Lab 3: Abstraction

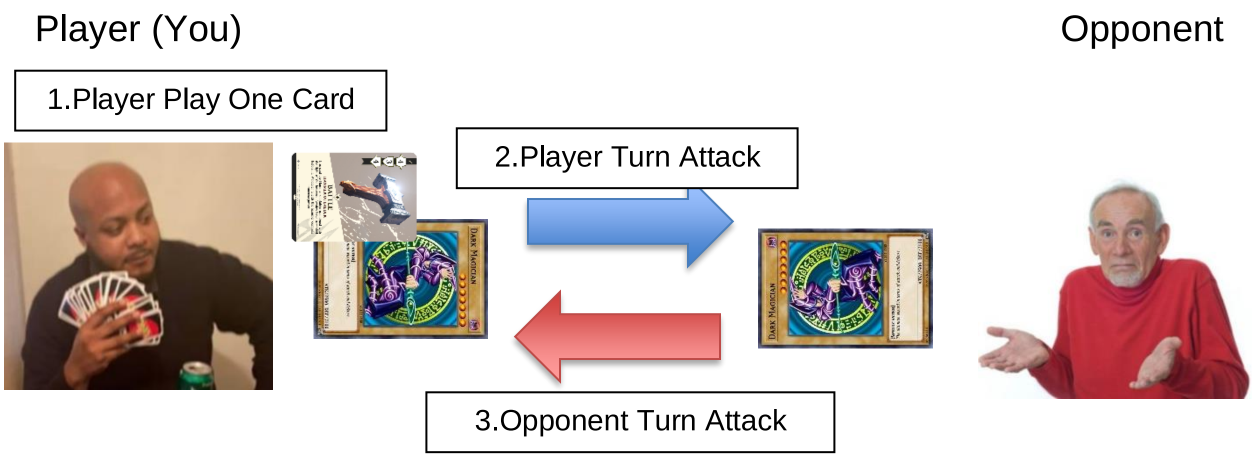
Instruction

1. Click the provided link on CourseVille to create your own repository.
2. Open Eclipse and then "File > new > Java Project" and set project name in this format **2110215_Lab3_2020_2_{ID}_{FIRSTNAME}**
   - Example: **2110215_Lab3_2020_2_6131234521_Jotaro**.
3. Initialize git in your project directory
   - **Add .gitignore.**
   - Commit and push initial codes to your GitHub repository.
4. Implement all the classes and methods following the details given in the problem statement file which you can download from CourseVille.
   - You should create commits with meaningful messages when you finish each part of your program.
   - Don't wait until you finish all features to create a commit.
5. Test your codes with the provided JUnit test cases, they are inside package **test.student**
   - **You have to write some tests by yourself**. Put them inside package **test.student**
   - Aside from passing all test cases, your program must be able to run properly without any runtime errors.
   - There will be additional test cases to test your code after you submit the final version, **make sure you follow the specifications in this document**.
6. After finishing the program, create a UML diagram using **ObjectAid** and put the result image (**UML.png**) at the root of your project folder.
7. Export your project into a jar file called **Lab3_2020_2_{ID}** and place it at the root directory of your project. **Include your source code in the jar file**.
   - Example: **Lab3_2020_2_6131234521.jar**
8. Push all other commits to your GitHub repository.

1. Problem Statement: Children's Card Game

After your success with creating a toy solar system, you receive an invitation from a toy company, "Wizards of the Rich!". The company needs your help to create a children's card game. But this company has no experience making one before (but that does not stop them because they want cash money). So, you decide to make a simple card game for them.

1.1 Gameplay



In your prototype, a player duels a fixed opponent who never plays any card, which means most of the card functions will only be used by the player.

1.1.1 Game Component

1.Player Character Card

- At the start of the game, each player will have a default character as an avatar. You choose Player Character before you choose your deck.

Each has different stats of:

- Life Point = if Life Point of the character in played reaches 0, the corresponding player loses the game.
- Attack Point = The amount of damage this card deals to the opponent when attacking.
- Defense Point = The amount of damage this card reduces when defending against an opponent's attack.

2.Deck

- Contains as many cards as you want, you can insert and remove any card at the card shop. However, you can only have up to 4 of the same cards.

3. Card

3.1 Character Card

- Similar to player character, Character card contains Life Point, Attack Point, and Defense Point.

Player can choose to perform 1 of 2 actions when playing this card.

1. Switch Character
- All of current character's equipped items will be retired.
- Replace current character in play with this card.
- Remaining Life Point percentage from previous character will be transfer to new character. (Ex Current character LP = 75/100 -> New character = 112/150)

2. Sacrifice
- With the cost of this card, you can increase current character Life Point (by percentage). Different types of card increase different percentage of Life Point.

3.1.1 Basic Character Card
- Character card with only switch and sacrifice actions.
- Sacrificing it heals the current character for 1/6 of the current character's max LP.

3.1.2 Main Character Card
- Character card that can level up when this card is in play and Player uses the same card. Level up increases all 3 of character stats.
- Sacrificing it heals the current character for 1/8 of current character's max LP.

3.1.3 Exodia Character Card
- Character card that when equip with 4 Exodia Part Cards (Item Card), Player will immediately win the game.
- Sacrificing it heals the current character for 1/10 of the current character's max LP.

3.2 Item Card
- Item Card is used to increased character stats, it contains LP bonus, Attack bonus, and Defense bonus.
- Item Card only has 1 action when played, equip item, which increases the current character's stats. Character can have any number of item cards equipped, but all equipped item cards will be lost when player switches character.

3.2.1 Basic Item Card
- Item card with only previously mentioned function.

3.2.2 Exodia Part Card
- Item card that only increases character Defense Point, if this card is equipped on to Exodia Character Card, Defense Point increase will be doubled.

1.1.2 Game Flow

At the start of the game, the player shuffles his deck and draw 5 cards. A player character card is played as a placeholder character.

1.Start with the player turn, the player draws a card.

2.Player chooses to play one card and pick its action.
3.Player character automatically attacks opponent character:
- Damage dealt = Attacker's Total Attack Point – Defender's Total Defense Point (Damage can't be negative)

4.Start the opponent turn, opponent character automatically attacks player character.
5.Repeat until win condition is met by either player.

2. Implementation Details:

To complete this assignment, you need to understand about **Abstract Classes** and **Junit Test Cases**.

To test your understanding about abstraction, **we will not provide class diagram for this assignment, and we will not indicate which methods and classes are abstract.** Try your best to figure out. There are **three** abstract classes and **three** abstract methods.

There are **five** packages in the provided files: `application`, `player`, `card`, `deck` and `test`.

You will be implementing most of the class in the `card` and `deck` package (Every class is partly given, while PremadeDeck class doesn't need to be modified).

There are some test cases given in package `test.student`. These will help test your code whether it will be able to run or not. However, **some conditions are not tested** in these test cases. **Look for those conditions in the class details**. **You must create your own test cases for such cases.**

You can define any additional number of private (but not public, protected or package) fields and methods in addition to the fields and methods specified below. You are encouraged to try to group your logic into private methods to **reduce duplicate code as much as possible**.

** Noted that Access Modifier Notations can be listed below*
+ (public), # (protected), - (private)

2.1 package `deck`

1.1.3 Class: `Deck`

This class represents a deck player will used. Deck have 3 attributes: **name, deckSize, and deckList**. Player can assign deck to use in gameplay. **Deck can only have at most 4 of the same card**

- name = name of the deck
- deckSize = number of cards the deck contains, need to change when Deck is created, when card is inserted into the Deck, and when card is removed from the Deck.
- deckList = list of the cards this deck contains. Each card must be assigned to an array without any empty slot.

2.1.1.1 Constructors

+ Deck(String name, Card[] deckList)	Construct a Deck object with the given name , deckList, and initialize `deckSize` to be the same size as the given deckList. (Hint: You should use Array (`Card[]`) to implement inventory.).

2.1.1.2 Methods

+ int insertCard(Card card) throws InsertCardFailedException	Insert the given card into **the bottom of card list (Recommend: create a new copy of deckList array with one more slot)** and modify deckSize to be accurate to the new deckList. Throw an InsertCardFailedException with message "`You can only put 4 of the same cards into the deck`" if there are already 4 of the same card in the deck. Otherwise, return the new deckSize. (Given, but can be changed or adjustd according to how student code inserts card). Hint: You can use isEqual(Card othercard) method from Class card.
+ Card `removeCard`(int slotNumber) throws RemoveCardFailedException	Unequip a card in the given slot number from the deck, rearrange the card from every slot after it to replace the empty slot. Modify deckSize to be accurate to the new deckList, and return the removed card.

	Throw an RemoveCardFailedException with message "`Number you insert exceed deck size`" when the slotNumber is greater than or equal to deckSize. (Already Implemented)
+ String toString()	You do **not** have to edit this method.
+ int Card[] getDeckList() + int String getName() + int getDeckSize()	Getter methods.
+ void setDeckSize(int deckSize)	Setter method.

2.2 package `card.base`

2.2.1 Class `Card`

This class is the base class of all cards in the card shop. Each card has its own name and description. `Card` **should never be instantiated into objects**, as it is only designed to be a base class so that consumer classes (like `Deck`) can easily use their subclasses.

2.2.1.1 Constructors

+ Card(String name, String Description)	Initialize the card with the given name and description.

2.2.1.2 Methods

+ String toString()	**this method should never be called with from this card, it can only be called from subclasses.**
+ String getName() + String getDescription()	Getter methods.

2.2.2 Class `CharacterCard`

This class is a base class of character card, used as an avatar for player to use in gameplay. It has lifePoint, attackPoint, and defensePoint. It provides shared implementation of displaying names across all upgradable items. `CharacterCard` **should never be instantiated to objects,** as it is only designed to be a base class so that consumer classes (like the `Deck and Player`) can easily use their subclasses.

2.2.2.1 Constructors

+ CharacterCard (String name, String Description , int lifePoint, int attackPoint, int defensePoint)	Initialize the character card with the given name , description, lifePoint, attackPoint, and defensePoint.

2.2.2.2 Methods

+ void switchCharacter(Player player)	Switch character of the player, which also include changing player lifePoint, attackPoint, and defensePoint using those of the card. Hint: Use `setNewCharacterLifePoint(int)` `,setAttack(int)` `, setDefense (int)` `, setAssignedCharacter (CharacterCard)` from Player class. setNewCharacterLifePoint already uses percentage calculation. It will change current life point from player's previous life point already. You can simply insert lifepoint into this method.
+ int sacrifice(Player player)	Called when using CharacterCard to increase player life point. Life point gain is different for each class **this method should never be called from this class. It can only be called from subclasses.**
+ String toString()	You do **not** have to edit this method.

2.2.3 Class `ItemCard`

This class is a base class of item card, a player can equip item to increase lifePoint, attackPoint, and/or defensePoint. It has lpBonus, attackBonus, and defenseBonus. It provides shared implementation of displaying names across all item cards. `ItemCard` **should never be instantiated to objects,** as it is only designed to be a base class so that consumer classes (like the `Deck and Player`) can easily use their subclasses.

2.2.2.1 Constructors

+ ItemCard(String name, String description, int lpBonus , int attackBonus, int	Initialize the item card with the given name ,description, lpBonus, attackBonus, and

defenseBonus)	defenseBonus

2.2.2.2 Methods

+ void equipItem(Player player)	Called when using ItemCard to increase player stats. **This method should never be called from this class. It can only be called from subclasses.**
+ String toString()	You do **not** have to edit this method.

2.3 package `card.cards`

This package contains implementation of the **concrete** card in card shop. Some classes should be extended from `card.base` package while some should be extended from this package. In any case, every class must be extended from Card class at its root.

2.3.1 Class `BasicCharacterCard`

This class represents "basic character card" that has no additional logic or properties

2.3.1.1 Constructors

+ BasicCharacterCard (String name, String description, int lifePoint, int attackPoint, int defensePoint)	Initialize the basic character card with the given name, description, and other attributes.

2.3.1.2 Methods

+ int sacrifice(Player player)	Called when using BasicCharacterCard to increase player life point. Life point gain is different for each class **Return life point increased from this method** (Do not have to consider whether healing lifePoint exceed maxLifePoint) **The player is healed equal to 1/6 of player maxLifePoint** Hint: Use `getMaxLifePoint()`

	, `healPlayer(int)` from Player class

2.3.2 Class `BasicItemCard`

This class represents “basic items card” that has no additional logic or properties

2.3.1.1 Constructors

+ BasicItemCard(String name, String description, int lpBonus, int attackBonus, int defenseBonus)	Initialize the item card with the given name, description, and the attribute bonuses.

2.3.1.2 Methods

+ void equipItem(Player player)	Called when using this card to increase player stats **and add this ItemCard to player inventory. In case of lpBonus, it must apply on both player’s currentLifePoint and maxLifePoint.** Hint: Use `addInventory(Itemcard)` , `setMaxLifePoint(int)` , `setCurrentLifePoint(int)` , `setAttack(int)` , `setDefense(int)` from Player class

2.3.3 Class `MainCharacterCard`

This class represents “main character card”. Very similar to CharacterCard but with 2 more properties and 3 more methods. Main character start with level 0. If a player currently plays main character card, it can **levelUp,** which increases level by 1 and increases player stats equal to **levelUpBonus multiplier**. (levelUp can be done by playing the same card as the main character card, but you don’t have to implement that part).

- int level = current level of this character, start with 0 when initialized
- float levelUpBonus = stat increase multiplier when levelUp,

Ex: If levelUpBonus = 0.3, when levelUp, player lifePoint, attackPoint, and defensePoint increase by 30% (round down to int)

2.3.3.1 Constructors

+ MainCharacterCard (String name, String description, int lifePoint, int attackPoint, int defensePoint, float levelUpBonus)	Initialize the main character card with the given name, description, levelUpBonus, and the attribute.

2.3.3.2 Methods

+ int sacrifice(Player player)	Called when using MainCharacterCard to increase player life point. Life point gain is different for each class. **Return life point increased from this method** (don't have to consider whether healing lifePoint exceed maxLifePoint) **Player is healed equal to 1/8 of player maxLifePoint.** Hint: Use `getMaxLifePoint()` `, healPlayer(int)` from Player class

+ float levelUp(Player player)	Called when using MainCharacterCard to **increase lifePoint, attackPoint, and defensePoint equal to its multiplier with levelUpBonus. Also increase level by 1.**
	
	Return levelUpBonus
	Ex: If levelUpBonus = 0.3, when levelUp, player lifePoint, attackPoint, and defensePoint increase by 30% (round down to int). Lifepoint increases from maximum lifepoint.
	Hint:
	Use `setNewCharacterLifePoint (int)`
	`, setAttack (int)`
	`, setDefense (int)`
	from Player class
	
	`setNewCharacterLifePoint(int)` method will change current life point from player previous life point already. You can simply insert lifepoint into this method.

+ String getName()	You do **not** have to edit this method.

+ int getLevel()	Getter methods.
+ void setLevel(int level)	Setter methods.

2.3.4 Class `ExodiaCharacterCard`

This class represents “exodia character card”. Exodia is a CharacterCard with:

name = “`"Exodia the Forbidden One"`”

description = `"With 4 or more Exodia Part Card equiped, you win the game"`

lifePoint = 800

attackPoint = 0

defensePoint = 25

Additionally, it has one more method called winConditionCheck that check if Exodia is equip with 4 ExodiaPartCard or not. If it’s true, player win the game immediately (You don’t have to implement winning part, you only have to return true or false).

2.3.4.1 Constructors

+ ExodiaCharacterCard()	Initialize the main character card with the given name, description, and the attribute. (See above)

2.3.4.2 Methods

+ int sacrifice(Player player)	Called when using ExodiaCharacterCard to increase player life point. Life point gain is different for each class **Return life point increased from this method** (don’t have to consider whether healing lifePoint exceed maxLifePoint) **Player is healed equal to 1/10 of player maxLifePoint.** Hint: Use `getMaxLifePoint()` `, healPlayer(int)` from Player class
+ boolean winConditionCheck(ItemCard[] inventory)	Called to check if the inventory contains 4 or more ExodiaPartCard. Return true if it has 4 or more, false otherwise. Hint: Use `instanceof to check if ItemCard is`

	ExodiaPartCard

2.3.5 Class `ExodiaPartCard`

This class represents "exodia items card" Exodia is an ItemCard with:

description = `"Assemble 4 of Exodia part card to win the game"`

lifeBonus = 0

attackBonus = 0

name and defenseBonus depends on initialization

2.3.5.1 Constructors

+ ExodiaPartCard(String name, int defense)	Initialize the main character card with the given name, description, and the attributes. (See above)

2.3.5.2 Methods

+ void equipItem(Player player)	Called when using this card to increase player stats **(only defensePoint in this case) and add this card to player inventory.**
	If player character is ExodiaCharacterCard, increase player stats by twice of this bonus stats instead
	Hint:
	Use `addInventory(Itemcard)`
	`, setMaxLifePoint(int)`
	`, setDefense(int)`
	`, getAssignedCharacter()`
	from Player class
	Use `instanceof to check if player's CharacterCard is ExodiaCharacterCard`

2.4 package `test.student`

2.4.1 Class `TestDeck`

This class test Card class from card.base package. All constructors and some test cases has already been implemented, **except for 2 test cases which student needs to implement.**

2.4.1.1 Methods (Test cases)

+ void testRemoveCard()	**This test case must test 2 scenarios.** 1. Remove any card from the deck, then check if that card inside the deck is no longer there, number of cards is reduced, and other cards are rearranged correctly. 2. Repeat scenario 1, but with a different deck.
+ void testRemoveNonExsistanceCard ()	Use assertThrows to check RemoveCardFailedException by removing card form the deck on the slot that doesn't have card in it.

2.4.2 Class `TestExodiaCharacterCard`

This class test ExodiaCharacterCard class from card.cards package. All constructors and some test cases has already been implemented, **except for 3 test cases which student needs to implement.**

2.4.2.1 Methods (Test cases)

+ void testSwitchCharacter ()	Switch player's character to Exodia, and check if player's stats is the same as ExodiaCharacterCard.
+ void testSacrifice ()	Set player's life point not to be full, and sacrifice Exodia to player and check if player's life point increases accordingly.
+ void testWinConditionCheck ()	Use winConditionCheck on different ItemCard array (already given) and check if results are correct for each array.

2.4.3 Class `TestExodiaPartCard`

This class test ExodiaPartCard class from card.cards package. All constructors and some test cases has already been implemented, **except for 1 test case which student needs to implement.**

2.4.3.1 Methods (Test cases)

+ void testEquipItemExodiaCase ()	Switch player's character into Exodia and equip 3

	different ExodiaPartCards to it, and check if player's Defense increase accordingly.

2.4.4 Class `TestMainCharacterCard`

This class test MainCharacterCard class from card.cards package. All constructors and some test cases has already been implemented, **except for 2 missing and 1 incomplete test cases which student needs to implement.**

2.4.2.1 Methods (Test cases)

+ void testSwitchCharacter ()	Switch player's character into testMainChar, and check if player's stats is the same as testMainChar
+ void testSacrifice ()	Set player's life point to not be full, and sacrifice testMainChar to player and check if player's life point increases accordingly.
+ void testLevelUp () (Given, but incomplete)	The code already tests levelUp method once from level 0 to level 1. Student must use levelup method on player again and check player's stats and level when moving from level 1 to level 2.

Score Criteria

TestBasicCharacterCard	5
testConstructor	1
testSwitchCharacter	2
testSacrifice	2
TestBasicItemCard	4
testConstructor	1
testEquipItem	3
TestDeck	12
testConstructor	1
testInsertCard	3
testInsertCardMoreThan4Card	2
Write testRemoveCard	3

	Write testNonExsistanceRemoveCard	3
TestExodiaCharacterCard		8
	testConstructor	1
	Write testSwitchCharacter	2
	Write testSacrifice	2
	Write testWinConditionCheck	3
TestExodiaPartCard		6
	testConstructor	1
	testEquipItem	3
	Write testEquipItemExodiaCase	2
TestMainCharacter		8
	testConstructor	1
	testLevelUp	3
	Write testSwitchCharacter	2
	Write testSacrifice	2