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

Computer Programming Lab, *Spring 2020*
Hearthstone
Milestone 1

Deadline: 6.3.2020 @ 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.cards`
2. `model.cards.minions`
3. `model.cards.spells`
4. `model.heroes`
5. `engine`
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()`

CARDS

2 Build the (Card) Class

Name : `Card`

Package : `model.cards`

Type : Class

Description : A class representing a game card. No objects of type `Card` can be instantiated.

2.1 Attributes

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

1. `String name`: The name of the card.
2. `int manaCost`: The number of mana crystals needed to play the card. This value cannot be less than 0 or greater than 10.
3. `Rarity rarity`: The `rarity` of the current card. This attribute is READ ONLY.

2.2 Constructors

1. `public Card(String name,int manaCost,Rarity rarity)`: Constructor that initializes the attributes of a `Card` object.

3 Build the (Rarity) Enum

Name : `Rarity`

Package : `model.cards`

Type : Enum

Description : An enum representing the different rarity values of any given card. Possible values are: BASIC,COMMON,RARE,EPIC and LEGENDARY.

Minions

4 Build the (Minion) Class

Name : `Minion`

Package : `model.cards.minions`

Type : Class

Description : A subclass of `Card` representing a minion card.

4.1 Attributes

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

1. `int attack`: The attack value of the minion. This values can never fall below 0.
2. `int maxHP`: The maximum health points that the minion can have.
3. `int currentHP`: The current health points of the minion. It starts with the value of the `maxHP` and cannot exceed it.
4. `boolean taunt`: Indicates whether a minion is taunt.
5. `boolean divine`: Indicates whether a minion has a divine shield.
6. `boolean sleeping`: Indicates whether a minion can attack this turn. Sleeping minions cannot attack. Minions start as sleeping when they are first played, with the exception of charge minions. Starting the following turn, they awake and will remain as is.
7. `boolean attacked`: Indicates whether this minion has already attacked this turn.

4.2 Constructors

1. `public Minion(String name,int manaCost,Rarity rarity, int attack,int maxHP,boolean taunt,boolean divine,boolean charge)`: Constructor that initializes the attributes of a `Minion` object. It should use the constructor of the superclass.

4.3 Subclasses

A special type of minion is the Icehowl. It has `manaCost` 9, `rarity` LEGENDARY, `attack`, and `maxHP` 10. It is a charge minion. This minion can only attack other minions and can not attack heroes.

SPELLS

5 Build the (AOESpell) Interface

Name : `AOESpell`

Package : `model.cards.spells`

Type : Interface

Description : Interface containing the methods available to all area of effect spells.

6 Build the (FieldSpell) Interface

Name : `FieldSpell`

Package : `model.cards.spells`

Type : Interface

Description : Interface containing the methods available to all spells that affect the field of the hero casting them.

7 Build the (HeroTargetSpell) Interface

Name : `HeroTargetSpell`

Package : `model.cards.spells`

Type : Interface

Description : Interface containing the methods available to all spells that affect heroes.

8 Build the (LeechingSpell) Interface

Name : `LeechingSpell`

Package : `model.cards.spells`

Type : Interface

Description : Interface containing the methods available to all leeching spells, i.e. that damage the opponent's while restoring the hero's own health.

9 Build the (MinionTargetSpell) Interface

Name : `MinionTargetSpell`

Package : `model.cards.spells`

Type : Interface

Description : Interface containing the methods available to all spells that affect minions.

10 Build the (Spell) Class

Name : `Spell`

Package : `model.cards.spells`

Type : Class

Description : A subclass of `Card` representing a spell card. No objects of type `Spell` can be instantiated.

10.1 Constructors

1. `public Spell(String name, int manaCost ,Rarity rarity)`: Constructor that initializes the attributes of a `Spell` object. It should use the constructor of the superclass.

10.2 Subclasses

There are 13 different types of spells available in the game. Each spell belongs to a specific type of hero and has its own type and functionality (to be handled in the next milestone). Each spell should be represented as a separate subclass of `Spell`.Each subclass should have its own constructor that utilizes the `Spell` constructor. Carefully consider the design of each constructor.

1. `CurseOfWeakness`: An AOE spell with the `name` "Curse of Weakness", `manaCost` 2 and `rarity` RARE.
2. `DivineSpirit`: A minion target spell with the `name` "Divine Spirit", `manaCost` 3 and `rarity` BASIC.
3. `Flamestrike`: An AOE spell with the `name` "Flamestrike", `manaCost` 7 and `rarity` BASIC.
4. `HolyNova`: An AOE spell with the `name` "Holy Nova", `manaCost` 5 and `rarity` BASIC.
5. `KillCommand`: A minion and hero target spell with the `name` "Kill Command", `manaCost` 3 and `rarity` COMMON.
6. `LevelUp`: A field spell with the `name` "Level Up!", `manaCost` 6 and `rarity` EPIC.
7. `MultiShot`: An AOE spell with the `name` "Multi-Shot", `manaCost` 4 and `rarity` BASIC.
8. `Polymorph`: A minion target spell with the `name` "Polymorph", `manaCost` 4 and `rarity` BASIC.
9. `Pyroblast`: A hero and minion target spell with the `name` "Pyroblast", `manaCost` 10 and `rarity` EPIC.

10. `SealOfChampions`: A minion target spell with the `name` "Seal of Champions", `manaCost` 3 and `rarity` COMMON.
11. `ShadowWordDeath`: A minion target spell with the `name` "Shadow Word: Death", `manaCost` 3 and `rarity` BASIC.
12. `SiphonSoul`: A leeching spell with the `name` "Siphon Soul", `manaCost` 6 and `rarity` RARE.
13. `TwistingNether`: An AOE spell with the `name` "Twisting Nether", `manaCost` 8 and `rarity` EPIC.

HEROES

11 Build the (Hero) Class

Name : `Hero`

Package : `model.heroes`

Type : Class

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

11.1 Attributes

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

1. `String name`: The name of the hero. This attribute is READ ONLY.
2. `int currentHP`: The current health points of the hero. All heroes initially have 30 health points and can never exceed it.
3. `boolean heroPowerUsed`: Indicates whether the hero has used their power during the current turn.
4. `int totalManaCrystals`: The number of mana crystals the hero starts the turn with. This value cannot exceed 10.
5. `int currentManaCrystals`: The number of mana crystals that are currently available for the hero to use during the current turn. Each turn, it starts with the value of the `totalManaCrystals`. This value cannot exceed 10.
6. `ArrayList<Card> deck`: The cards of the hero's deck. This attribute is READ ONLY.
7. `ArrayList<Minion> field`: The minion cards available on the hero's field. This attribute is READ ONLY.
8. `ArrayList<Card> hand`: The cards in the hero's hand. This attribute is READ ONLY.
9. `int fatigueDamage`: The damage a hero receives when trying to draw a card from an empty deck. This attribute is NEITHER READ NOR WRITE.

11.2 Constructors

1. `public Hero(String name)`: Constructor that initializes the attributes of a `Hero` object and build the hero's deck (will be discussed later in this document).

11.3 Subclasses

There are 5 different types of heroes available in the game. Each hero has their own special ability and spells, (to be handled in the next milestone). Each hero type should be represented as a separate subclass of `Hero`. Each subclass should have its own constructor that utilizes the `Hero` constructor. Carefully consider the design of each constructor.

1. `Hunter`: A hero named "Rexxar".
2. `Mage`: A hero named "Jaina Proudmoore".
3. `Paladin`: A hero named "Uther Lightbringer".
4. `Priest`: A hero named "Anduin Wrynn".
5. `Warlock`: A hero named "Gul'dan".

11.4 Deck Building

A hero's deck consists of 20 cards (a mix of minions and spells). The minions are either hero-specific or neutral that can be owned by any hero. Each hero has a specific number of neutral minions in his deck, alongside a special hero-specific legendary minion and a specific number of spells. A hero can have up to two **copies** of the same minion in his deck, with the exception of the legendary ones (only one copy per legendary minion). For spells, they are all hero-specific and are thus assigned according to the hero type. The number of spells that will be in the hero's deck depends on the hero type.

The data of all available neutral minions that any hero can own will be available in a comma-separated values format file (CSV) named `neutral_minions.csv`. Each hero should **randomly** select his specific number of neutral minions from the minions their data are provided in that file abiding to the rules mentioned above.

11.5 CSV file format

The information of the neutral minions available to all heroes 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 minions are found in the file titled with the following format:

- Each line represents a minion.
- The data has no header, i.e. the first line represents the first minion.
- The parameters are separated by a comma (,).
- Each line contains a minion's data as follows: NAME, MANA_COST,RARITY, ATTACK, MAX_HP, TAUNT, DIVINE, CHARGE.
- Taunt, divine and charge data will be represented with TRUE or FALSE values.
  - RARITY: It can be one of the following values:
    1. b representing BASIC
    2. c representing COMMON
    3. r representing RARE
    4. e representing EPIC
    5. l representing LEGENDARY

11.6 Methods

1. `static ArrayList<Minion> getAllNeutralMinions(String filePath) throws IOException`: This method is given a path of a CSV file containing the data of neutral minions with the format mentioned above and is required to return an array list of Minion objects containing all the minions that their data are represented in the CSV file. This method should not be overridden by any subclass.
2. `static ArrayList<Minion> getNeutralMinions(ArrayList<Minion> minions,int count)`: Randomly selects the `count` minions one by one from the `minions` ArrayList and then returns them in an array list. This method should not be overridden by any subclass.
3. `void buildDeck() throws IOException`: Builds and **shuffles** the deck of the hero. This method, behaves differently based on the hero type. It should be separately implemented for each hero, as follows:
   (a) `Hunter`: The deck should contain:
      - 15 neutral minions
      - 2 KillCommand spells
      - 2 MultiShot spells
      - One charge minion with `name` "King Krush", `manaCost` 9 and `rarity` LEGENDARY, 8 `attack`, and 8 `maxHP`.

   (b) `Mage`: The deck should contain:
      - 13 neutral minions
      - 2 Polymorph spells
      - 2 FlameStrike spells
      - 2 PyroBlast spells
      - One minion with `name` "Kalycgos", `manaCost` 10 and `rarity` LEGENDARY, 4 `attack`, and 12 `maxHP`.

   (c) `Paladin`: The deck should contain:
      - 15 neutral minions
      - 2 SealOfChampions spells
      - 2 LevelUp spells
      - One taunt and divine shield minion with `name` "Tirion Fordring", `manaCost` 4 and `rarity` LEGENDARY, 6 `attack`, and 6 `maxHP`.

   (d) `Priest`: The deck should contain:
      - 13 neutral minions
      - 2 DivineSpirit spells
      - 2 HolyNova spells
      - 2 ShadowWordDeath spells
      - One minion with `name` "Prophet Velen", `manaCost` 7 and `rarity` LEGENDARY, 7 `attack`, and 7 `maxHP`.

   (e) `Warlock`: The deck should contain:
      - 13 neutral minions
      - 2 CurseOfWeakness spells
      - 2 SiphonSoul spells
      - 2 TwistingNether spells
      - One minion with `name` "Wilfred Fizzlebang", `manaCost` 6 and `rarity` LEGENDARY, 4 `attack`, and 4 `maxHP`.

GAME

12 Build the (Game) Class

Name : `Game`

Package : `engine`

Type : Class

Description : A class representing the main game class where the whole card game is initialized and monitored.

12.1 Attributes

All the class attributes are NEITHER READ NOR WRITE.

1. `Hero firstHero`: The first hero of the game.
2. `Hero secondHero`: The second hero of the game.
3. `Hero currentHero`: The hero whose turn it currently is. This attribute is READ ONLY.
4. `Hero opponent`: The current opposing hero (whose turn it isn't). This attribute is READ ONLY.

12.2 Constructors

1. `public Game(Hero p1, Hero p2)`: Constructor that initializes a new game by initializing the two game heroes with p1 and p2 respectively. It also randomly chooses the hero who will start the game. Initially, the starting hero will have only one mana crystal.

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.

13 Build the (HearthstoneException) Class

Name : `HearthstoneException`

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.

13.1 Constructors

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

14 Build the (FullFieldException) Class

Name : `FullFieldException`

Package : `exceptions`

Type : Class

Description : A subclass of `HearthstoneException` representing an exception that occurs when trying to add a minion card to an already full field.

14.1 Constructors

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

15 Build the (FullHandException) Class

Name : `FullHandException`

Package : `exceptions`

Type : Class

Description : A subclass of `HearthstoneException` representing an exception that occurs when trying to add a card to an already full hand.

15.1 Attributes

- `Card burned`: The card that will be burned (discarded) as the player's hand is already full.

15.2 Constructors

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

16 Build the (CannotAttackException) Class

Name : `CannotAttackException`

Package : `exceptions`

Type : Class

Description : A subclass of `HearthstoneException` representing an exception that occurs when the current minion cannot be used to attack. This can occur if the minion is sleeping or has already been used to attack this turn.

16.1 Constructors

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

17 Build the (HeroPowerAlreadyUsedException) Class

Name : `HeroPowerAlreadyUsedException`

Package : `exceptions`

Type : Class

Description : A subclass of `HearthstoneException` representing an exception that occurs when a hero attempts to use their hero power more than once per turn.

17.1 Constructors

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

18 Build the (InvalidTargetException) Class

Name : `InvalidTargetException`

Package : `exceptions`

Type : Class

Description : A subclass of `HearthstoneException` representing an exception that occurs when trying to do an action with a wrong target (Hint: think about Icehowl).

18.1 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.

19 Build the (NotEnoughManaException) Class

Name : `NotEnoughManaException`

Package : `exceptions`

Type : Class

Description : A subclass of `HearthstoneException` representing an exception that occurs when a hero attempts to perform an action with insufficient mana, e.g. playing a card or using hero power.

19.1 Constructors

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

20 Build the (NotSummonedException) Class

Name : `NotSummonedException`

Package : `exceptions`

Type : Class

Description : A subclass of `HearthstoneException` representing an exception that occurs when trying to attack with a minion that is not yet summoned to the field (i.e. still in hand).

20.1 Constructors

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

21 Build the (NotYourTurnException) Class

Name : `NotYourTurnException`

Package : `exceptions`

Type : Class

Description : A subclass of `HearthstoneException` representing an exception that occurs when a hero attempts to perform an action outside their turn.

21.1 Constructors

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

22 Build the (TauntBypassException) Class

Name : `TauntBypassException`

Package : `exceptions`

Type : Class

Description : A subclass of `HearthstoneException` representing an exception that occurs when a minion is trying to attack a target while the opponent has a taunt minion(s) in their field.

22.1 Constructors

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