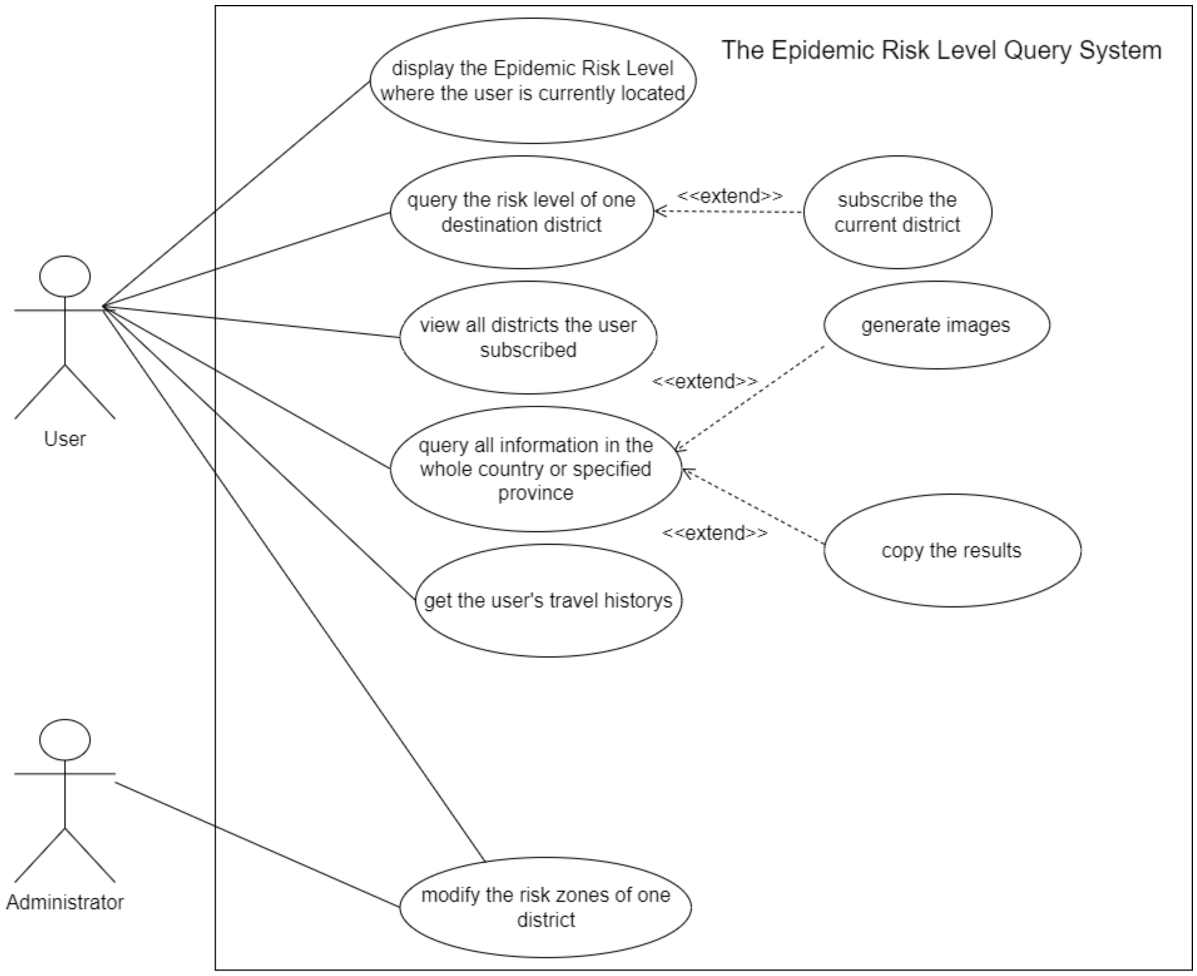
OOAD Assignment1

Name: 邱逸伦

SID: 12013006

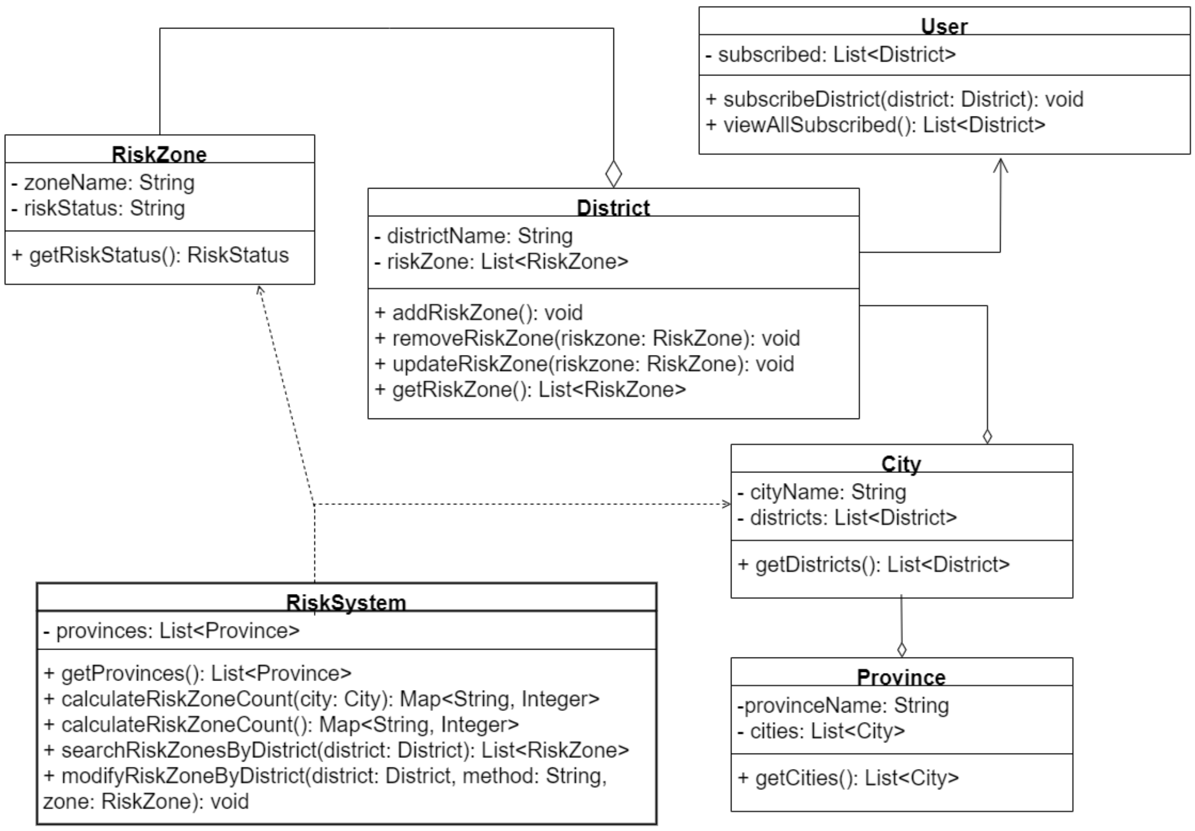
Question 1: Use case diagram



The Epidemic Risk Level Query System
display the Epidemic Risk Level where the user is currently located
query the risk level of one destination district
<<extend>>
subscribe the current district
view all districts the user subscribed
generate images
<<extend>>
query all information in the whole country or specified province
<<extend>>
copy the results
get the user's travel historys
User
Administrator
modify the risk zones of one district

In this use case diagram, I designed two actors: *User* and *Administrator*, and the system is called "*The Epidemic Risk Level Query System*". The use cases all serve as the function in the system while these two actors refer to the external element that interact with system. For the User, there are six main use cases. For the case "query the risk level of one destination district", there is one extended case "subscribe the current district". For the case "query all information in the whole country or specified province", there are two cases as its extended cases: "generate images" and "copy the results". For the Administrator, there is only one main use case "modify the risk zones of one district". But this case needs to notify all subscribers when the risk level changes. So it is also the main use case for the User viewed as the subscriber here.

Question 2: Class diagram


User
- subscribed: List<District>
+ subscribeDistrict(district: District): void
+ viewAllSubscribed(): List<District>
RiskZone
- zoneName: String
- riskStatus: String
+ getRiskStatus(): RiskStatus
District
- districtName: String
- riskZone: List<RiskZone>
+ addRiskZone(): void
+ removeRiskZone(riskzone: RiskZone): void
+ updateRiskZone(riskzone: RiskZone): void
+ getRiskZone(): List<RiskZone>
City
- cityName: String
- districts: List<District>
+ getDistricts(): List<District>
RiskSystem
- provinces: List<Province>
+ getProvinces(): List<Province>
+ calculateRiskZoneCount(city: City): Map<String, Integer>
+ calculateRiskZoneCount(): Map<String, Integer>
+ searchRiskZonesByDistrict(district: District): List<RiskZone>
+ modifyRiskZoneByDistrict(district: District, method: String, zone: RiskZone): void
Province
-provinceName: String
- cities: List<City>
+ getCities(): List<City>

In this class diagram, I designed seven classes: *RiskStatus*, *RiskZone*, *RiskSystem*, *District*, *City*, *Province*, and *User*. *All getter methods in this diagram can be regarded as the same as the getter methods in Java, I will skip them in the description.*

RiskZone: it is a zone with a specific risk level. It contains two fields: *zoneName* and *riskStatus*, here zo denotes the name of a specific riskzone and *riskStatus* is a String which describes the risk level of this riskzone. There are three possible values for *riskStatus*: blue, yellow, and red, which means low risk, medium risk, and high risk, respectively.

District: it is a district with several riskzones. It contains two fields: *districtName* and *riskZone*, here *districtName* denotes the name of a specific district and *riskZone* is a list of RiskZone which contains the riskzones belonging to the current district. There are several methods including *addRiskZone()*, *removeRiskZone(riskzone: RiskZone)*, and *updateRiskZone(riskzone: RiskZone)*, these methods can modify *riskZone*.

City: it is a city with several districts. It contains two fields: *cityName* and *districts*, here *cityName* denotes the name of a specific city and *districts* is a list of District which contains the districts belonging to the current city.

Province: it is a province with several cities. It contains two fields: *provinceName* and *cities*, here *provinceName* denotes the name of a specific province and *cities* is a list of City which contains the cities belonging to the current province.

User: it is a user class. Here the user has a field called *subscribed* which contains the districts the user already subscribed. This class has two methods: *subscribeDistrict* and *viewAllSubscribed*, here *subscribeDistrict* lets the user subscribe the specific district and *viewAllSubscribed* can return all districts that the current user has subscribed.

RiskSystem: it is a control class that has a field called *provinces* which contains all Province objects in China and several methods.

calculateRiskZoneCount(city: City) can calculate the number of different risk zones in the given city. In this method, the program will first query all districts in the given city, then query all riskzones in each district, finally, the program will count the number of riskzones and sum up them. The return type of calculateRiskZoneCount(city: City) is a map, the key is a String which describes the risk level and the value is an Integer. An entry in the return map is the number of riskzones of the corresponding risk status.

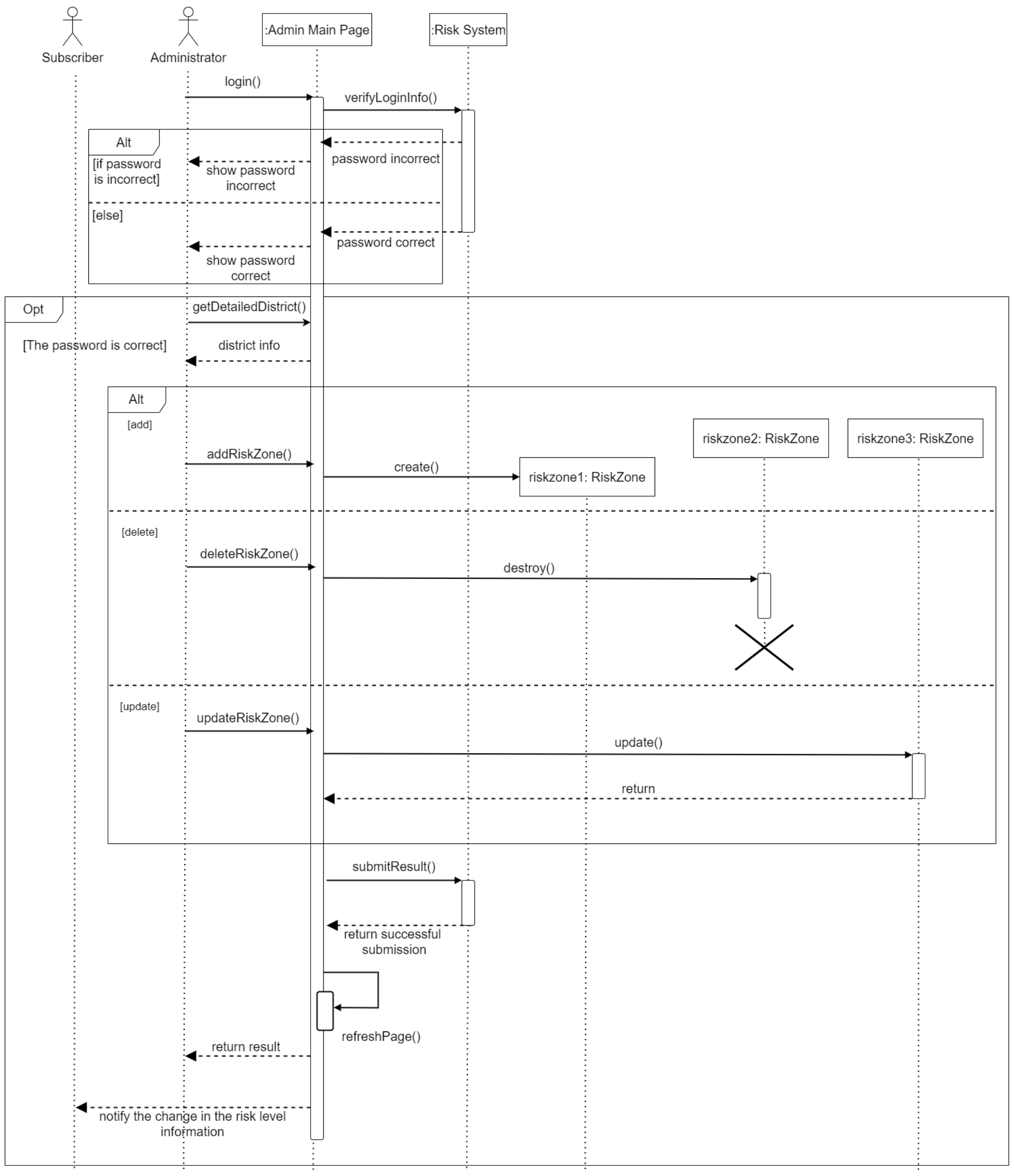
calculateRiskZoneCount() is just almost the same as the above method. The difference is that this method starts from the field *provinces* to get all Province objects in China, then queries all these provinces to get the cities which belong to each province. Then the counting method and the return type is just the same as the above method.

searchRiskZonesByDistrict(district: District) is based on the parameter *district* to get all riskzones in the given district, then return a list of RiskZone.

modifyRiskZoneByDistrict(district: District, method: String, zone: RiskZone) can operate the riskzone in the corresponding district. This method can first invoke the method *searchRiskZonesByDistrict(district: District)* to get the list of riskzone in the given district. The parameter *method* can only be "add", "remove", or "update", which corresponds to add riskzone, remove riskzone, and update riskzone. If the *method* equals "add", then the *district* can add the parameter *riskzone* to *riskZone*; if the *method* equals "update", then the *district* can invoke *updateRiskZone(riskzone: RiskZone)*; if the method equals "remove", then the *district* can invoke *removeRiskZone(riskzone: RiskZone)*.

Relationship: All relationships among these classes can be seen in the class diagram.

Question 3: Sequence diagram


Subscriber
Administrator
:Admin Main Page
:Risk System
login()
verifyLoginInfo()
Alt
[if password is incorrect]
show password incorrect
password incorrect
[else]
password correct
show password correct
Opt
[The password is correct]
getDetailedDistrict()
district info
Alt
[add]
addRiskZone()
create()
riskzone1: RiskZone
riskzone2: RiskZone
riskzone3: RiskZone
[delete]
deleteRiskZone()
destroy()
[update]
updateRiskZone()
update()
return
submitResult()
return successful submission
refreshPage()
return result
notify the change in the risk level information