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

Computer Programming Lab, *Spring 2015*
Yu Gi Oh: **Milestone 1**

Deadline: Fri 6.3.2015 @23:59

This milestone is an *exercise* on the concepts of **inheritance** and **encapsulation**. The following sections describe the requirements of the milestone. Refer to the (**Game Setup**) section in the project description for more details about the terminology.

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

- A packaging hierarchy for your code
- An initial implementation for all the needed data structures
- An implementation for the actions of the spell cards activated in the Main phases.

1 Build the Project Hierarchy

1.1 Add the packages

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

1. `eg.edu.guc.yugioh.cards`
2. `eg.edu.guc.yugioh.cards.spells`
3. `eg.edu.guc.yugioh.board`
4. `eg.edu.guc.yugioh.board.player`
5. `eg.edu.guc.yugioh.exceptions`
6. `eg.edu.guc.yugioh.gui`
7. `eg.edu.guc.yugioh.listeners`
8. `eg.edu.guc.yugioh.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 attributes and methods.

1.2 Naming and privacy conventions

Please note that all your class attributes must be **private**. You should implement the appropriate setters and getters conforming with the access constraints. Please note that getters and setters should match the Java naming conventions. The method name starts by the verb 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`

2 Build the (Card) Class

Name : `Card`

Package : `eg.edu.guc.yugioh.cards`

Description : A class representing a single card

2.1 Class attributes

1. `String name`: Represents the name of the card (E.g.: Gelo Monster)
2. `String description`: The description that should be displayed for the card.
3. `boolean isHidden`: If true, the monster is hidden from the opponent, otherwise it is visible.
4. `Location location`: Value from Enumeration (`Location`). It indicates the current location of the card. Possible values are: `DECK`, `HAND`, `FIELD` and `GRAVEYARD`. Cards are initially in the deck.
5. `Board board`: Instance of the board used for the current game. It should be the same instance for all cards.

- The name and the description should be initialized once in the constructor. Afterwards, they should be READ-ONLY.
- The rest of the attributes are READ and WRITE attributes.

2.2 Class methods

1. `public Card(String name, String description)`: Constructor to initialize instances of the class. The card is invisible by default.
2. `public void action(MonsterCard monster)`: Represents the sequence of actions possible by this card. This method should be overridden by each subclass of this class to customize the actions of the cards. If the required action does not need a monster, pass the parameter as `null`.

3 Build the (MonsterCard) Class

Name : `MonsterCard`

Package : `eg.edu.guc.yugioh.cards`

Description : Subclass of (`Card`) class representing the monsters in the game.

3.1 Class attributes

All the class attributes are READ and WRITE.

1. `int level`: Value indicating the needed actions to add a monster to the board. It is a numerical value between 1 - 8 inclusive. It is a READ-ONLY attribute.
2. `int defensePoints`: Numeric value representing the defense power of this monster. The higher the number, the stronger the monster.

3. `int attackPoints`: Numeric value representing the attacking power of this monster. The higher the number, the stronger the monster.
4. `Mode mode`: Value from Enumeration (`Mode`). Possible values are: **ATTACK** and **DEFENSE**. All cards are initially in **DEFENSE** position.

3.2 Class methods

1. `public MonsterCard(String name, String description, int level, int attack, int defence)`: Constructor to initialize the level, defense and attack points of a new monster. Your constructor must utilize the constructor of the (`Card`) class.

4 Model the Spells

All spells should be added to `eg.edu.guc.yugioh.cards.spells` package.

1. Build a class called `SpellCard` to represent all spell cards. The class is a subclass of the `Card` class.
2. `public SpellCard(String name, String description)`: Constructor to initialize the name and description of the spell card. Your constructor must utilize the constructor of the (`Card`) class.
3. Implement all the spells in the game. Each spell is modelled by a subclass from the (`SpellCard`) class or any of the other spells. Each spell will be modelled in a separate class. You should implement the actions of the spells in this milestone.

   Example 2 *Spell (A) doubles the attacking power of the monster. Spell (B) double the attacking power of the monster and the defense power of the monster. Therefore, B inherits from A*
4. There are 10 spells in the game. You can load them from the CSV file. The names and the actions of the spells are:
   (a) **Card Destruction**: Each player discards their entire hand, then draws the same number of cards they discarded. (**Class name:** `CardDestruction`)
   (b) **Change Of Heart**: You choose one monster from your opponent and add it to your own monsters. (**Class name:** `ChangeOfHeart`)
   (c) **Dark Hole**:Destroy all monsters on the field. (**Class name:** `DarkHole`)
   (d) **Graceful Dice**: Generates a random number between 1 and 10. All monsters you currently control gain ATK and DEF equal to the random number x 100. (**Class name:** `GracefulDice`)
   (e) **Harpie's Feather Duster**: Destroy all spell cards your opponent controls. (**Class name:** `HarpieFeatherDuster`)
   (f) **Heavy Storm**: Destroy all spell cards on the board for both players. (**Class name:** `HeavyStorm`)
   (g) **Mage Power**: Choose one monster from the field. This monster gains 500 ATK and DEF for each Spell Card you have on the field. (**Class name:** `MagePower`)
   (h) **Monster Reborn**: Summon the monster with the highest attacking power in both graveyards into your own monster area. This monster does not count in your own (1 monster per turn) rule. This monster does not require any sacrifices regardless its level. (**Class name:** `MonsterReborn`) **Hint:** You will need to use `instance of` to check the type of the cards in the graveyard. Search Java docs for the keyword.
   (i) **Pot of Greed**: Draw 2 extra cards. (**Class name:** `PotOfGreed`)
   (j) **Raigeki**: Destroy all monsters your opponent controls. (**Class name:** `Raigeki`)

5 Build the (Deck) Class

Name : `Deck`

Package : `eg.edu.guc.yugioh.board.player`

Description : Represents the group of cards used by each player to draw new cards

5.1 Class attributes

1. `ArrayList<Card> monsters`: List of all 30 monsters available in the game. The list is loaded from the CSV file. It should be loaded only **once** and used by **all** `Deck` instances. The attribute is *private*.
2. `ArrayList<Card> spells`: List of all 10 spells available in the game. The list is loaded from the CSV file. It should be loaded only **once** and used by **all** `Deck` instances. The attribute is *private*.
3. `ArrayList<Card> deck`: List of cards available as the deck for one player. The deck starts with 20 cards. It must have 15 monsters and 5 spells. The attribute is READ ONLY.

5.2 Description of CSV files format

1. Each line represents a card.
2. The data has no header, i.e. the first line represents the first card.
3. The parameters are separated by a comma (,).
4. The data of the spell cards is found in `Database-Spells.csv`. The data of a **spell** is added in the file as follows: **Type, Name, Description**. The type is `Spell`.
5. The data of the monster cards is found in `Database-Monsters.csv`. The data of a **monster** is added in the file as follows: **Type, Name, Description, Attack points, Defense points, Level**. The type is `Monster`.

Example 3 *An entry for a **Monster** will look like:*
Monster,My monster,Strongest monster in the world,2000,100,5

Example 4 *An entry for a **Spell** will look like:*
Spell,My spell,Ends the game

5.3 Class methods

1. `public Deck()`: Constructor to read the cards from the CSV if they have not been read before. The constructor also builds a shuffled deck of 20 cards, 15 of which are monsters and 5 are spell cards.
2. `public ArrayList<Card> loadCardsFromFile(String path)`: Loads a list of cards from the CSV file.
3. `private void buildDeck(ArrayList<Card> monsters, ArrayList<Card> spells)`: Builds a deck from the list of available cards in the game randomly. The deck must have 15 monsters and 5 spells in this order. Duplicate cards are allowed in the deck. **Hint:** Copy the values of the card to a new instance before adding it to the deck. You can use the `clone()` method.
4. `private void shuffleDeck ()`: Randomizes the order of cards in the deck
5. `public ArrayList<Card> drawNCards(int n)`: Returns N cards from the top of the deck
6. `public Card drawOneCard()`: Returns 1 card from the top of the deck

6 Build the (Field) Class

Name : `Field`

Package : `eg.edu.guc.yugioh.board.player`

Description : Represents the field controlled by each player

6.1 Class attributes

All the class variables are READ ONLY. However, the `Phase` is READ and WRITE.

1. `Phase phase`: Value from (`Phase`) enumeration. Possible values are: `MAIN1`, `BATTLE`, `MAIN2`.
2. `ArrayList<MonsterCard> monstersArea`: List of the monsters on the board for a player. Maximum size is 5.
3. `ArrayList<SpellCard> spellArea`: List of the spells on the board for a player. Maximum size is 5.
4. `Deck deck`: List of cards used as the deck of the player.
5. `ArrayList<Card> hand`: List of cards in my hands used to add cards to the board.
6. `ArrayList<Card> graveyard`: List of cards removed from the field and dispensed.

6.2 Class methods

1. Empty constructor to initialize all the field parameters to empty lists except the deck. The deck should have 20 shuffled cards, 15 of which are monsters and 5 are spells.
2. `public void addMonsterToField(MonsterCard monster, Mode m, boolean isHidden)`: The method is used to add a monster to the field if there are available empty monster spaces. The mode defines whether its **DEFENSE** or **ATTACK** and the 3rd parameter defines its visibility.
3. `public void addMonsterToField(MonsterCard monster, Mode mode, ArrayList<MonsterCard> sacrifices)`: The method also adds a monster to the field if there are available empty monster slots. It also checks if the monster has enough sacrifices or not and performs the sacrifices if needed. If the monster does not need any sacrifices, `sacrifices` should be `null`.
4. `public void removeMonsterToGraveyard(MonsterCard monster)`: A method to remove a monster from the field to the graveyard
5. `public void removeMonsterToGraveyard(ArrayList<MonsterCard> monsters)`: A method to remove multiple monsters from the field to the graveyard.
6. `public void addSpellToField(SpellCard action,MonsterCard monster, boolean hidden)`: Adds a spell card to the field if there are available slots. The spell card could be either set or activated based on its visibility. If the spell card is set or does not require a monster for its actions, the `monster` should be `null`.
7. `pubic void activateSetSpell(SpellCard action, MonsterCard monster)`: Activates a spell card that was set in the field then sends it to the graveyard once it finishes its action. If the card does not need any monsters for its action, the `monster` should be `null`.
8. `public void removeSpellToGraveyard(SpellCard spell)`: A method to remove a spell from the field to the graveyard
9. `public void removeSpellToGraveyard(ArrayList<SpellCard> spells)`: A method to remove multiple spells from the field to the graveyard

10. `public void addCardToHand()`: A method to remove one card from the deck and add it to the hand.
11. `public void addNCardsToHand(int n)`:A method to remove n cards from the deck and add it to the hand.

7 Build the (Player) Class

Name : `Player`

Package : `eg.edu.guc.yugioh.board.player`

Description : Represents a player in the game

7.1 Class attributes

1. `String name`: Represents the player name. The name should be initialized in the constructor once. Afterwards, it should be READ-ONLY.
2. `int lifePoints`: Integer representing the life points of the player. The initial value is 8000. It is a READ and WRITE variable.
3. `Field field`: An object representing the playing setup for 1 player. The board is composed of 2 fields. The field is a READ-ONLY attribute.

7.2 Class methods

1. `public Player(String name)`: A constructor to initialize the parameters of the class.

8 Build the (Board) Class

Name : `Board`

Package : `eg.edu.guc.yugioh.board`

Description : Main class for running the game

8.1 Class attributes

All class variables are READ and WRITE.

1. `Player activePlayer`: The player who currently has the turn
2. `Player opponentPlayer`: The player who is currently inactive
3. `Player winner`: Indicates the winner of the game. No one has won yet if it is set to `null`.

8.2 Class methods

1. Create an empty public constructor for the class. Remember to initialize the `board` attribute in `Card` class.
2. `public void whoStarts(Player p1, Player p2)`: Defines randomly whether player 1 or player 2 should start. The starting player is set as the `activePlayer`.
3. `public void startGame(Player p1, Player p2)`: This method starts the game by drawing five cards for each player. Afterwards, it selects a random player to start the game. It also draws an extra card for the chosen player.
4. `public void nextPlayer()`: Switches the role to the next player and makes him draw 1 card.