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

Computer Programming Lab, *Spring 2018*
Super Heroes Chess
Milestone 1

Deadline: 19.3.2018 @ 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. `model.game`
2. `model.pieces`
3. `model.pieces.heroes`
4. `model.pieces.sidekicks`
5. `controller`
6. `view`
7. `exceptions`
8. `tests`

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()`

CHESS BOARD SETUP

2 Build the (Player) Class

Name : `Player`

Package : `model.game`

Type : Class

Description : A class representing one `Player` of the chess game.

2.1 Attributes

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

1. `String name`: The `Player`'s name. This attribute is READ ONLY.
2. `int payloadPos`: The progress of the `Player`'s payload.
3. `int sideKilled`: The number of sidekicks that the `Player` killed.
4. `ArrayList<Piece> deadCharacters`: Contains the dead characters i.e. `Pieces` of the player. This attribute is READ ONLY.

2.2 Constructors

1. `Player(String name)`: Constructor that initializes a `Player` object with the given `name`.

3 Build the (Direction) Enum

Name : `Direction`

Package : `model.game`

Type : Enum

Description : An enum representing the different possible moving directions. Possible values are: RIGHT, LEFT, UP, DOWN, UPRIGHT, UPLEFT, DOWNRIGHT and DOWNLEFT.

4 Build the (Cell) Class

Name : `Cell`

Package : `model.game`

Type : Class

Description : A class representing a `Cell` within the chess board.

4.1 Attributes

All the class attributes are READ and WRITE.

1. `Piece piece`: The piece positioned in the `Cell`.

4.2 Constructors

1. `Cell()`: Constructor that initializes a `Cell` object with no pieces.
2. `Cell(Piece piece)`: Constructor that initializes a `Cell` object with the given `piece`.

5 Build the (Game) Class

Name : `Game`

Package : `model.game`

Type : Class

Description : A class representing the main game class where the whole chess game is monitored.

5.1 Attributes

All the class attributes are READ ONLY unless otherwise specified. All indices start from 0.

1. `int payloadPosTarget`: The required number of steps that a player's payload should move for them to win. This attribute is set only once to 6 at the beginning of the game. Its value does not change.
2. `int boardWidth`: The width of the chess board. This attribute is set only once to 6 at the beginning of the game. Its value does not change.
3. `int boardHeight`: The height of the chess board. This attribute is set only once to 7 at the beginning of the game. Its value does not change.
4. `Player player1`: The first player.
5. `Player player2`: The second player.
6. `Player currentPlayer`: The player whose turn it currently is. This attribute is READ and WRITE.
7. `Cell[][] board`: The 2D array consisting of `Cell`s that represent the board. The first index represents the height of the board and the second represents its width. This attribute is neither READ nor WRITE.

5.2 Constructors

1. `Game(Player player1, Player player2)`: Constructor that initializes a new game and its board of cells with the `boardHeight` and `boardWidth`. It sets the current player to the first player.

CHESS PIECES

6 Build the (Movable) Interface

Name : `Movable`

Package : `model.pieces`

Type : Interface

Description : Interface containing the methods available to all movable pieces within the chess board.

7 Build the (Piece) Class

Name : `Piece`

Package : `model.pieces`

Type : Class

Description : A class representing a piece within a cell in the chess board. No objects of type `Piece` can be instantiated. All `Pieces` are `Movable` objects.

7.1 Attributes

All the class attributes are READ ONLY unless otherwise specified.

1. `String name`: The name of the piece.
2. `Player owner`: The `Player` to whom the piece belongs.
3. `Game game`: The `Game` to which the piece belongs.
4. `int posI`: The "i" position of the `Piece` in the 2D `Cell` array representing the board. This attribute is READ and WRITE.
5. `int posJ`: The "j" position of the `Piece` in the 2D `Cell` array representing the board. This attribute is READ and WRITE.

7.2 Constructors

1. `public Piece(Player player, Game game, String name)`: Constructor that initializes the attributes of a `Piece` object.

8 Build the (Hero) Class

Name : `Hero`

Package : `model.pieces.heroes`

Type : Class

Description : A subclass of `Piece` representing a hero piece. No objects of type `Hero` can be instantiated.

8.1 Constructors

1. `public Hero(Player player, Game game, String name)`: Constructor that initializes the attributes of a `Hero` object. It should use the constructor of the superclass.

8.2 Subclasses

There are 6 different types of hero pieces available in the game. Each hero has their own superpower and allowed movement directions as detailed in the **Game Description Document** (to be handled in the next milestone). The heroes can be divided into two types: `ActivatablePowerHero` and `NonActivatablePowerHero` which should each be represented as a separate subclass of `Hero`. No instances of type `ActivatablePowerHero` and `NonActivatablePowerHero` can be instantiated. Each subclass should have its own constructor that utilizes the `Hero` constructor. Carefully consider the design of each constructor.

1. `ActivatablePowerHero`:
   (a) `Attributes:`
   `boolean powerUsed`: Indicates whether the hero's super power is used. The attribute is READ and WRITE.
   (b) `Subclasses:`
   Each activatable-power hero type should be implemented in a separate class within the same package as the `Hero` class. Each subclass should have its own constructor that utilizes the `ActivatablePowerHero` constructor. Carefully consider the design of each constructor. For more details about the different hero types please refer to the **Game Description Document**. There are four `ActivatablePowerHero` subclasses:
   - `Medic`
   - `Ranged`
   - `Super`
   - `Tech`
2. `NonActivatablePowerHero`:
   (a) `Subclasses:`
   Each non-activatable-power hero type should be implemented in a separate class within the same package as the `Hero` class. Each subclass should have its own constructor that utilizes the `NonActivatablePowerHero` constructor. Carefully consider the design of each constructor. For more details about the different hero types please refer to the **Game Description Document**. There are two `NonActivatablePowerHero` subclasses:
   - `Armored`: The class attribute `boolean armorUp` is READ and WRITE. The `armorUp` is true if the hero's super power is currently activated thus the armor is raised. It is initially set to true.
   - `Speedster`

9 Build the (SideKick) Class

Name : `SideKick`

Package : `model.pieces.sidekicks`

Type : Class

Description : A subclass of `Piece` representing a sidekick piece. No objects of type `SideKick` can be instantiated.

9.1 Constructors

1. `public SideKick(Player player, Game game, String name)`: Constructor that initializes the attributes of a `Sidekick` object. It should use the constructor of the superclass.

9.2 Subclasses

We distinguish between the sidekicks of the first and second player as two types of `Sidekick`s. Each player's sidekick is modelled as a subclass of the `SideKick` class. Each sidekick type should be implemented in a separate class within the same package as the `SideKick` class. Each subclass should have its own constructor that utilizes the `SideKick` constructor. Carefully consider the design of each constructor.

1. `SideKickP1`: A sidekick belonging to the first player.
2. `SideKickP2`: A sidekick belonging to the second player.

EXCEPTION CLASSES

You should only implement the exception classes to be later thrown and handled in milestone 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.

10 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 instances of this exception can be created.

10.1 Attributes

All class attributes are READ only.

1. `Piece trigger`: The `piece` that triggered the exception.

10.2 Constructors

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

11 Build the (InvalidPowerUseException) Class

Name : `InvalidPowerUseException`

Package : `exceptions`

Type : Class

Description : A subclass of `GameActionException` representing an exception that occurs upon trying to use a power in an invalid manner. No instances of this exception can be created.

11.1 Constructors

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

12 Build the (InvalidPowerDirectionException) Class

Name : `InvalidPowerDirectionException`

Package : `exceptions`

Type : Class

Description : A subclass of `InvalidPowerUseException` representing an exception that occurs upon trying to use a power in an invalid direction.

12.1 Attributes

All class attributes are READ only.

1. `Direction direction`: The invalid `direction` where the power was activated.

12.2 Constructors

1. **InvalidPowerDirectionException(Piece trigger, Direction d)**: Initializes an instance of an `InvalidPowerDirectionException` by calling the constructor of the super class.
2. **InvalidPowerDirectionException(String s, Piece trigger, Direction d)**: Initializes an instance of an `InvalidPowerDirectionException` by calling the constructor of the super class with a customized message.

13 Build the (InvalidPowerTargetException) Class

Name : `InvalidPowerTargetException`

Package : `exceptions`

Type : Class

Description : A subclass of `InvalidPowerUseException` representing an exception that occurs upon trying to use a power on an invalid target.

13.1 Attributes

All class attributes are READ only.

1. `Piece target`: The invalid `target` piece on which the power should have been applied.

13.2 Constructors

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

14 Build the (PowerAlreadyUsedException) Class

Name : `PowerAlreadyUsedException`

Package : `exceptions`

Type : Class

Description : A subclass of `InvalidPowerUseException` representing an exception that occurs upon trying to use a power that has already been used.

14.1 Constructors

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

15 Build the (InvalidMovementException) Class

Name : `InvalidMovementException`

Package : `exceptions`

Type : Class

Description : A subclass of `GameActionException` representing an exception that occurs upon trying to perform an invalid movement. No objects of type `InvalidMovementException` can be instantiated.

15.1 Attributes

All class attributes are READ only.

1. `Direction direction`: The invalid `direction` where the power was activated.

15.2 Constructors

1. **InvalidMovementException(Piece trigger, Direction direction)**: Initializes an instance of an `InvalidMovementException` by calling the constructor of the super class.
2. **InvalidMovementException(String s, Piece trigger, Direction direction)**: Initializes an instance of an `InvalidMovementException` by calling the constructor of the super class with a customized message.

16 Build the (UnallowedMovementException) Class

Name : `UnallowedMovementException`

Package : `exceptions`

Type : Class

Description : A subclass of `InvalidMovementException` representing an exception that occurs upon trying to perform a movement that is not allowed. The allowed movements can be found in the **Game Description Document**.

16.1 Constructors

1. **UnallowedMovementException(Piece trigger, Direction direction)**: Initializes an instance of an `UnallowedMovementException` by calling the constructor of the super class.
2. **UnallowedMovementException(String s, Piece trigger, Direction direction)**: Initializes an instance of an `UnallowedMovementException` by calling the constructor of the super class with a customized message.

17 Build the (OccupiedCellException) Class

Name : `OccupiedCellException`

Package : `exceptions`

Type : Class

Description : A subclass of `InvalidMovementException` representing an exception that occurs upon trying to perform a movement towards an already occupied cell.

17.1 Constructors

1. **OccupiedCellException(Piece trigger, Direction direction)**: Initializes an instance of an `OccupiedCellException` by calling the constructor of the super class.
2. **OccupiedCellException(String s, Piece trigger, Direction direction)**: Initializes an instance of an `OccupiedCellException` by calling the constructor of the super class with a customized message.

18 Build the (WrongTurnException) Class

Name : `WrongTurnException`

Package : `exceptions`

Type : Class

Description : A subclass of `GameActionException` representing an exception that occurs upon trying to perform an action in the wrong turn.

18.1 Constructors

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