COMP3021 Java Programming

Spring 2019 Programming Assignment #1 (Civilization – ASCII mode)

(Deadline: 11:55 PM, 16 April 2019)

Note:

- **This is an individual assignment; all the work must be your own.**
- **Download the project skeleton at https://course.cse.ust.hk/comp3021/assignments/assignment1/PA1.zip and finish the implementations using the skeleton.**
- **Zip your complete project directory to PA1.zip and submit it to CASS through https://course.cse.ust.hk/cass/student/. Refer to http://cssystem.cse.ust.hk/UGuides/cass/index.html for the details of uploading assignment using CASS.**
- **You can submit the assignment for as many times as you wish, we will only mark you latest submission.**
- **This PA accounts for 10% of the total course grade.**
- **Cheating/plagiarism will be caught and punished HEAVILY!**

  University's Honor code: http://tl.ust.hk/integrity/student-1.html
  University's Penalties for Cheating: http://tl.ust.hk/integrity/student-5.html

Background

Civilization is a turn based strategy game where the objective is to conquer other players' cities by developing our own resources and troops (reference). In this assignment, you will complete a partial implementation of a simplified, ASCII-version civilization game.

This game involves two human players (according to the configuration in the `players.txt` file that comes with the skeleton). In each turn, players are able to perform tasks on cities through their ministers. Each human player has a set of cities under his/her governance. The human player will have a list of ministers to carry out the various tasks on the cities. Some of the possible actions are: collect tax, collect science and production points, build improvements (banks, roads, universities) and recruit troops, see the `selectAndPerformAction()` method in the `GameEngine.java` file for the list of possible actions. Each minister has different intelligence, experience and leadership values, these values are stored in the instance variables of the `minister` object. These values affect how much gold, science and production collected. A minister of the player can do one single task in each turn. If the player has 3 ministers, the player will be able to perform three tasks per turn.

Gold, science points, and production points are the main resources of the game. The player can spend these resources to perform various tasks in the game, such as building banks, roads, universities, upgrading technologies, and recruiting troops. Using the recruited troops, players can attack adjacent cities to conquer them. The game will end when one player has conquered all cities on the map.

Please play with the supplied executable file on a Windows platform to see how the game could be played.

Getting Started

Importing the skeleton code

1. Download the skeleton code from the course website and unzip the file
2. In IntelliJ, go to `File > New > Project From Existing Sources...`
3. Choose the root directory of the skeleton code
4. Click Next, make sure the project format is .idea
5. When prompted to choose JDK version, choose `JDK 10`
6. Click finish

The List and Map interfaces

We use the List and Map interfaces extensively to store references to game objects. The List interface defines the set of methods you can do on a List (ArrayList, LinkedList, etc). Whereas the Map interface defines the methods of a Map (key, value data structure such as HashMap).

You can refer to the following documentations on the methods of these interfaces:

- List: https://docs.oracle.com/javase/10/docs/api/java/util/List.html
- Map: https://docs.oracle.com/javase/10/docs/api/java/util/Map.html

You will need the following methods for this assignment:

- List: size(), add(), get()
- Map: put(), get()

Since List and Map are interfaces, they can't be instantiated. Instead, you can assign a class that implements the interface. For example:

```
List<T> list = new ArrayList<>();
Map<K, V> map = new HashMap<>();
```

The `T`, `K`, `V` letters above are generic datatypes. See the following examples for the details of replacing these generic datatypes with a real existing datatype.

If you wish the ArrayList to store a particular type of elements, you will need to create an ArrayList and indicate the type of elements being stored. For example, the following statement:

```
List<Player> players=new ArrayList<Player>();
```

creates an empty ArrayList, `players`, which contains `player` reference as the elements. To add new player to the ArrayList, we can do:

```
// Create a player object with name being "Alex", and the
// initial gold being 100 units
Player player = new Player("Alex",100,100,100);

// add the "player" to the "players" ArrayList
Players.add(player);
```

A Map on the other hand acts as a “dictionary”. For each element in the Map, there are two fields, one is the Key (`K`), the other field is the Value (`V`). The lines below create a Map and each element of the Map consists of a String and a Double type data. Mind that the `K`, and `V` are generic datatypes, so they can be any non-primitive data types (but cannot be primitive).

```
Map<String,Double> fruitPrices=new HashMap<String,Double>();
```

After creating the Map, we add the prices of fruits to the Map `fruitPrices`:

```
fruitPrices.put("Apple",3.5);
fruitPrices.put("Pineapple",35.0);
```

The fruit name Strings (“Apple” and “Pineapple”) will become the Keys of the two entries, and if you wish to retrieve the Value field of “Pineapple”, you simply do this:

```
Double price=fruitPrices.get(“Pineapple”);
```

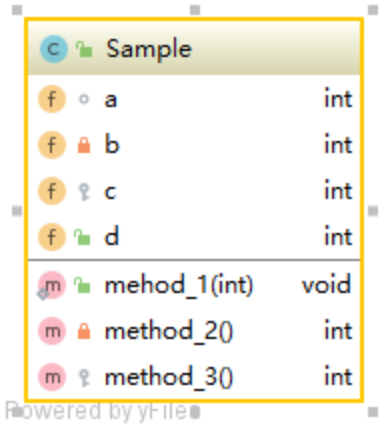
Then the variable, `price`, will hold the price of the “`Pineapple`”, which is `35.0`. As you can see, Map provides you an extremely convenient way of retrieving data using a more meaningful index (a String representing the name of a fruit here, instead of the meaningless index numbers 0,1,2,…n-1 in the case of an array).

To Use ArrayList and Map, you will need to have the following line in your Java file:

```
import java.util.*;
```

UML Diagram Notation

The UML diagrams used in this document is generated by IntelliJ (Ctrl + Alt + Shift + U to enter UML diagram edition mode). The notations used by IntelliJ is slightly different than the standard notations we discussed during our lectures. We will illustrate the difference using the Sample.java class UML diagram below:



The character "c" in the blue circle on the top row indicates that this is a class, the character "f" in the yellow circle indicates this is a field (i.e. variable), and the character "m" in the red circle indicates this is a "method". The hollow dot at the left of the variable "a" indicates that "a" has the "default" access right, the locked red lock at the left of the variable "b" indicates that "b" has the ~~"protected"~~ "private" access right, the gray key at the left of the variable "c" indicates that "c" has the ~~"private"~~ "protected" access right. The opened lock at the left of the variable "d" indicates that "d" has the "public" access right. For a method or a constructor, the input type is shown in the parentheses and the output type is shown at the right of the method name.

Tasks

Your tasks are to complete all the TODO items in the skeleton code. A detailed description of what you need to do is given in the Javadoc comments above each TODO. In IntelliJ, you can go to `View > Tool Windows > TODO` to jump to each TODO in the project.

More information about the TODO tasks are given below.

The TODOs in the GameEngine class (GameEngine.java)

You need to complete the following methods:

```
public boolean isGameOver() {
/**
* Determine if the game is over by checking if there is only one player with at least one city
*
* @return true if game is over, false otherwise
*/
}
```

```
public Player getWinner() {
/**
* Get the the only player with at least one city
*
* @return
*/
}
```

The TODOs in the GameMap class (GameMap.java)

You need to complete the following methods:

```
public void loadMap(String filename) throws IOException {
/**
* Load the map from a text file
* The outline of the map format is given in the assignment description
*
* You should instantiate cityLocations and connectedCities here
*
* @param filename
* @throws IOException
*/
}
```

This functions reads the content of map.txt and stores the map information inside these two data structures `Map<Integer, Cell> cityLocations` and `Map<Integer, List<Integer>> connectedCities`.

cityLocation is an associative array where the key is a unique ID of a city and the value is the location.

connectedCities is an associative array where the key is a unique ID of a city and the value is a list of neighboring city IDs.

The map text file is structured as follows:

```
48 // comment: this is the width of the text-mode game board
12 // comment: this is the height of the text-mode game board
2 //  comment: this is the number of cities in the game

0 16,4 // comment: this is the cityID,(x,y)-coordinates of the city
1 32,4//  comment: this is another city, cityID==1

0 1,2 // comment: this is the city with cityId==0, its neighboring cityIds
1 0,3 // comment: this is the city with cityId==1, and neighboring cityIds
```

public void loadPlayers(String filename) **throws** IOException {

```
/**
* Loads player data from text file
* The outline of the player file format is given in the assignment description
* You should instantiate the member variable players here
*
* @param filename
* @throws IOException
*/
```

}

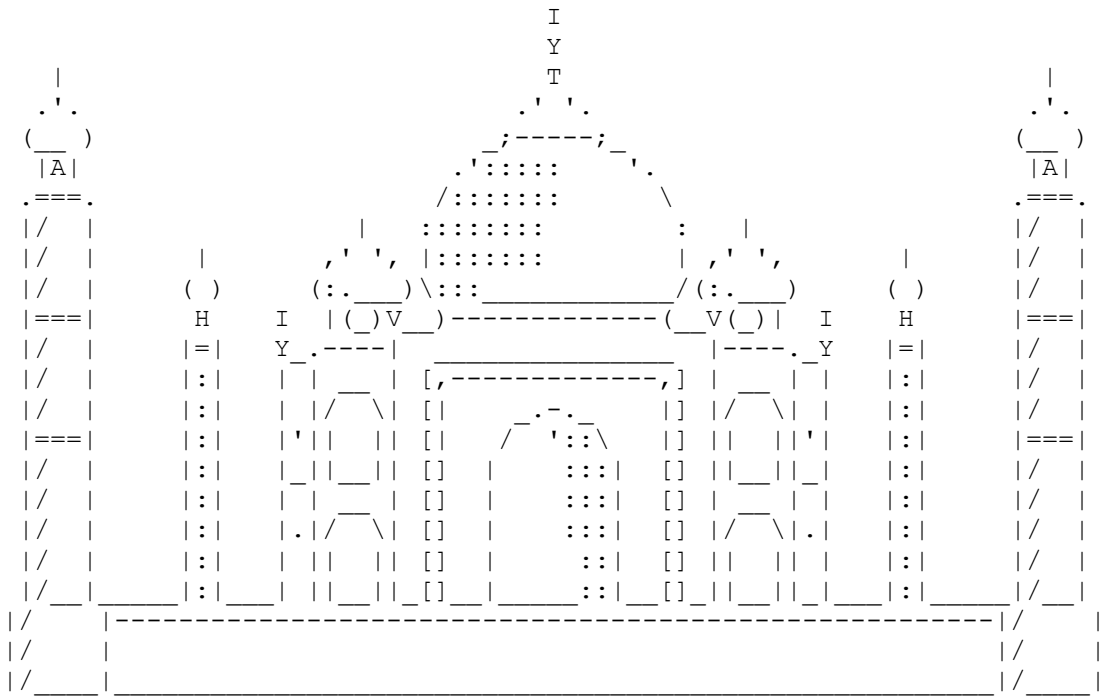
This method reads the content of players.txt and store player information in players list. The structure of the players text file is as follows:

```
2 // number of players

Blue 100 100 100 2 3  // playerName, gold, science, production, cities, ministers
1 SF 300 50 500  // cityId,name,population,troops,crop yields
3 LA 500 100 600 // cityId,name,population,troops,crop yields
Economist 80 100 60 // ministerType,intelligence,experience,leadership
WarGeneral 100 80 60 // ministerType,intelligence,experience,leadership
WarGeneral 60 80 100// ministerType,intelligence,experience,leadership
                      // note the blank line between each player
Red 100 100 100 2 2     // Player 2, the "Red" player and its information
```

```
0 BJ 200 0 100
2 HK 500 0 800
Economist 80 80 80
Scientist 100 100 100
```

```
public List<City> getAllCities() {
/**
* @return list of all cities from every player
*/
}
```

```
public Cell getCityLocation(City city) {
/**
* @param city
* @return Cell object representing the city's location on the game map
*/
}
```



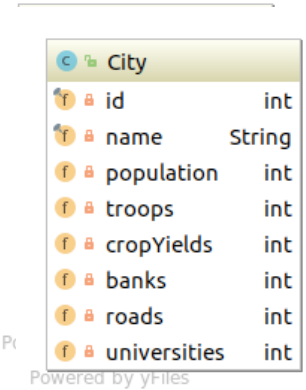
The TODOs in the Player Class(Player.java)

The player class contains the cities, ministers, technologies of the player as well as gold, production and science points. Please complete the following methods:

```
public Player(String name, int gold, int science, int production) {
  technologies.add(new WarTech());
  technologies.add(new TradingTech());
  technologies.add(new ManufacturingTech());

/**
* Initializes member variables.
*
* @param name
* @param gold
* @param science
* @param production
*/
  // TODO: complete initialization of member variables  using the parameters
}


public boolean hasCity(City city) {
/**
* @param city
* @return true if the given city belongs to this player
*/
}


public boolean hasAnyCity() {
/**
* Hint: use a method in the List class
*
* @return true if this player has at least one city
*/
}
```

```
public boolean hasReadyMinister() {
/**
* @return true if the player has at least one ready minister
*/
}
```

The TODOs in the City Class (City.java)



In addition to completing unfinished methods, you need to declare the member variables according to the UML diagram. The lock symbol indicates that the variable is private and the pin symbol indicates final variables.

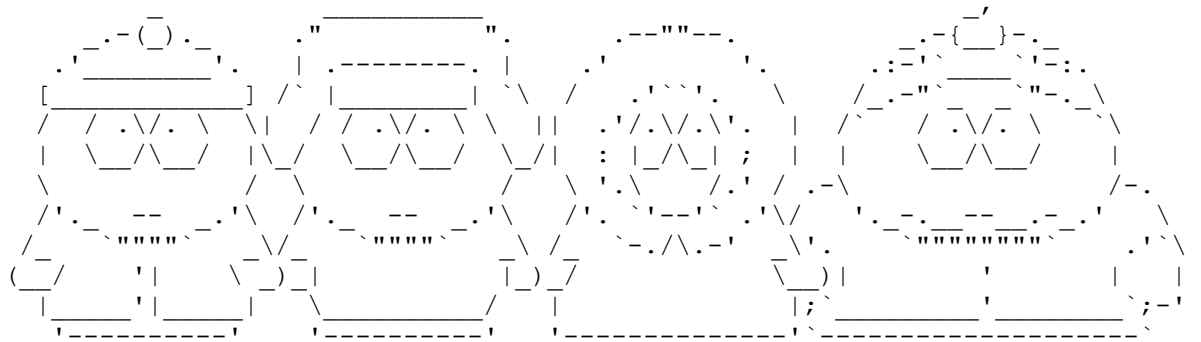
```
public void addBank() {
/**
* Adds number of banks by one
*/
}
```

```
public void addRoad() {
/**
* Adds number of roads by one
*/
}
```

```
public void addUniversity() {
/**
* Adds number of universities by one
*/
}
```

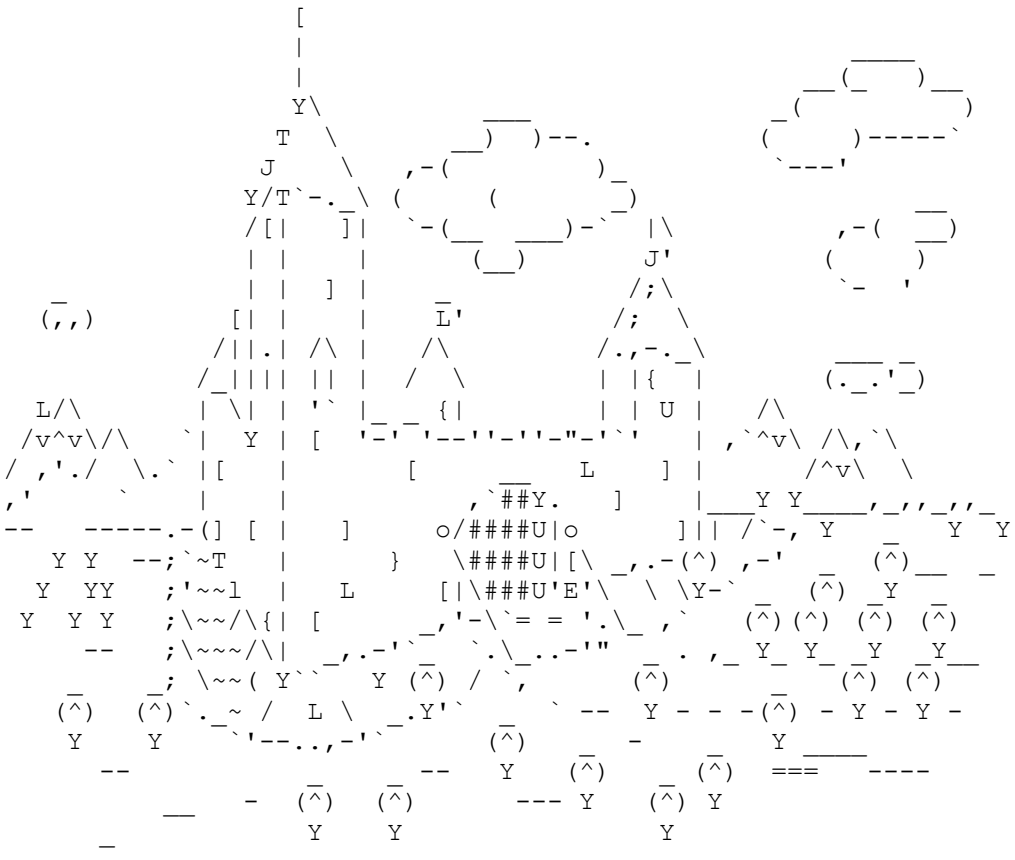
```
public void addTroops(int increment) {/**
* Adds number of troops by specified increment.
* If the increment is negative, invoke decreaseTroops() within this method
instead.
*
* @param increment
*/
}

public void decreaseTroops(int decrement) {
/**
* Decreases number of troops by the amount specified
* Caps the number of troops at 0
*
* @param decrement
*/
}

public void improveCrops(int addition) {
/**
* Add to crop yields the amount specified
*
* @param addition
*/
}

public int getExcessCrops() {
/**
* Calculates excess crops.
*
* @return crop yields - population - 2 * troops
*/
}
```

```
@Override
public boolean equals(Object o) {
/**
* Checks whether two cities are equal
* Two cities are equal when they have the same id
* Return false if Object o is not an instance of City
*
* @param o object to be compared
* @return result of equality check
*/
}


public void growAtTurnEnd() {
/**
* Increases population by increment = round(excess crops * 0.5)
* If the increment turns out to be negative, leave population unchanged
* Print a line that says:
* "Turn end: [city name]'s population has grown by [increment]
* e.g. "Turn end: HK's population has grown by 50"
*/
}
```



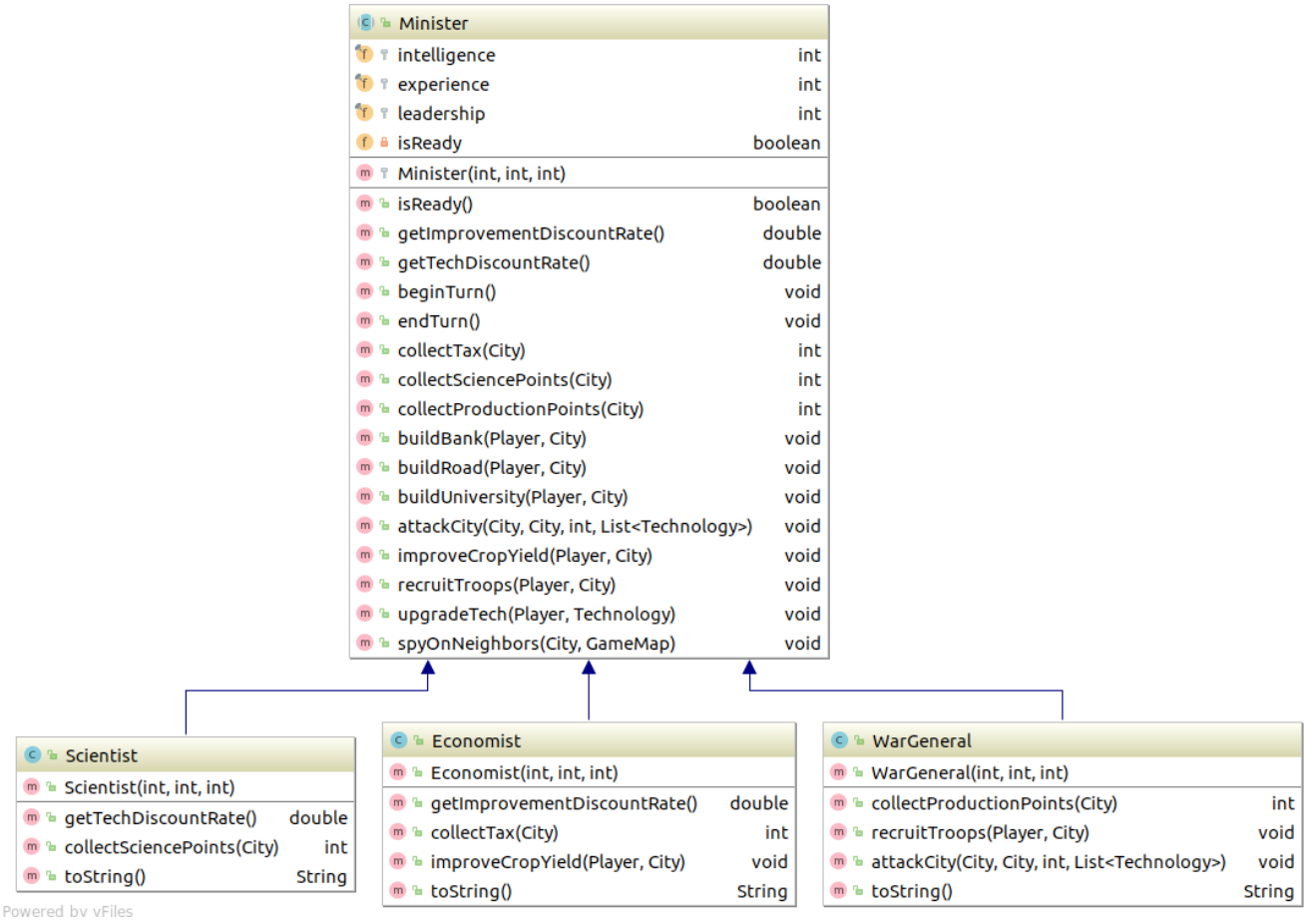
```
public void invokeRandomEvent(double rand) {

/**
* Invoke a random event at the end of turn.
* There are two types of events, a disaster and baby boom.
* A disaster happens when rand <= 0.4, it halves the population.
* A baby boom happens when rand > 0.4 AND rand <= 0.8 it multiplies the
population by 1.5.
* Otherwise the population is left unchanged
* Print a message in the following format when each event happens
* "Random event: A disaster in [city name] has happened, population was
reduced significantly"
* "Random event: A baby boom in [city name] has happened, population was
increased significantly"
*
* @param rand random number between 0 and 1 supplied by the function caller
*/
}

public Cost getBankCost() {
/**
* Cost of building a bank
* gold = production points = (# of banks + 1) * 400
*
* @return Cost object encapsulating the gold and production costs
*/
}

public Cost getRoadCost() {
/**
* Cost of building a road
* gold = production points = (# of roads + 1) * 100
*
* @return Cost object encapsulating the gold and production costs
*/
}
```

```
public Cost getUniversityCost() {
/**
* Cost of building a university
* gold = production points = (# of universities + 1) * 1500
*
* @return Cost object encapsulating the gold and production costs
*/
}
```



The TODOs in the pa1.ministers *package* (in the ministers folder)

You need to complete the implementation of the Minister abstract class (Minister.java) and its subclasses. The Minister abstract class defines the general behavior of a minister when performing tasks. The subclasses (WarGeneral, Scientist, Economist) introduce some task-specific perks and bonuses which are implemented as method overriding. Since actions in the game are done through ministers, these classes will contain most of your work.

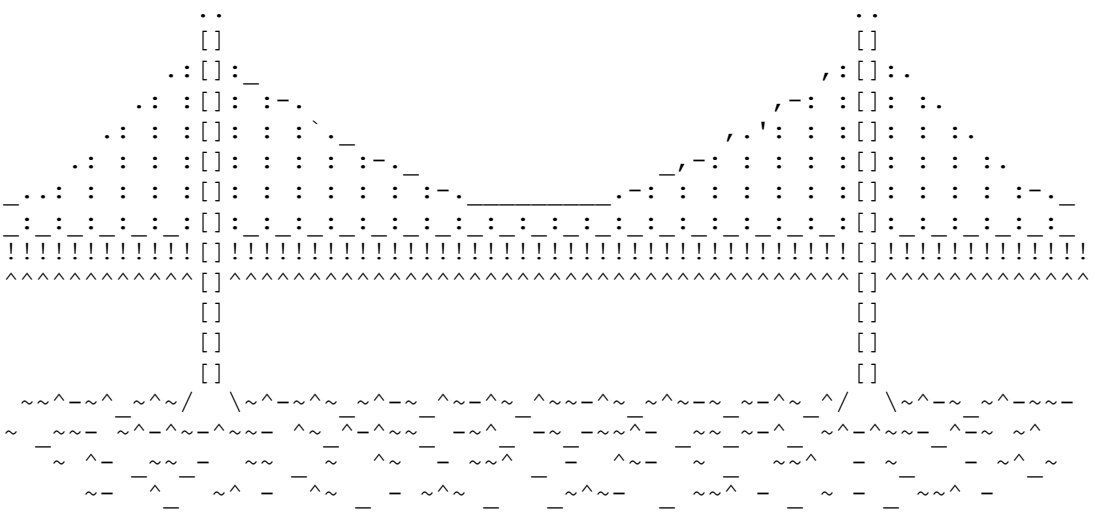


The TODOs in the Minister Abstract Class (Minister.java)

```
public boolean isReady() {
/**
* @return Whether or not the minister is ready
*/
}

public void beginTurn() {
/**
* Changes isReady to true
*/
}

public void endTurn() {
/**
* Changes isReady to false
*/
}

public int collectTax(City city) {
/**
* Collect gold from a city
* amount collected = (city population + minister experience + minister leadership)
* (# of banks  + 1) * 0.2
*
* @param city to collect gold from
* @return amount of gold collected
*/
}
```

```
public int collectSciencePoints(City city) {
/**
* Collect science points from a city
* amount collected = (city population + minister experience + minister
intelligence) * (# of universities  + 1) * 0.2
*
* @param city to collect science points from
* @return amount of science points collected
*/
}

public int collectProductionPoints(City city) {
/**
* Collect production points from a city
* amount collected = (city population + minister intelligence + minister leadership)
* (# of roads  + 1) * 0.2
*
* @param city to collect production points from
* @return amount of production points collected
*/
}


public void buildBank(Player player, City city) throws TooPoorException {
/**
* Build a bank in the city
* 1. Get the cost of building a bank, with applied minister discount
* 2. Check whether player has enough gold and production points
* 3. If not, throw an exception
* 4. Subtract the cost from the player's gold and production point
* 5. Add number of bank in the city by one
* HINT:
* - the Cost class has a getDiscountedCost() method
* - the Minister class has a getImprovementDiscountRate() method, to obtain the
* discount rate of building a bank
*
* @param player
```

```
* @param city
* @throws TooPoorException
*/
}


public void buildRoad(Player player, City city) throws TooPoorException {
/**
* Build a road in the city
* 1. Get the cost of building a road, with applied minister discount
* 2. Check whether player has enough gold and production points
* 3. If not, throw an exception
* 4. Subtract the cost from the player's gold and production point
* 5. Add number of road in the city by one
* <p>
* HINT:
* - the Cost class has a getDiscountedCost() method
* - the Minister class has a getImprovementDiscountRate() method, to obtain the
* discount rate of building a road
*
* @param player
* @param city
* @throws TooPoorException
*/
}
```



```
public void buildUniversity(Player player, City city) throws TooPoorException {
/**
* Build a university in the city
* 1. Get the cost of building a university, with applied minister discount
* 2. Check whether player has enough gold and production points
* 3. If not, throw an exception
* 4. Subtract the cost from the player's gold and production point
* 5. Add number of university in the city by one
* <p>
* HINT:
* - the Cost class has a getDiscountedCost() method
* - the Minister class has a getImprovementDiscountRate() method, to obtain the
* discount rate of building a university
*
* @param player
* @param city
* @throws TooPoorException
*/
}
```

```
public void attackCity(City attacker, City defender, int attackingTroops,
List<Technology> technologyList) {
/**
* Attack a target city
* Attacking city loses troops equal to min(# of troops attacking, # of troops in the
defending city)
* Defending city loses round(0.8 * # of attacking troops * product of tech attack
bonuses)
* <p>
* This method is overridden in the WarGeneral class
*
* Print the following messages:
* "[attacker city name] loses [number of troops lost by attacker] troops while
attacking"
* "[defender city name] loses [number of troops lost by defender] troops while
defending"
```

```
*
* @param attacker       attacking city
* @param defender       defending city
* @param attackingTroops number of troops deployed for the attack
* @param technologyList  technologies owned by the attacking player
*/
}

public void improveCropYield(Player player, City city) throws
TooPoorException {
/**
* Improve the crop yields in a city
* Improve Crop yields by 50 for 500 golds
* If the player does not have enough gold, throw an exception
* Else decrease 500 golds from the player and improves crops by 50
* <p>
* This method is overridden in the Economist class
*
* @param player
* @param city
* @throws TooPoorException
*/
}

public void recruitTroops(Player player, City city) throws TooPoorException {
/**
* Recruit troops to be stationed in a city
* Recruit 50 troops for 500 golds
* If the player does not have enough gold, throw an exception
* <p>
* Overridden in WarGeneral class
*
* @param player
* @param city
* @throws TooPoorException
*/
}
```

```
public void upgradeTech(Player player, Technology technology) throws
TooPoorException {
/**
* Upgrades tech
* 1. Get the cost of upgrading tech, with applied minister discount
* 2. Check whether player has enough gold, production, and science points
* 3. If not, throw an exception
* 4. Subtract the costs from the player's balance
* 5. Add level of technology by one
* <p>
* HINT:
* - the Cost class has a getDiscounterCost() method
* - the Minister class has a getTechDiscountRate() method
*
* @param player
* @param technology
* @throws TooPoorException
*/
}


public void spyOnNeighbors(City city, GameMap map) {
/**
* This method is called when you want to spy on your neighbors.
* <p>
* Print the information of a City's neighbors
* <p>
* HINT:
* - GameMap class has a getNeighboringCities() method
* - City class overrides the toString() method which returns the
* String representation of a city
*
* @param city
* @param map
*/
}
```

The TODOs in the Economist class (Economist.java)

```
public Economist(int intelligence, int experience, int leadership) {
/**
* Call the superclass' constructor
* @param intelligence
* @param experience
* @param leadership
*/
}


@Override
public double getImprovementDiscountRate() {
/**
* @return improvement discount rate equals to 1 - (intelligence + experience) /
1500
*/
}


@Override
public int collectTax(City city) {
/**
* @param city to collect gold from
* @return 1.5 times the amount collected by other types of minister
*/
}
```

```
@Override
public void improveCropYield(Player player, City city) throws
TooPoorException {
/**
* Economists get a bonus when doing crops improvements
* Crop improvement still costs 500 gold
* Crop is improved by 50 + round((intelligence + experience + leadership) * 0.2)
* If the player does not have enough gold, throw an exception
* Else decrease 500 golds from the player and improves crops by the calculated
amount
*
* @param player
* @param city
* @throws TooPoorException
*/
}

@Override
public String toString() {
/**
* Example string representation:
* "Economist | intelligence: 100 | experience: 100 | leadership: 100 | READY" -
when isReady() == true
* "Economist | intelligence: 100 | experience: 100 | leadership: 100 | DONE" -
when isReady() == false
*
* @return string representation of this object
*/
}
```

The TODOs in the Scientist class (Scientist.java)

```
public Scientist(int intelligence, int experience, int leadership) {
/**
* Calls the superclass' constructor
*
* @param intelligence
* @param experience
* @param leadership
*/
}


@Override
public double getTechDiscountRate() {
/**
* @return tech discount rate equals to 1 - (intelligence + experience) / 1500
*/
}


@Override
public int collectSciencePoints(City city) {
/**
* @param city to collect science points from
* @return 1.5 times the amount collected by other types of minister
*/
}
```

```
@Override
public String toString() {
/**
* Example string representation:
* "Scientist | intelligence: 100 | experience: 100 | leadership: 100 | READY" - when isReady() == true
* "Scientist | intelligence: 100 | experience: 100 | leadership: 100 | DONE" - when isReady() == false
*
* @return string representation of this object
*/
}
```

The TODOs in the WarGeneral class (WarGeneral.java)

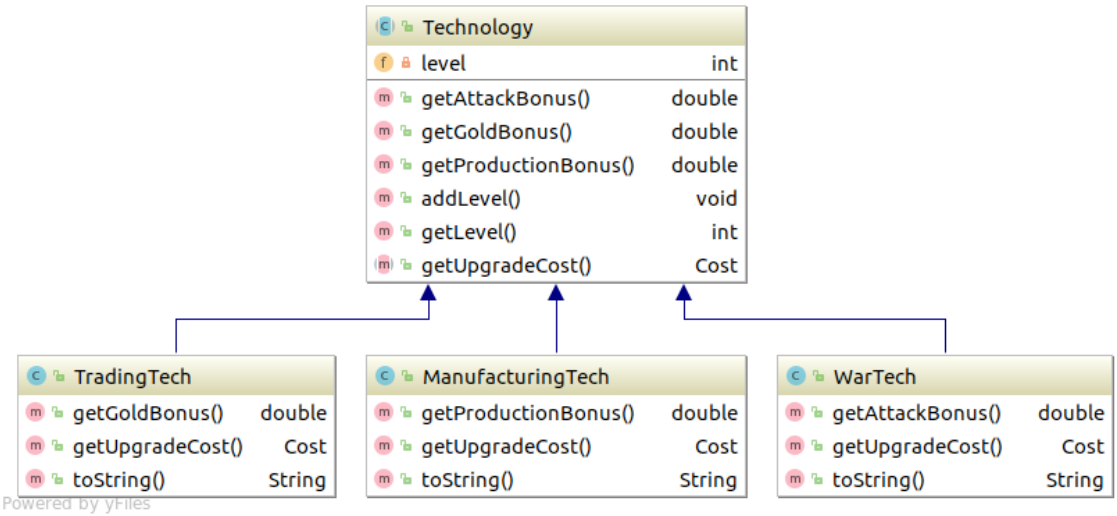
```
public WarGeneral(int intelligence, int experience, int leadership) {
/**
* Calls the superclass' constructor
*
* @param intelligence
* @param experience
* @param leadership
*/
}


@Override
public int collectProductionPoints(City city) {

/**
* @param city to collect production points from
* @return 1.5 times the amount collected by other types of minister
*/
}
```

```
@Override
public void recruitTroops(Player player, City city) throws TooPoorException {
/**
* Recruits more troops than superclass implementation,
* troops added to city = 50 + round(leadership * 0.2).
* Recruitment still costs 500 golds. Throw an exception
* when player does not have enough gold.
*
* @param player
* @param city
* @throws TooPoorException
*/
}

@Override
public void attackCity(City attacker, City defender, int troops, List<Technology>
technologyList) {
/**
* WarGeneral gets attack bonus when attacking a city.
* bonus_multiplier = 1 + (intelligence + experience + leadership) / 1500
* <p>
* Attacking city loses troops equal to min(# of troops attacking, # of troops in the
defending city)
* Defending city loses round(bonus_multiplier * # of attacking troops * product of
tech attack bonuses)
* <p>
* "[attacker city name] loses [number of troops lost by attacker] troops while
attacking"
* "[defender city name] loses [number of troops lost by defender] troops while
defending"
*
* @param attacker      attacking city
* @param defender      defending city
* @param troops        number of troops deployed for the attack
* @param technologyList
*/
}
```

```
@Override
public String toString() {
/**
* Example string representation:
* "WarGeneral | intelligence: 100 | experience: 100 | leadership: 100 | READY" -
when isReady() == true
* "WarGeneral | intelligence: 100 | experience: 100 | leadership: 100 | DONE" -
when isReady() == false
*
* @return string representation of this object
*/
}
```

The TODOs in the pa1.technologies *package* (in the technologies folder)

The technology package contains classes represent the technologies owned by a player and they provide bonuses to various aspects of the game. They are structured similarly to the minister classes with one abstract class and many specific subclasses with different behaviors. You need to complete the implementation of these subclasses.



The TODOs in the TradingTech class (TradingTech.java)

```
@Override
public double getGoldBonus() {
/**
* @return gold bonus equal to 1 + (level * 0.5);
*/
}
```

```
@Override
public Cost getUpgradeCost() {
/**
* Upgrade costs:
* gold = production = science = (current level + 1) * 1000;
*
* @return upgrade costs
*/
}
```

```
@Override
public String toString() {
/**
* Example string representation:
* "TradingTech | level: 1 | gold bonus: 1.50"
*
* @return String representing this object
*/
}
```

The TODOs in the ManufacturingTech class (ManufacturingTech.java)

```
@Override
public double getProductionBonus() {
/**
* @return production bonus equal to 1 + current level * 0.5
*/
}
```

```
@Override
public Cost getUpgradeCost() {
/**
* Upgrade costs:
* gold = production = (current level + 1) * 1000;
* science = 0
*
* @return upgrade costs
*/
}
```

```
@Override
public String toString() {
/**
* Example string representation:
* "ManufacturingTech | level: 1 | production bonus: 1.50"
*
* @return String representing this object
*/
}
```

The TODOs in the WarTech class (WarTech.java)

```
@Override
public double getAttackBonus() {
/**
* @return attack bonus equal to 1 + level * 0.5
*/
}
```

```
@Override
public Cost getUpgradeCost() {
/**
* Upgrade costs:
* gold = science = (current level + 1) * 1000;
* production = 0
*
* @return upgrade costs
*/
}
```

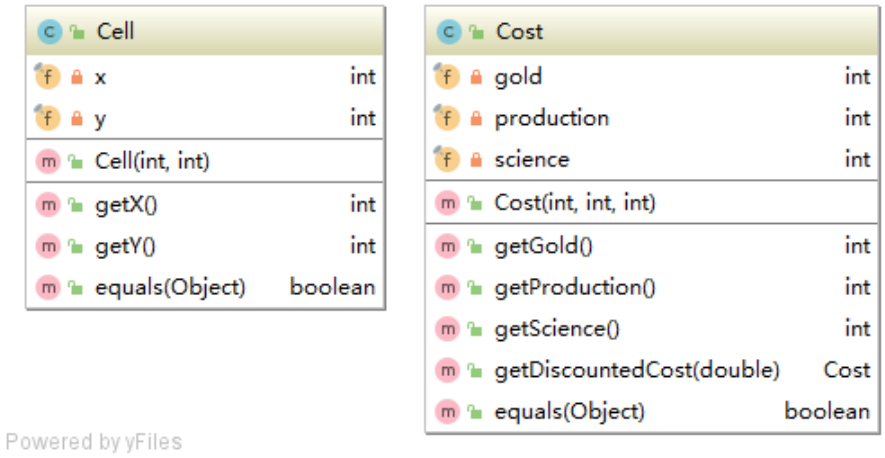
```
@Override
public String toString() {
/**
* Example string representation:
* "WarTech | level: 1 | attack bonus: 1.50"
*
* @return String representing this object
*/
}
```

Additional Classes

These additional classes help encapsulate data. The `Cell` class represent an x, y location on the game map. The `Cost` class encapsulates gold, production and science points costs for performing a task.

The TODOs in the Cell class (Cell.java)

Implement this class from scratch according to the UML diagram



The TODOs in the Cost class (Cost.java)

```
public Cost getDiscountedCost(double rate) {
/**
* Get a new Cost object with applied discount rate
*
* @param rate discount rate
* @return Discounted Cost = round(rate * current cost)
*/
}


@Override
public boolean equals(Object obj) {
/**
* Two Cost objects are equal if their
* gold costs AND production costs AND science costs
* are equal
* <p>
* Return false if obj is not an instance of Cost
*
* @param obj
* @return
*/
}
```

Exceptions

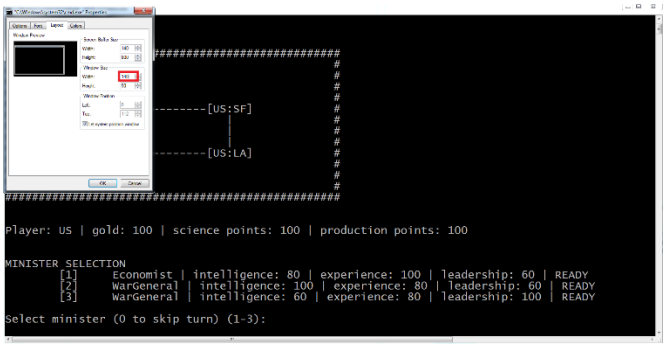
You also need to implement a custom exception class, `TooPoorException`. This exception will be thrown when the player does not have enough resources to complete a task. You will need to throw this exception in your implementation of the minister classes.

TooPoorException.java

```
@Override
public String getMessage() {
/**
* Constructs an error message in the form:
* "need (%d golds, %d production points, %d science points), have (%d golds,
%d production points, %d science points)"
*
* @return
*/
}
```

Running the executable file

An executable file for the PA could be downloaded from the PA1 link at the course web. You could use it to verify the correctness of your finished PA. To run the executable file under Windows (we only have Windows machine to compile it :( ), first you have to make sure the two .txt files (map.txt and players.txt files) are in the same directory as the executable file. Then Open a Windows command window, adjust the width of the command window as indicate below for a better display:



Then issue the command:

ASCII_civilization.exe

After you are done with the PA

Congratulations! Hope you have enjoyed it! Now you can Zip your complete project directory to PA1.zip and submit this file to the CASS link shown on page 1. You can submit for as many times as you wish before the deadline, but we will only mark you latest submission.

Late Submission Policy

There will be a penalty of -1 point (out of a maximum 100 points) for every minute you submit the PA late. If you submit your PA1 at 00:30am on 17 April 2019 (Wednesday), there will be a penalty of -35 points for your assignment. The lowest score you may get from the assignment is zero. If you submit your PA1 at 2:35am on 17 April 2019 or later, you will have zero for the PA1. So be sure to submit the finished PA1 as early as possible.