and money equal to zero. This constructor must utilize another constructor.

4.1.3. Methods

`+ action(Person other): void`	TODO. Does an action to `other`. If the other person is a merchant, this person must buy something from the merchant. Otherwise, he does nothing (apart from outputting).
`+ getName(): String`	TODO. Gets the name of this person.
`+ getMoney(): int`	TODO. Gets the amount of the money this person has.
`+ setMoney(int money): void`	TODO. Sets the amount of the money this person has. It must not allow the money to be below zero. In that case, it must set the money to exact zero.
`+ increaseMoney(int amount): void`	TODO. Increases the money by a specified amount. This method must use the `setMoney` method.

`+ decreaseMoney(int amount): void`	TODO. Decreases the money by a specified amount. This method must use the `setMoney` method.
`+ toString(): String`	Returns a String representation of this person.
`+ getNameWithRole(): String`	Returns a String representing the name and the role of this person.

4.2. Class character.Merchant (extends character.Person)

This class represents a merchant in the game.

4.2.1. Fields

`- thieves: ArrayList<Thief>`	TODO. A list of thieves that this merchant remembers.
`- price: int`	TODO. The price of an item that this merchant sells.

4.2.2. Constructors

`+ Merchant(String name)`	TODO. Creates a merchant with the specified name and random price between 11 and 50 (inclusively). This constructor must utilize another constructor within the same class. Use an appropriate method from `RandomUtils` class to random a value.
`+ Merchant(String name, int price)`	TODO. Creates a merchant with specified name and price. This constructor must utilize a constructor of the superclass to initialize the name and the money. It must also set the price by using the method `setPrice` to prevent invalid price.

4.2.3. Methods

`+ action(Person other): void`	TODO. Does an action to `other`. It should do as follows: - If the other person is a thief, this merchant should remember the thief. - If the other person is a police and this merchant currently remembers at least one thief, it should report all the thieves that he remembers to the police. It should also clear the set of remembered thieves after reporting. - Merchant does not sell things to a police, thief, or another merchant. This merchant should just meet `other` person. o If the other person does not fall into any of the conditions described above, this merchant must sell something to that person. (Note that there is no need to call `sell` method. It should only print out the output.)
`+ sell(Person buyer): void`	TODO. Sells something to specified buyer. Selling something increases merchant's money and decreases another person's money. It must check whether the buyer has enough money first. If the buyer's money is less than the price, do nothing.
`+ getPrice(): int`	TODO. Gets the price of the thing that this merchant sells.

`+ setPrice(int price): void`	TODO. Sets the price of the thing that this merchant sells. It must not allow the price to be below zero. In that case, this method must set the price to be zero.
`+ getThieves(): ArrayList<Thief>`	TODO. Gets a list of thieves that this merchant remembers.
`+ toString(): String`	Returns a String representation of this merchant.

4.3. Class character.Thief (extends character.Person)

This class represents a thief in the game.

4.3.1. Fields

`- stolenAmount: int`	TODO. The amount of the money that this thief can steal in each action.

4.3.2. Constructors

`+ Thief(String name)`	TODO. Creates a thief with specified name and random value of `stolenAmount` between 6 and 30, inclusively. This constructor must utilize another constructor from the same class. Use an appropriate method from `RandomUtils` class to random a value.
`+ Thief(String name, int stolenAmount)`	TODO. Creates a thief with specified name and `stolenAmount`. It must utilize a constructor of a superclass. It must also use the `setStolenAmount` method to prevent invalid value of `stolenAmount`.

4.3.3. Methods

`+ getStolenAmount(): int`	TODO. Gets the amount of the money that this thief can steal in each action.
`+ setStolenAmount(int stolenAmount): void`	TODO. Sets the amount of the money that this thief can steal in each action. It must not allow the `stolenAmount` to be below zero. In that case, this method must set the `stolenAmount` of this thief to be zero.
`+ action(Person other): void`	TODO. Does an action to `other`. If the other person is not a police or a thief, steal money from him. Otherwise, it should do nothing (apart from outputting). The thief must try to steal the money from the other person as much as possible without exceeding the amount of money that the other person currently has. Stealing money increases the thief's money and decreases the other person's money.
`+ toString(): String`	Returns a String representation of this thief.

4.4. Class character.Police (extends character.Person)

This class represents a police in the game.

4.4.1. Fields

`+ static ARREST_AWARD: int`	TODO. The amount of money that a police gets when he arrests a thief. It must be equal to 200.

4.4.2. Constructor

`+ Police(String name)`	TODO. Creates a police with specified name. This constructor must utilize a constructor of the superclass.

4.4.3. Methods

`+ action(Person other): void`	TODO. Does an action to `other`. If the other person is not a thief, this method should do nothing (apart from outputting). Otherwise, check whether the thief is involved in any theft cases. If so, for each case, do the followings: - Increase the victim's money by the thief's `stolenAmount`. - Decrease the thief's money by his own `stolenAmount`. - Add the thief to jail and remove the thief from the game. - Close the case by removing the case from the station. Also, increase the police's money by `ARREST_AWARD`.
`+ receiveReport(Person victim, Thief thief): void`	TODO. Receives a report of theft involving specified victim and thief. If the thief is not already in jail, it must add a theft case to the police station. Otherwise, do the followings: - Decrease the thief's money by the thief's `stolenAmount`. - Increase the victim's money by the thief's `stolenAmount`.

4.5. Class place.City

This class represents the city of the game. There can be only one city in the game.

4.5.1. Fields

`+ static DAILY_PASSIVE_INCOME: int`	TODO. The amount of the money that every person (except thief, police, and merchant) receives at the beginning of each day. It should be equal to 100.
`- static instance: City`	An instance of the city.
`- personsInGame: ArrayList<Person>`	TODO. A list of all persons that are currently in the game. Note that this list excludes the persons that are in the jail.
`- dayCount: int`	TODO. The number of days since beginning of the game. This must initially be zero.
`- station: Station`	TODO. An instance of the station belonging to this city.

4.5.2. Constructor

`- City()`	TODO. Creates a city and initializes non-static fields.

4.5.3. Methods

`+ static getInstance(): City`	Gets an instance of the city. It will create a new instance if not already existed.
`+ static destroyInstance(): void`	Destroys an existing instance of the city if exists. This should only be used in the JUnit tests.
`+ passDay(): void`	TODO. Simulates a day passed within the game. This method must increase the `dayCount` by one and increase each person's money by an amount of `DAILY_PASSIVE_INCOME`. Note that, according to the game rule, a thief, police, and merchant should not receive any money.
`+ runActionEvent(): void`	Runs an action event, which is a set of actions that will occurred within a day. This method also prints out daily report.
`- printDailyReport(): void`	Prints out the daily report which includes the persons that are currently in the game, the cases, and the persons that are in jail.
`+ getDayCount(): int`	TODO. Gets the number of days since beginning of the game.
`+ getStation(): Station`	TODO. Gets an instance of the station belonging to this city.
`+ addPersonToGame( Person person): void`	TODO. Adds the given person into the game if not already in.
`+ removePersonFromGame( Person person): void`	TODO. Removes the person from the game. If the person is already not in the game, this method does nothing.
`+ getPersonsInGame(): ArrayList<Person>`	TODO. Gets a list of all persons that are currently in the game.

4.6. Class place.Station

This class represents a police station.

4.6.1. Fields

`- cases: ArrayList<Case>`	TODO. An ArrayList consisting of the theft cases.
`- jail: LinkedHashSet<Thief>`	TODO. A LinkedHashSet consisting of the persons currently in the jail.

4.6.2. Constructor

`+ Station()`	TODO. Initializes all fields.

4.6.3. Methods

`+ addCase(Person victim, Thief thief): void`	TODO. Adds a new theft case involving specified victim and thief.
`+ getCases(): ArrayList<Case>`	TODO. Returns an ArrayList consisting of the theft cases.
`+ addToJail(Thief thief): void`	TODO. Adds the specified thief into the jail.
`+ getJail(): LinkedHashSet<Thief>`	TODO. Returns a LinkedHashSet consisting of the persons currently in the jail.

4.7. Class other.Case

This class represents a theft case.

4.7.1. Fields

`- victim: Person`	TODO. A victim involved in this case.
`- thief: Thief`	TODO. A thief involved in this case.

4.7.2. Constructor

`+ Case(Person victim, Thief thief)`	TODO. Creates a case involving specified victim and thief.

4.7.3. Methods

`+ getVictim(): Person`	TODO. Returns a victim involved in this case.
`+ getThief(): Thief`	TODO. Returns a thief involved in this case.
`+ toString(): String`	Return a String representation of this case.

4.8. Class RolePlay

A main class of the program.

4.8.1. Methods

`+ static main(String[] args): void`	A program entry point. It sets the random seed, generates persons in the game, and simulates the game time for 20 days.
`- static generatePersons(): void`	Generates persons in the game.

4.9. Class util.RandomUtils

This class contains utility methods for randomizing.

4.9.1. Useful Methods

`+ static randomRange( int min, int max): int`	Randomize an integer within the given range, including the minimum value and the maximum value.
`+ static getOtherRandomElemInList( ArrayList<T> list, T meElem): T`	Gets any random element in the specified list that is not the specified element `meElem`. This method returns a random element within the list that is not `meElem`, or null if the list has the size of 1 or less.