Lab 3: Abstraction

Instruction

1. Click the provided link on MyCourseVille to create your own repository.
2. Open Eclipse and create a project

   Progmeth_lab3_2024_1_{ID}_{FIRSTNAME}
   - Example: Progmeth_lab3_2024_1_**6631234521_Sataporn**.
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 (copying initial code from the given lab file) 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 the features to create a commit.
5. Test your codes with the provided JUnit test cases, they are inside package **grader.student**
   - **You have to write some tests by yourself**. Put them inside package **grader.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 and put the result image (**UML.png**) at the root of your project folder.
7. Export your project into a jar file called **Lab3_2024_1_{ID}** and place it at the root directory of your project. **Include your source code in the jar file**.
   - Example: **Lab3_2024_1_6631234521.jar**
8. Push all other commits to your GitHub repository.

1. Problem Statement: Lanna Ghost War

In this lab, we will implement the traditional believe about the ghost of Lanna. This is the greatest ghost game in all of labs in this course and all of games in the world "Lanna Ghost War". There have many of Lanna's ghosts and many of actors and items in this lab. Wait a minute! There will be an easter egg from the Lab II. Now, let's read and implement it together.

1. Game Flow

1. This game has a lot of actors in actors, a lot of items in items and a lot of ghosts in ghosts.
2. At the start of game, there will be a villager in actor, empty items and randomly generated 10 LowGhost in ghosts. There will be 10 HP and 0 scores.
3. On each player's turn, the player chooses an actor to play.
4. There can be effects produced by items.
5. A ghost: the first ghost in ghosts (we call it the current ghost) list attacks the player.
6. An actor attacks the current ghost.
7. If a ghost is destroyed, then the player' score will increase by that ghost's level. Next, the ghost is removed from ghosts list and the system will assign a new ghost randomly to ghosts list.
8. Steps 2-7 are repeated for each player until the game ends.
9. The game ends when the actors list is empty or the player's hp is lower than or equal to 0 (you lose).

2. Ghosts

1. Class Ghost is the base of all ghosts in the game.
2. LowGhost is the type of Ghost that has the same level and ability to attack players.
3. HighGhost is the type of Ghost that is more powerful than LowGhost and has ability to attack and damage players.
4. GaGhost: one of the LowGhost that has an energy and the ability to attack the player.
5. MaBongGhost: one of the LowGhost that has a speed of running, power and the ability to attack the player.
6. PryGhost: one of the LowGhost that has a power and the ability to attack the player. We know that PryGhost will have the ability to attack the player relevant to the quantity of a substance (see more detail later).
7. PongGhost: one of the HighGhost that has a power and the ability to attack and damage the player.
8. PooYaGhost: one of the HighGhost that has a power and the ability to attack and damage the player. This is the most powerful ghost in this game.

3. Actors

1. Class Actor is the base of all actors in the game.
2. Villager is an actor that has the lowest ability to attack ghost.
3. Monk is the only actor that has the ability to attack HighGhost because there are the propagators of Buddhism.
4. GhostDoctor is an actor that can kill a LowGhost in one attack.
5. Monkey is an actor from Lopburi Monkey Lab. Monkey can kill a LowGhost with one hit if the player has a banana in items list.

4. Items

1. Class Item is the base of all items in the game.
2. Amulet is an item that has the effect of healing the player’s hp.
3. Leklai is an item that has the effect of helping actors when attacking a ghost.
4. Banana is an item that has an effect to show all the ghosts in ghosts list. It can help the player manage game planning.

2. Implementation Details:

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

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

There are packages in the provided files: *application*, *logic.actor*, logic.ghost, logic.item, *logic.game*, *test* and *utils*.

You will be implementing all of the classes in the *logic.ghost, logic.item, logic.actor* package and some methods in *logic.game*. (Every class is partly given.) Note that only important methods that you need to write and methods that you need to complete this question are listed below.

There are some test cases given in package *grader.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 <u>private</u> (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 logic.ghost

2.1.1 Class Ghost /*This class is partially given*/

This class is a base class for all the Ghosts. It contains all common elements of a ghost in this game. Note that this class should never be instantiated. It acts like a base code for other types of ghosts.

Fields

- int hp	The ghost's hp.

Methods

+ Ghost(int **hp**)	Constructor. Set fields to the given parameter values.
+ isDestroyed()	Returns **true** if hp is a nonpositive integer.
+ void decreaseHp(int **amount**)	Decrease **ghost's hp** by the given **amount**. *Note: that **hp** cannot be a negative integer.*
+ int getLevel()	Return a ghost's level. Each type of ghost has a different level.
+ int getHp()	Return the ghost's hp.
+void attack()	To attack the player. Each of HighGhost has different ways to damage the player.

2.1.2 Class LowGhost /*This class is partially given*/

This class extends Ghost that represents a Low Ghost: the type of ghost. All of Low ghosts has a same level. Note that this class should never be instantiated. It acts like a base code for other types of low ghosts.

Methods

+ LowGhost()	Constructor. Set hp field of a super class to **Config.LowGhostHp** . *Note: You can use the constants in **class Config**.*

+ int getLevel()	Returns **Config.LowGhostLevel** . *Note: You can use the constants in **class Config**.*

2.1.3 Class HighGhost /*This class is partially given*/

This class extends Ghost that represents a High Ghost: the type of ghost. Note that this class should never be instantiated. It acts like a base code for other types of high ghosts.

Methods

+ HighGhost()	Constructor. Set hp field of a super class to **Config.HighGhostHp** . *Note: You can use the constants in **class Config***
+ void damage()	To damage the player. Each of HighGhost has a different way to damage the player.

2.1.4 Class GaGhost /*This class is partially given*/

This class represents GaGhost: a one of the LowGhost that has an energy and the ability to attack the player.

Fields

- int energy	The ghost's energy.

Methods

+ GaGhost()	Constructor. Set energy to **Config.GaGhostEnergy** . *Note: You can use the constants in **class Config*** .
+ GaGhost(int **energy**)	Constructor. Set energy to **energy** .
+ String toString()	Returns *"GaGhost [HP: **${hp}** , Energy: **${energy}**]"* Where ● **${hp}** is this ghost's hp ● **${energy}** is this ghost's energy.

+ void attack()	GaGhost attacks the player. ● Decrease **player's hp** by **ghost's energy**. *Hint: You can set player's hp by using setter in* ***GameController.getInstance()***

2.1.4 Class MaBongGhost /*This class is partially given*/

This class represents MaBongGhost: a one of the LowGhost that has a speed of running, power and the ability to attack the player.

Fields

- int power	The Mabong ghost's power.
- int speed	The Mabong ghost's speed

Methods

+ MaBongGhost()	Constructor. Set power to **Config.MaBongGhostPower** and set speed to **Config.MaBongGhostSpeed**. *Note: You can use the constants in* ***class Config***.
+ MaBongGhost(int **power**)	Constructor. Set power to **power** and set speed to **Config.MaBongGhostSpeed**. *Note: You can use the constants in* ***class Config***.
+ MaBongGhost(int **power,** int **speed**)	Constructor. Set power to **power** and set speed to **speed.**
+ String toString()	Returns *"MaBongGhost [HP:* ***${hp}*** *, Power:* ***${power}*** *, Speed:* ***${speed}****]"* Where ● **${hp}** is a ghost's hp ● **${power}** is a ghost's power. ● **${speed}** is a ghost's speed.
+ void attack()	MaBongGhost attacks the player. ● Decrease **player's hp** by **ghost's power * ghost's speed**. *Hint: You can set player's hp by using setter in* ***GameController.getInstance()***
+ getter and setters	Getter and Setters for **speed**.

2.1.5 Class PryGhost /*This class is partially given*/

This class represents PryGhost: a one of the LowGhost that has a power, the ability to attack the player and ppt: usually means "parts per trillion", it occasionally means "parts per thousand". A part per trillion (ppt) is a measurement of the quantity of a substance in the air, water or soil. We know that PryGhost will have the ability to attack the player relevant to the quantity of a substance.

Fields

- int power	The ghost's power.
- int ppt	A part per trillion (ppt) is a measurement of the quantity of a substance in the air, water or soil.

Methods

+ PryGhost()	Constructor. Set power to **Config.PryGhostPower** and set ppt to **0**. *Note: You can use the constants in **class Config** .*
+ PryGhost(int **power**)	Constructor. Set power to **power** and ppt to 0. *Note: You can use the constants in **class Config** .*
+PryGhost(int **power,** int **ppt**)	Constructor. Set power to **power** and set ppt to **ppt .**
+ String toString()	Returns *"PryGhost [HP: **${hp}** , Power: **${power}** , PPT: **${ppt}**]"* Where ● **${hp}** is a ghost's hp ● **${power}** is a ghost's power. ● **${ppt}** is a quantity of a substance.
+ void attack()	PryGhost attacks the player. ● Decrease **player's hp** by **ghost's power - substance's ppt**. *Hint: You can set player's hp by using setter in **GameController.getInstance()***
+ getter and setters	Getter and Setters for **ppt**.

2.1.6 Class PongGhost /*This class is partially given*/

This class represents PongGhost: a one of the **HighGhost** that has a power and the ability to attack and damage the player.

Fields

- int power	The ghost's power.

Methods

+ PongGhost()	Constructor. Set power to **Config.PongGhostPower** *Note: You can use the constants in **package Config** .*
+ PongGhost(int **power**)	Constructor. Set power to **power**
+ int getLevel()	Returns **Config.PongGhostLevel .** *Note: You can use the constants in **package Config** .*
+ String toString()	Returns *"PongGhost [HP: **${hp}** , Power: **${power}**]"* Where ● **${hp}** is a ghost's hp ● **${power}** is a ghost's power.
+ void attack()	PongGhost attacks the player. ● Decrease **player's hp** by **ghost's power** *Hint: You can set player's hp by using setter in **GameController.getInstance() .***
+ void damage()	PongGhost damages the player. ● For each **ghost** in **ghosts list**, if that **ghost** is LowGhost, then increase **ghost's hp** by PongGhost's power. *Hint:* 1. *You can get Game's **ghosts list** by using getter in **GameController.getInstance().getGhosts()*** 2. *You can increase **ghost's hp** by using **decreaseHp(int amount). (use negative amount)***

2.1.7 Class PooYaGhost /*This class is partially given*/

This class represents PooYaGhost: a one of the **HighGhost** that has a power and the ability to attack and damage the player. There is the most powerful ghost in this game.

Fields

- int power	The ghost's power.

Methods

+ PooYaGhost(int **power**)	Constructor. Set power to **power**
+ int getLevel()	return **Config.PooYaGhostLevel .** *Note: You can use the constants in **package Config** .*
+ String toString()	Returns *"PooYaGhost [HP: **${hp}** , Power: **${power}**]"* Where ● **${hp}** is a ghost's hp ● **${power}** is a ghost's power.
+ void attack()	PooYaGhost attacks the player. ● Decrease **player's hp** by **ghost's power** ● Decrease **player's score** by **ghost's power** *Hint: You can set player's hp and player's score by using setter in **GameController.getInstance() .***
+ void damage()	PooYaGhost damages the player. ● For each **ghost** in **ghosts list**, increase that **ghost's hp** by PooYaGhost's power. *Hint:* 1. *You can get Game's **ghosts** list by using getter in **GameController.getInstance().getGhosts()*** 2. *You can increase **ghost's hp** by using **decreaseHp(int amount) . Use negative amount.***

2.2 package logic.actor

2.2.1 Class Actor /*This class is partially given*/

This class is a base class for all the Actors. It contains all common elements of an actor in this game. Note that this class should never be instantiated. It acts like a base code for other types of actors.

Methods

+ int getLevel()	Returns an actor's level. Each type of actor has a different way to implement this.
+void attack()	To attack a ghost. Each type of actor has a different way to attack a ghost.

2.2.2 Class Villager /*This class is partially given*/

This class represents Villager: a one of the **Actor** that has the ability to attack a ghost in a player turn.

Methods

+ Villager()	Constructor.
+ int getLevel()	Returns **Config.VillagerLevel .** *Note: You can use the constants in **package Config** .*
+ void attack()	If **the first ghost** in **ghosts** list is a HighGhost, do nothing. If **the first ghost** in **ghosts** list is a LowGhost, and the villager has an amulet, decrease the first ghost's hp by **Villager's level + 1**. If **the first ghost** in **ghosts** list is a LowGhost, and the villager does not have an amulet, decrease the first ghost's hp by **Villager's level**. *Hint: You can get any fields by using getter in **GameController.getInstance()***
+ String toString()	Returns the class's name with a capital letter.

2.2.3 Class Monk /*This class is partially given*/

This class represents Monk: a one of the **Actor** that has the ability to attack a ghost in a player turn. Monk is the only actor that have the ability to attack HighGhost because there are the propagators of Buddhism.

Methods

+ Monk()	Constructor.
+ int getLevel()	Returns **Config.MonkLevel .** *Note: You can use the constants in **package Config** .*

+ void attack()	If **the first ghost** in **ghosts** list is a HighGhost, then decrease the ghost's hp by the **Monk's level.** *Hint: You can get any fields by using getter in **GameController.getInstance()**.*
+ String toString()	Returns the class's name with a capital letter.

2.2.4 Class Monkey /*This class is partially given*/

This class represents Monkey: a one of the **Actor** that has the ability to attack a ghost in a player turn. Monkey is an actor from Lopburi Monkey Lab. There can kill a LowGhost at one time if the player have a banana in items.

Methods

+ Monkey()	Constructor.
+ int getLevel()	Returns **Config.MonkeyLevel .** *Note: You can use the constants in **package Config** .*
+ void attack()	If **the first ghost** in **ghosts** list is a not a LowGhost, do nothing. If **the first ghost** in **ghosts** list is a LowGhost: • If the player does not have a banana, then decrease the ghost's hp by **Monkey's level**. • Otherwise, if the player has a **Banana** in his items list, the Monkey will kill that ghost. *Hint: You can get any fields by using getter in **GameController.getInstance()***
+ String toString()	Returns the class's name with a capital letter.

2.2.5 Class GhostDoctor /*This class is partially given*/

This class represents GhostDoctor: one of the **Actor** that has the ability to attack a ghost in a player's turn. GhostDoctor is an actor with a powerful ability to attack ghost. He can kill a LowGhost in a single hit.

Methods

+ GhostDoctor()	Constructor.
+ int getLevel()	Returns **Config.GhostDoctorLevel .** *Note: You can use the constants in **package Config** .*
+ void attack()	If **the first ghost** in **ghosts** list is a LowGhost, then GhostDoctor kills that ghost. *Hint: You can get any fields by using getter in **GameController.getInstance()***
+ String toString()	Returns the class's name with a capital letter.

2.3 package logic.item

2.3.1 Class Item /*This class is partially given*/

This class is a base class for all the Items. It contains all common elements of an item in this game. Note that this class should never be instantiated. It acts like a base code for other types of items.

Methods

+ int getLevel()	Return an item's level. Each type of item has a different level.
+void effect()	Produce the item's effect. Each item has a different effect.

2.3.2 Class Amulet /*This class is partially given*/

This class represents Amulet: one of the **Item** that has the ability to produce an effect in a turn.

Methods

+ Amulet()	Constructor.
+ int getLevel()	Returns **Config.AmuletLevel .** *Note: You can use the constants in **package Config**.*
+void effect()	If the Player's hp is equal or lower than 5, then set the Player's hp to 5. *Hint: You can get any fields by using getter in **GameController.getInstance()***
+ String toString()	Returns the class's name with a capital letter.

2.3.3 Class Banana /*This class is partially given*/

This class represents Banana: a one of the **Item** that has the ability to effect in a player turn.

Methods

+ Banana()	Constructor.
+ int getLevel()	Returns **Config.BananaLevel .** *Note: You can use the constants in **package Config** .*
+ void effect()	Show each ghost's information from ghosts list (each ghost info is separated by a space). Must end by printing a new line. You don't have implement any test case for this class, but you can use it as a reference.
+ String toString()	Returns the class's name with a capital letter.

2.3.4 Class Leklai /*This class is partially given*/

This class represents Leklai: a one of the **Item** that has the ability to effect in a player turn. This item can help actors to attack ghosts.

Methods

+ Leklai()	Constructor.
+ int getLevel()	Returns **Config.LeklaiLevel .** *Note: You can use the constants in **package Config** .*
+ void effect()	For each **ghost** in **ghosts** list, if the ghost's level is equal or lower than Leklai's level, then follows this step. ● If the ghost is a LowGhost, then decrease the ghost's hp by 5. ● If the ghost is a HighGhost, then decrease the ghost's hp by 4. *Hint: You can get any fields by using getter in **GameController.getInstance()***
+ String toString()	Returns the class's name with a capital letter.

2.4 package logic.game

2.4.1 Class GameController /*This class is mostly given*/

- int hp	The player's hp.
- int score	The player's score.
- ArrayList <Actor> actors	The list of player's actors that a player can choose to use in a player's turn.
- ArrayList <Item> items	The list of player's items that all of the items can effect to game ina player's turn.
- ArrayList <Ghost> ghosts	The list of ghosts that are visible in the game. *Note: that only the first ghost that can attack the player in a ghost's turn.*
- GameLogic instance	The current game instance.

Methods to Implement

- void initGame()	Inits game system at the start of game following the steps. ● set Hp to 10. ● set Score to 0. ● add a **new Villager** to **actors.** ● assign randomly **10 new LowGhosts** to **ghosts list**. *Hint: You can create a new LowGhost randomly by using **GameUtils.getRandomGhost(false)***
+ void play(Actor selectedActor)	The following steps are needed: 1. **Activate the Effects** all of **item** in **items** list. 2. Assign local variable **currentGhost** as the first **ghost** in **ghosts list**. 3. currentGhost attacks. 4. **selectedActor attacks** the **currentGhost.** 5. For each **ghost** to be removed from the ghosts list, **increase** the **player's score** by **destroyed ghost's level.** 6. Removes all **destroyed ghost** from **ghosts** list. 7. Assign randomly new **Ghosts** to **ghosts** list The number of these added ghosts must be equal to the number of the **destroyed ghosts.** *Hint: You can create a new Ghost randomly by using **GameUtils.getRandomGhost(true)***
+ boolean isGameOver()	Returns **TRUE** when actors list is empty or player's hp is lower than or equal 0 (you lose).

Useful Methods Provided

+ GameLogic	Constructor. init all of fields and init the game.
+ GameLogic getInstance()	Used to get the game's instance. There is only one instance per game.
+ void addNewActor(Actor **actor**)	Adds **actor** to **actors** list.
+ void addNewGhost(Ghost **ghost**)	Adds **ghost** to **ghosts** list.

+ void addNewItem(Item **item**)	Adds **item** to **items** list.
+ getters	Getter for ghosts list and items list.
+ getters and setters	Getter and Setter for Hp, Score

2.4.2 Class GameIO

This class contains the methods about game input/output: such as selectActor, showItemList, showCurrentGhost, showGameState, buyNewActor and buyNewItem. You don't have to implement any test case for this class, but you can use the class for reference.

2.5 package utils

2.5.1 Class Config

This class contains many constants that are necessary for your implementations. You don't have to implement any test case in this class, but you can use the class for reference.

2.5.2 Class GameUtils

This class contains many methods that are necessary for your implements including getNewGhost, getRandomGhost, getNewActor and getNewItem. You don't have to implement any test case for this class.

2.5.3 Class Randomizer

This class contains methods including getRandomizer that returns java Random. You don't have to implement any test case for this class.

2.6 package grader.student

Your assignment is to **implement additional classes as follows**:

2.6.1 Class GaGhostTest

This class tests GaGhost class from logic.ghost package. All constructors and some test cases have already been implemented, except just one test case below:

void testIsDestroyedFalse()	Test in case GaGhost is not destroyed.

2.6.2 Class MaBongGhostTest

This class tests MaBongGhost class from logic.ghost package. Some test cases have already been implemented, except two test casesbelow:

void testConstructor()	Check all of constructors. You must test all of the different construct cases.
void testDecreaseHpBelowZero()	Test when decrease hp below zero.

2.6.3 Class PongGhostTest

This class tests PongGhost class from logic.ghost package. Some test cases has already been implemented, except two test cases which student needs to implement.

void testConstructor()	Check all of constructors. You must test all of different construct cases.

2.6.4 Class PryGhostTest

This class tests PryGhost class from logic.ghost package. Some test cases have already been implemented, except just one test:

Methods to fill (partially implemented)

void testAttack()	Test different two cases by checking player's hp after being attacked.

2.6.5 Class PooYaGhostTest

This class tests PooYaGhost class from logic.ghost package. Some test cases have already been implemented, except just one test case:

void testDamage()	Test damage by checking all of ghost's hp in the ghosts list. Remember that ghosts list must contain at least 3 ghosts.

2.6.6 Class GhostDoctorTest

This class tests GhostDoctor class from logic.ghost package. Some test cases have already been implemented, except just one test case:

void testAttackHighGhost()	Test attack where there is only one ghost as HighGhost Checking if high ghost's hp is correct after the attack.

2.6.7 Class GameControllerTest

This class tests PryGhostTest class from logic.ghost package. Some test cases has already been implemented, except three test cases which student needs to implement.

Methods to fill (partially implemented)

void testPlayDestroyedGhost()	Test play method where there are 3 ghosts in ghosts list and **currentGhost** is destroyed by the attack Check if the first **ghost** in **ghosts** list is a correct ghost. Check the size of ghosts list after played. *Hint: Please see information about **currentGhost** from **2.4.1 Class GameController**.*
void testPlayWithItem()	Test play method with an item in ArrayList items where there are GaGhost, PryGhost and PooYaGhost After play, check all of ghost's hp in the ghosts list.
void testIsGameOver()	Test game over in 2 different cases.

3. Scoring Criteria (Consider just the total of 100 and be scaled down to the score of 2.5)

Implement Abstract Class Correctly.	**10 Points**
package logic.ghost	**40 Points**
○ GaGhostTest	8 Points
○ MaBongTest	8 Points
○ PryGhostTest	8 Points
○ PongGhostTest	8 Points
○ PooYaGhostTest	8 Points
package logic.item	**13 Points**
○ AmuletTest	4 Points
○ LeklaiTest	5 Points
○ VillagerTest	4 Points
package logic.actor	**17 Points**
○ MonkTest	4 Points
○ GhostDocterTest	4 Points
○ MonkeyTest	4 Points
○ GameController Test	5 Points
Write JUnit.	**15 Points**
○ GaGhostTest	1 Points
○ MaBongGhostTest	2 Points
○ PongGhostTest	1 Points
○ PryGhostTest	1 Points
○ PooYaGhostTest	2 Points
○ GhostDoctorTest	2 Points
○ GameControllerTest	6 Points
UML	**5 Points**
Total	**100 Points**