German University in Cairo
Media Engineering and Technology
Prof. Dr. Slim Abdennadher
Dr. Nourhan Ehab
Dr. Ahmed Abdelfattah

CSEN 401 - Computer Programming Lab, *Spring 2023*
The Last of Us: Legacy
Milestone 1

Deadline: 24.03.2023 @ 23:59

This milestone is an *exercise* on the concepts of **Object Oriented Programming (OOP)**. The following sections describe the requirements of the milestone.

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 the Project Hierarchy

1.1 Add the packages

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

1. `engine`
2. `exceptions`
3. `model.characters`
4. `model.collectibles`
5. `model.world`
6. `tests`
7. `views`

Afterwards, 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 by 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()`

Example 2 *You want a setter for an instance variable called* `milkCount` → *Method name =* `setMilkCount()`

Example 3 *You want a getter for a boolean variable called* `availableStock` → *Method name =* `isAvailableStock()`

2 Build the (Collectible) Interface

Name : `Collectible`

Package : `model.collectibles`

Type : Interface

Description : Interface containing the methods available to all Collectible objects within the game map. All Vaccines and Supplies are Collectible. For this milestone, you will be leaving this interface empty.

3 Build the (Vaccine) Class

Name : `Vaccine`

Package : `model.collectibles`

Type : Class

Description : A class representing `Vaccine`s in the game.

3.1 Constructors

1. `Vaccine()`: Default constructor.

4 Build the (Supply) Class

Name : `Supply`

Package : `model.collectibles`

Type : Class

Description : A class representing `Supplies` in the game.

4.1 Constructors

1. `Supply()`: Default constructor.

5 Build the (Character) Class

Name : `Character`

Package : `model.characters`

Type : Abstract class.

Description : A class representing the `Character`s available in the game. No objects of type `Character` can be instantiated.

5.1 Attributes

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

1. `String name`: A variable representing the name of the character. This attribute is READ ONLY.
2. `Point location`: A point representing the x, y coordinates of the character's location, using the built-in class "Point".
3. `int maxHp`: The maximum health points belonging to this character. This is the upper bound of character's currentHP. This attribute is READ ONLY.
4. `int currentHp` : An integer representing the current health points for the character.
5. `int attackDmg` : This number represents the damage inflicted by the character on its target when attacking. This attribute is READ ONLY.
6. `Character target`: A variable representing the target character that will be affected by any possible action done by the character.

5.2 Constructors

1. `Character(String name, int maxHp, int attackDmg)`: Constructor that initializes a `Character` object with the given parameters as the attributes.

5.3 Subclasses

There are two different types of characters available in the game. Each type is modelled as a subclass of the `Character` class. Each Character type should be implemented in a separate class within the same package as the `Character` class. Each of the subclasses representing the different characters should have its own constructor that utilizes the `Character` constructor. Carefully consider the design of the constructor of each subclass.
The following list gives the different class names.

Class name
`Hero`
`Zombie`

6 Build the (Hero) Class

Name : `Hero`

Package : `model.characters`

Type : Class.

Description : A class representing `Hero`s in the game.This class is a subclass of the `Character` class. No objects of type `Hero` can be instantiated.

6.1 Attributes

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

1. `int actionsAvailable` : An int representing the number of the actions available for each hero in a turn.
2. `int maxActions` : An int representing the maximum number of actions a hero can make in a turn. This attribute is READ ONLY.
3. `boolean specialAction` : A boolean represents if the hero has used his special action.

4. `ArrayList<Vaccine> vaccineInventory` : A list representing the vaccines collected by each hero. This attribute is READ ONLY.
5. `ArrayList<Supply> supplyInventory` : A list representing the supplies collected by each hero.This attribute is READ ONLY.

6.2 Constructors

1. `Hero(String name, int maxHp, int attackDmg, int maxActions)`: Constructor that initializes a `Hero` object with the given parameters as the attributes. Initially the actionsAvailable starts with the maxActions.

6.3 Subclasses

There are three different types of heroes available in the game. Each type is modelled as a subclass of the `Hero` class. Each Hero type should be implemented in a separate class within the same package as the `Hero` class. Each of the subclasses representing the different heroes should have its own constructor that utilizes the `Hero` constructor. Carefully consider the design of the constructor of each subclass.

7 Build the (Zombie) Class

Name : `Zombie`

Package : `model.characters`

Type : Class

Description : A class representing Zombies that are in the game. This class is a subclass of the `Character` class.

7.1 Attributes

1. `static int ZOMBIES_COUNT`: An int representing the number of zombies created.

7.2 Constructors

1. `Zombie()`: Constructor that initializes the values of the `Zombie` object. Zombies all have 40 hp, 10 attack damage, and are named according to how many Zombie objects have been created, ex: “Zombie 1”, “Zombie 2”, etc..

8 Build the (Fighter) Class

Name : `Fighter`

Package : `model.characters`

Type : Class

Description : A class representing `Fighter`s in the game. This class is a subclass of the `Hero` class.

8.1 Constructors

1. `Fighter(String name, int maxHp, int attackDmg, int maxActions)`: Constructor that initializes the values of the `Fighter` object with the given parameters as the attributes.

9 Build the (Medic) Class

Name : `Medic`

Package : `model.characters`

Type : Class

Description : A class representing `Medic`s in the game. This class is a subclass of the `Hero` class.

9.1 Constructors

1. `Medic(String name, int maxHp, int attackDmg, int maxActions)`: Constructor that initializes the values of the `Medic` object with the given parameters as the attributes.

10 Build the (Explorer) Class

Name : `Explorer`

Package : `model.characters`

Type : Class

Description : A class representing `Explorer`s in the game. This class is a subclass of the `Hero` class.

10.1 Constructors

1. `Explorer(String name, int maxHp, int attackDmg, int maxActions)`: Constructor that initializes an `Explorer` object.

11 Build the (Direction) Enum

Direction : `Direction`

Package : `model.characters`

Type : Enum

Description : An enum representing the different possible directions for character to move in. Possible values are: UP, DOWN, LEFT, RIGHT.

12 Build the (Cell) Class

Name : `Cell`

Package : `model.world`

Type : Class.

Description : A class representing `Cell`s in the game.
No objects of type `Cell` can be instantiated.

12.1 Attributes

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

1. `boolean isVisible`: Boolean representing if the cell is visible or not.

12.2 Constructors

1. `Cell()`: Default constructor.

12.3 Subclasses

There are 3 different types of cells available in the game. Each cell type is modelled as a subclass of the `Cell` class. Each Cell type should be implemented in a separate class within the same package as the `Cell` class. Each of the subclasses representing the different cells should have its own constructor that utilizes the `Cell` constructor. Carefully consider the design of the constructor of each subclass.

The following list gives the different class names.

Class name	Extra Attributes	Attributes Access
`CharacterCell`	Character character, boolean isSafe:	READ AND WRITE
`CollectibleCell`	Collectible collectible:	READ ONLY
`TrapCell`	int trapDamage (Random either 10, 20, or 30)	READ ONLY

Game Setup

13 Build the (Game) Class

Name : `Game`

Package : `engine`

Type : Class

Description : A class representing the `Game` itself. This class will represent the main engine of the game, and will ensure all game rules are followed.

13.1 Attributes

All the class attributes are `public`.

1. `static ArrayList<Hero> availableHeroes`: An arraylist representing the available Heroes in the game.
2. `static ArrayList<Hero> heroes`: An arraylist representing the Heroes participating in the game.
3. `static ArrayList<Zombie> zombies`: An arraylist representing the 10 zombies generated in the game.
4. `static Cell [][] map`: A 2D array representing the map in the game.

13.2 Methods

1. `public static void loadHeroes(String filePath)`: Reads the CSV file with `filePath` and loads the Heroes into the `availableHeroes` ArrayList.

13.3 Description of CSV files format

You should add `throws Exception` to the header of any constructor or method that reads from a csv file to accommodate for any checked exception that could arise.

Heroes

1. The Heroes are found in a file titled `Heroes.csv`.
2. Each line represents the information of a single Hero.
3. The data has no header, i.e. the first line represents the first Hero.

4. The parameters are separated by a comma (,).
5. The line represents the Heroes' data as follows: **name, Type, maxHp, maxActions, attackDmg** .
6. The type represents the type of Hero:-
   - FIGH for Fighter
   - EXP for Explorer
   - MED for Medic

Exceptions

14 Build the (GameActionException) Class

Name : `GameActionException`

Package : `exceptions`

Type : Class

Description : Class representing a generic exception that can occur during the game play. These exceptions arise from any invalid action that is performed.
No objects of type `GameActionException` can be instantiated.
This class is a subclass of the java `Exception` class. This class has four subclasses; InvalidTargetException, MovementException, NoAvailableResourcesException, and NotEnoughActionsException.

14.1 Constructors

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

15 Build the (InvalidTargetException) Class

Name : `InvalidTargetException`

Package : `exceptions`

Type : Class

Description : A subclass of `GameActionException` representing an exception that occurs upon trying to target a wrong character with an action.

15.1 Constructors

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

16 Build the (MovementException) Class

Name : `MovementException`

Package : `exceptions`

Type : Class

Description : A subclass of `GameActionException` representing an exception that occurs when a character tries to make an invalid movement.

16.1 Constructors

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

17 Build the (NoAvailableResourcesException) Class

Name : `NoAvailableResourcesException`

Package : `exceptions`

Type : Class

Description : A subclass of `GameActionException` representing an exception that occurs when a character tries to use a Collectible he does not have.

17.1 Constructors

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

18 Build the (NotEnoughActionsException) Class

Name : `NotEnoughActionsException`

Package : `exceptions`

Type : Class

Description : A subclass of `GameActionException` representing an exception that occurs when a character tries take an action without the sufficient action points available.

18.1 Constructors

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