German University in Cairo
Media Engineering and Technology
Prof. Dr. Slim Abdennadher

Computer Programming Lab, *Spring 2021*
Empire Building: **Milestone 1**

Deadline: 07.05.2021 @ 23:59

This milestone is an *exercise* on the concepts of **Object Oriented Programming (OOP)**. The following sections describe the requirements of the milestone. Refer to the (**Game Setup**) section in the project description for more details about the game rules.

By the **end of this milestone**, you should have:

- A packaging hierarchy for your code
- An initial implementation for all the needed data structures
- Basic data loading capabilities from a CSV file

1 Build Project Hierarchy

1.1 Add the packages

Create a new `Java` project and build the following hierarchy of packages:

1. `buildings`
2. `units`
3. `engine`
4. `view`
5. `exceptions`
6. `tests`

Afterward, proceed by implementing the following classes. You are allowed to add more classes, attributes and methods. However, you must use the same names for the provided classes, attributes and methods.

1.2 Naming and privacy conventions

Please note that all your class attributes must be `private` and all methods should be `public` unless otherwise stated. You should implement the appropriate setters and getters conforming with the access constraints. Throughout the whole milestone, if a variable is said to be READ then we are allowed to get its value. If the variable is said to be WRITE then we are allowed to change its value. Please note that getters and setters should match the Java naming conventions. If the instance variable is of type boolean, the getter method name starts by **is** followed by the **exact** name of the instance variable. Otherwise, the method name starts with the verb (get or set) followed by the **exact** name of the instance variable; the first letter of the instance variable should be capitalized. Please note that the method names are case-sensitive.

Example 1 *You want a getter for an instance variable called* `milkCount` → *Method name =* `getMilkCount()`

BUILDINGS

2 Build the (Building) Class

Name : `Building`

Package : `buildings`

Type : Class

Description : A class representing a building. No objects of type `Building` can be instantiated.

2.0.1 Attributes

All the class attributes are READ and WRITE unless otherwise specified.

1. `int cost`: The cost for creating a building. This attribute is READ ONLY.
2. `int level`: The current level of the building. All buildings start from level 1.
3. `int upgradeCost`: The cost for upgrading building's level.
4. `boolean coolDown`: A variable stating if the building is cooling down. Initially, this variable should be set to `true`.

2.0.2 Constructors

1. `public Building(int cost,int upgradeCost)`: Constructor that initializes the attributes of a `Building` object.

3 Build the (EconomicBuilding) Class

Name : `EconomicBuilding`

Package : `buildings`

Type : Class

Description : A subclass of `Building` representing economical buildings. No objects of type `EconomicBuilding` can be instantiated.

3.0.1 Constructors

1. `public EconomicBuilding(int cost,int upgradeCost)`: Constructor that initializes the attributes of a `EconomicBuilding` object. It should use the constructor of the superclass.

3.1 Subclasses

There are two different types of economic buildings. Each building has its own cost and upgrade cost. Each economic building type should be represented as a separate subclass of `EconomicBuilding`. Each subclass should have its own constructor that utilizes the `EconomicBuilding` constructor. Carefully consider the design of each constructor.

1. `Farm`: A farm has 1000 `cost` and 500 `upgradeCost`.
2. `Market`: A market has 1500 `cost` and 700 `upgradeCost`.

4 Build the (MilitaryBuilding) Class

Name : `MilitaryBuilding`

Package : `buildings`

Type : Class

Description : A subclass of `Building` representing military buildings. No objects of type `MilitaryBuilding` can be instantiated.

4.0.1 Attributes

All the class attributes are READ and WRITE unless otherwise specified.

1. `int recruitmentCost`: The cost for recruiting a unit.
2. `int currentRecruit`: Current number of units recruited by a building inside a turn.
3. `int maxRecruit`: Maximum number of units a building can recruit per turn. Any building can recruit only 3 units per turn and **cannot be changed**. This attribute is READ ONLY.

4.0.2 Constructors

1. `public MilitaryBuilding(int cost, int upgradeCost,int recruitmentCost)`: Constructor that initializes the attributes of a `MilitaryBuilding` object. It should use the constructor of the superclass.

4.1 Subclasses

There are three different types of military buildings. Each building has its own cost and upgrade cost. Each military building type should be represented as a separate subclass of `MilitaryBuilding`. Each subclass should have its own constructor that utilizes the `MilitaryBuilding` constructor. Carefully consider the design of each constructor.

1. `ArcheryRange`: An archery range has 1500 `cost`, 800 `upgradeCost` and 400 `recruitmentCost`.
2. `Barracks`: A barrack has 2000 `cost`, 1000 `upgradeCost` and 500 `recruitmentCost` .
3. `Stable`: A stable has 2500 `cost`, 1500 `upgradeCost` and 600 `recruitmentCost` .

UNITS

5 Build the (Unit) Class

Name : `Unit`

Package : `units`

Type : Class

Description : A class representing a unit. No objects of type `Unit` can be instantiated.

5.0.1 Attributes

All the class attributes are READ and WRITE unless otherwise specified.

1. `int level`: The current level of a unit. This attribute is READ ONLY.
2. `int maxSoldierCount`: The maximum number of soldiers a unit can hold. This attribute is READ ONLY.

3. `int currentSoldierCount` : The current number of soldiers inside a unit.
4. `double idleUpkeep` : The amount of food a unit will consume when being idle. This attribute is READ ONLY.
5. `double marchingUpkeep` : The amount of food a unit will consume when marching to another city. This attribute is READ ONLY.
6. `double siegeUpkeep` : The amount of food a unit will consume when laying siege. This attribute is READ ONLY.

5.0.2 Constructors

1. `public Unit(int level,int maxSoldierCount,double idleUpkeep, double marchingUpkeep,double siegeUpkeep)`: Constructor that initializes the attributes of an `Unit` object.

6 Build the (Status) Enum

Name : `Status`

Package : `units`

Type : Enum

Description : An enum representing the status of the army. Possible values are: IDLE,MARCHING,BESIEGING.

7 Build the (Army) Class

Name : `Army`

Package : `units`

Type : Class

Description : A class representing the player army.

7.0.1 Attributes

All the class attributes are READ and WRITE unless otherwise specified.

1. `Status currentStatus`: The current status of an army. Initially, an army is IDLE.
2. `ArrayList<Unit> units`: An ArrayList containing the units of the army.
3. `int distancetoTarget`: The distance needed to reach the target city. Initially, the distance to target is -1.
4. `String target`: The target city. Initially the target location is `""`.
5. `String currentLocation`: The current location of the army. The army can be either in a city or on road to another one.
6. `int maxToHold`: The maximum number of units a unit can hold. This attribute should be set to 10 and **cannot be changed**. This attribute is READ ONLY.

7.0.2 Constructors

1. `public Army(String currentLocation)`: Constructor that initializes the attributes of a `Army` object.

8 Build the (Archer) Class

Name : `Archer`

Package : `unit`

Type : Class

Description : A subclass of `Unit` representing Archers.

8.0.1 Constructors

1. `public Archer(int level, int maxSoldierCount, double idleUpkeep, double marchingUpkeep, double siegeUpkeep)`: Constructor that initializes the attributes of an `Archer` object. It should use the constructor of the super class.

8.1 Values

These values shall be used later when loading CSV files. This will come later in this milestone

level	maxSoldier	idleKeep	marchkeep	siegeKeep
1	60	0.4	0.5	0.6
2	60	0.4	0.5	0.6
3	70	0.5	0.6	0.7

9 Build the (Infantry) Class

Name : `Infantry`

Package : `unit`

Type : Class

Description : A subclass of `Unit` representing Infantries.

9.0.1 Constructors

1. `public Infantry(int level, int maxSoldierCount, double idleUpkeep, double marchingUpkeep, double siegeUpkeep)`: Constructor that initializes the attributes of an `Infantry` object. It should use the constructor of the super class.

9.1 Values

These values shall be used later when loading CSV files. This will come later in this milestone

level	maxSoldier	idleKeep	marchkeep	siegeKeep
1	50	0.5	0.6	0.7
2	50	0.5	0.6	0.7
3	60	0.6	0.7	0.8

10 Build the (Cavalry) Class

Name : `Cavalry`

Package : `unit`

Type : Class

Description : A subclass of `Unit` representing Cavalries.

10.0.1 Constructors

1. `public Cavalry(int level, int maxSoldierCount, double idleUpkeep, double marchingUpkeep, double siegeUpkeep)`: Constructor that initializes the attributes of an `Cavalry` object. It should use the constructor of the super class.

10.1 Values

These values shall be used later when loading CSV files. This will come later in this milestone

level	maxSoldier	idleKeep	marchkeep	siegeKeep
1	40	0.6	0.7	0.75
2	40	0.6	0.7	0.75
3	60	0.7	0.8	0.9

ENGINE

11 Build the (City) Class

Name : `City`

Package : `engine`

Type : Class

Description : A class representing a city.

11.0.1 Attributes

All the class attributes are READ and WRITE unless otherwise specified.

1. `String name`: The name of the city. This attribute is READ ONLY.
2. `ArrayList<EconomicBuilding> economicalBuildings`: An ArrayList containing the economical buildings inside the city. This attribute is READ ONLY.
3. `ArrayList<MilitaryBuilding> militaryBuildings`: An ArrayList containing the military buildings inside the city. This attribute is READ ONLY.
4. `Army defendingArmy`: The defending army of the city. This attribute should be initialized by creating a new army.
5. `int turnsUnderSiege`: Number of turns the city has been sieged.
6. `boolean underSiege`: Variable checking if the city is under siege or not. Initially, this variable should be set to `false`.

11.0.2 Constructors

1. `public City(String name)`: Constructor that initializes the attributes of a `City` object.

12 Build the (Distance) Class

Name : `Distance`

Package : `engine`

Type : Class

Description : A class representing the distance between two cities.

12.0.1 Attributes

All the class attributes are READ and WRITE unless otherwise specified.

1. `String from`: The name of the city that the army will begin moving from. This attribute is READ ONLY.
2. `String to`: The name of the city that the army will move to. This attribute is READ ONLY.
3. `int distance`: The distance between the two cities. This attribute is READ ONLY.

12.0.2 Constructors

1. `public Distance(String from,String to, int distance)`: Constructor that initializes the attributes of a `Distance` object.

13 Build the (Player) Class

Name : `Player`

Package : `engine`

Type : Class

Description : A class representing the game player.

13.0.1 Attributes

All the class attributes are READ and WRITE unless otherwise specified.

1. `String name`: The name of the player. This attribute is READ ONLY.
2. `ArrayList<City> controlledCities`: An ArrayList containing the player's controlled cities. This attribute is READ ONLY
3. `ArrayList<Army> controlledArmies`: An ArrayList containing the player's controlled armies. This attribute is READ ONLY
4. `double treasury`: The amount of gold the player has.
5. `double food`: The amount of food the player has.

13.0.2 Constructors

1. `public Player(String name)`: Constructor that initializes the attributes of a `Player` object.

14 Build the (Game) Class

Name : `Game`

Package : `engine`

Type : Class

Description : A class representing the game.

14.0.1 Attributes

All the class attributes are READ and WRITE unless otherwise specified.

1. `Player player`: The current player of the game.
2. `ArrayList<City> availableCities`: An ArrayList containing the cities in the game. This attribute is READ ONLY
3. `ArrayList<Distance> distances`: An ArrayList containing the distances between the cities. This attribute is READ ONLY
4. `int maxTurnCount`: Maximum number of turns in the Game. This variable should be set to 30 and **cannot be changed**. This variable is READ ONLY
5. `int currentTurnCount`: Current number of turns. At the start of the game, the turn count should be 1.

14.0.2 Constructors

1. `public Game(String playerName,String playerCity) throws IOException`: Constructor that initializes the attributes of a `Game` object. Carefully think about how will you initialize the army of the defending cities.

14.1 Loading Army

The data of each city army will be available in a comma-separated values format file (CSV) named `"cityname".csv`. After the player chooses his city, the LoadArmy method should be called to initialize the defending army of the defending cities. For example, if the player chooses to play with Cairo, so LoadArmy method should be called on Rome and Sparta

14.2 CSV file format

The information of the Armies is available in a CSV file. You should add `throws IOException` to the header of any constructor or method that reads from a CSV file to compensate for any exceptions that could arise.
The armies are found in the file titled with the following format:

- Each line represents a unit.
- The data has no header, i.e. the first line represents the first minion.
- The parameters are separated by a comma (,).
- each line contains unit data as follows: TYPE,LEVEL.

14.3 Loading City and distances

The distance between each two cities will be available in a comma-separated values format file (CSV) named `distances.csv`. loadCitiesAndDistances method should be called to initialize the distances between each city.

14.4 CSV file format

The information of the distances will be available in a CSV file. You should add `throws IOException` to the header of any constructor or method that reads from a CSV file to compensate for any exceptions that could arise.
The distances are found in the file titled with the following format:

- Each line represents a unit.

- The data has no header, i.e. the first line represents the first minion.
- The parameters are separated by a comma (,).
- each line contains distances data as follows: CITY1NAME,CITY2NAME,DISTANCE.

14.5 Methods

1. `public void loadArmy(String cityName,String path) throws IOException`: This method is given a path of a CSV file containing the data of a specific city's army with the format mentioned above and is required to initialize the defending army of the given city. when creating a unit you need to refer to the given values for each level to know how to initialize it correctly.
2. `private void loadCitiesAndDistances() throws IOException`: This method should read the `distances.csv` file and update the `availableCities` and `distances` variables accordingly.

EXCEPTION CLASSES

You should only implement the exception classes to be later thrown and handled in milestones 2 and 3, respectively. Always be sure to make the exception messages as descriptive as possible, as these messages should be displayed whenever any exception is thrown.

15 Build the (EmpireException) Class

Name : `EmpireException`

Package : `exceptions`

Type : Class

Description : A subclass of `Exception`. Class representing a generic exception that can occur during the game play. These exceptions arise from any invalid action that is performed. No instances of this exception can be created.

15.0.1 Constructors

1. **EmpireException()**: Initializes an instance of a `EmpireException` by calling the constructor of the super class.
2. **EmpireException(String s)**: Initializes an instance of a `EmpireException` by calling the constructor of the super class with a customized message.

16 Build the (BuildingException) Class

Name : `BuildingException`

Package : `exceptions`

Type : Class

Description : A subclass of `EmpireException` representing an exception that occurs when trying to do invalid actions related to buildings. No instances of this exception can be created.

16.0.1 Constructors

1. **BuildingException()**: Initializes an instance of a `BuildingException` by calling the constructor of the super class.
2. **BuildingException(String s)**: Initializes an instance of a `BuildingException` by calling the constructor of the super class with a customized message.

17 Build the (ArmyException) Class

Name : `ArmyException`

Package : `exceptions`

Type : Class

Description : A subclass of `EmpireException` representing an exception that occurs when trying to do invalid actions related to armies. No instances of this exception can be created.

17.0.1 Constructors

1. **ArmyException()**: Initializes an instance of a `ArmyException` by calling the constructor of the super class.
2. **ArmyException(String s)**: Initializes an instance of a `ArmyException` by calling the constructor of the super class with a customized message.

18 Build the (BuildingInCoolDownException) Class

Name : `BuildingInCoolDownException`

Package : `exceptions`

Type : Class

Description : A subclass of `BuildingException` representing an exception that occurs when trying to do an action with a building while the building is cooling down.

18.0.1 Constructors

1. **BuildingInCoolDownException()**: Initializes an instance of a `BuildingInCoolDownException` by calling the constructor of the super class.
2. **BuildingInCoolDownException(String s)**: Initializes an instance of a `BuildingInCoolDownException` by calling the constructor of the super class with a customized message.

19 Build the (NotEnoughGoldException) Class

Name : `NotEnoughGoldException`

Package : `exceptions`

Type : Class

Description : A subclass of `BuildingException` representing an exception that occurs when trying to do an action with a building while there isn't enough gold(treasury) for this action.

19.0.1 Constructors

1. **NotEnoughGoldException()**: Initializes an instance of a `NotEnoughGoldException` by calling the constructor of the super class.
2. **NotEnoughGoldException(String s)**: Initializes an instance of a `NotEnoughGoldException` by calling the constructor of the super class with a customized message.

20 Build the (MaxRecruitedException) Class

Name : `MaxRecruitedException`

Package : `exceptions`

Type : Class

Description : A subclass of `BuildingException` representing an exception that occurs when trying to recruit a unit with a building while the building reaches the maximum number of unit per turn.

20.0.1 Constructors

1. **MaxRecruitedException()**: Initializes an instance of a `MaxRecruitedException` by calling the constructor of the super class.
2. **MaxRecruitedException(String s)**: Initializes an instance of a `MaxRecruitedException` by calling the constructor of the super class with a customized message.

21 Build the (MaxLevelException) Class

Name : `MaxLevelException`

Package : `exceptions`

Type : Class

Description : A subclass of `BuildingException` representing an exception that occurs when trying to upgrade a building which is in level 3.

21.0.1 Constructors

1. **MaxLevelException()**: Initializes an instance of a `MaxLevelException` by calling the constructor of the super class.
2. **MaxLevelException(String s)**: Initializes an instance of a `MaxLevelException` by calling the constructor of the super class with a customized message.

22 Build the (MaxCapacityException) Class

Name : `MaxCapacityException`

Package : `exceptions`

Type : Class

Description : A subclass of `ArmyException` representing an exception that occurs when trying to add more units in an army while the maximum capacity was reached .

22.0.1 Constructors

1. **MaxCapacityException()**: Initializes an instance of a `MaxCapacityException` by calling the constructor of the super class.
2. **MaxCapacityException(String s)**: Initializes an instance of a `MaxCapacityException` by calling the constructor of the super class with a customized message.

23 Build the (FriendlyFireException) Class

Name : `FriendlyFireException`

Package : `exceptions`

Type : Class

Description : A subclass of `ArmyException` representing an exception that occurs when trying to attack a friendly army.

23.0.1 Constructors

1. **FriendlyFireException()**: Initializes an instance of a `FriendlyFireException` by calling the constructor of the super class.
2. **FriendlyFireException(String s)**: Initializes an instance of a `FriendlyFireException` by calling the constructor of the super class with a customized message.

24 Build the (FriendlyCityException) Class

Name : `FriendlyCityException`

Package : `exceptions`

Type : Class

Description : A subclass of `ArmyException` representing an exception that occurs when trying to attack a friendly city.

24.0.1 Constructors

1. **FriendlyCityException()**: Initializes an instance of a `FriendlyCityException` by calling the constructor of the super class.
2. **FriendlyCityException(String s)**: Initializes an instance of a `FriendlyCityException` by calling the constructor of the super class with a customized message.

25 Build the (TargetNotReachedException) Class

Name : `TargetNotReachedException`

Package : `exceptions`

Type : Class

Description : A subclass of `ArmyException` representing an exception that occurs when trying to attack a with an army while it haven't reached the target city location. yet.

25.0.1 Constructors

1. **TargetNotReachedException()**: Initializes an instance of a `TargetNotReachedException` by calling the constructor of the super class.
2. **TargetNotReachedException(String s)**: Initializes an instance of a `TargetNotReachedException` by calling the constructor of the super class with a customized message.