Lab3: Polymorphism

1 Objective

- Be able to apply the polymorphism concept.

2 Problem Statement: Shooting Game

Shooting is a basic type of game that challenges players to gain points by shooting objects on the screen. Figure 1 is the main screen of the game. In this game, there are many types of gun which have various attack power and many types of object which give different score and some of them are items (gun and bullet). The game ends when remaining time reaches zero, after that, the player can enter his/her name to record the score as shown in Figure 2. Note that a shooting command is "spacebar" or "left click". The game can be paused by using "enter" key.

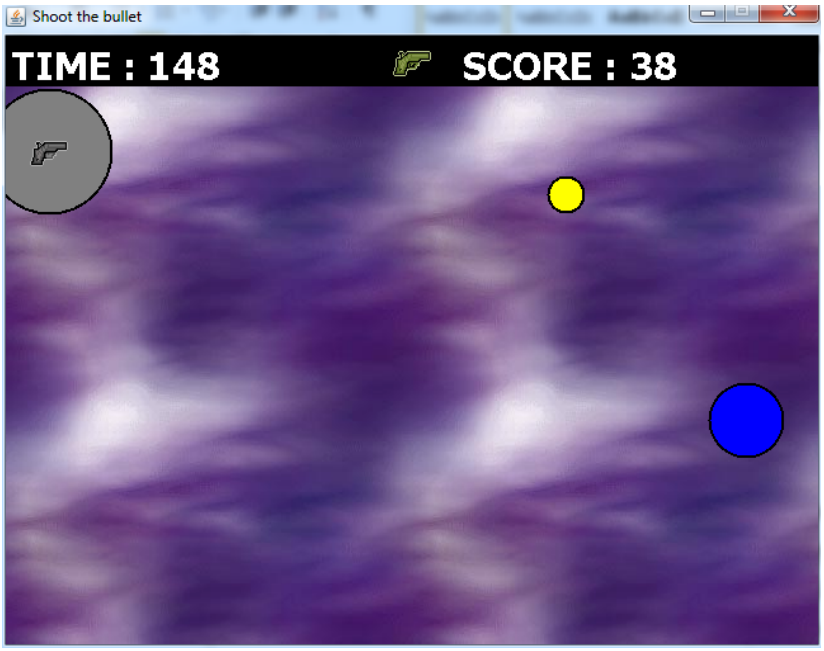

Figure 1. The GUI of Shooting Game

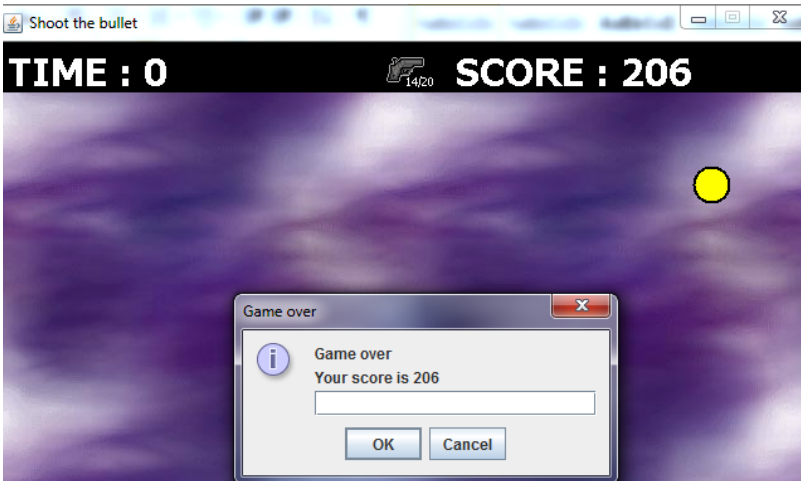

Figure 2. The score recording window

2.1 Player

- There is only one player at a time.
- Player can hold only one gun at a time.
- Player gains score by shooting the objects until remaining time reaches zero.

2.2 Gun

- Each type of gun has different attack power.
- Basic gun comes with unlimited bullets.
- Special gun has limited maximum bullets but it has higher attack power.

2.3 Target Object

- Each target object has different characteristics including starting point defined by (x,y), radius, moving duration, and moving type.
- There are two types of target objects:
    - Shootable object gives score to player who is able to destroy it. An object has life which will be decreased by hit. There are three subtypes of shootable objects: simple target, small target and splitter target. Splitter target breaks into multiple small targets after it has been destroyed.
    - Collectible object gives item to player who grabs and successfully collects them. “Grab” can be done by holding mouse pointer over the object for a specific time duration. “Collect” can be done after “grab”. Item may be special gun or special bullets.

3 Implementation Details

In this part, an implementation of shooting game must follow the class diagram in Figure 3. Students are allowed to add fields and methods if necessary.

Because this game includes drawing picture and playing audio, so, after creating project, two folders are needed:

- Folder named "required jar" contains multiple JARs which are libraries for later use.
- Folder named "res" contains images and audio files.
  - Place "res" folder under "bin" folder.

3.1 Interface IRenderableObject (Library)

This is an *interface* which resides in "lab3_lib_0.jar". The interface specifying method required for the object that can be drawn onto the screen.

3.1.1 Method

- boolean isVisible(); This method returns "true" if the object is visible.
- int getZ(); This method returns index of the object in z-axis. Note that object can overlap other objects on the screen. The top one has the highest value.
- void render(); This method is used for drawing object.

3.2 Class PlayerStatus

It represents a status bar showing current status of the game which is drawn at the top of screen, so, it is renderable (`IRenderableObject`).

3.2.1 Field

- int remainingTime; The remaining time of the game in "tick" unit.
- int score; The current score of player
- Gun currentGun; The current gun that player equips
- boolean pause; It is "true" if the game is paused.

3.2.2 Constructor

- PlayerStatus(); Set `remainingTime` based on `ConfigurableOption.timelimit`. Note that `ConfigurableOption.timelimit` returns time in second, so, you have to convert into "tick" unit using `GameManager.TICK_PER_SECONDS` ratio.

3.2.3 Method

- int getRemainingTime(), int getScore(), Gun getCurrentGun, boolean isPause(); Getters of `remainingTime`, `score`, `currentGun`, and `pause`
- void setCurrentGun (Gun currentGun); Setter of `currentGun`
- void setPause(boolean pause); Setter of `pause`

- void decreaseRemainingTime(int amount); It decreases `remainingTime` by the specified amount. The value must be capped at zero if it goes below zero.
- void increaseScore(int score); It increases score by the given amount.
- boolean isDisplayingArea(int x, int y); It returns `true` if the position (x,y) is in status bar drawing area which is 40 pixel-high. *[provided]*
- boolean isVisible(); This method returns `true` as the status bar is always visible.
- int getZ(); This returns the highest possible value since the status bar must be drawn on top of everything else.
- void render(); This method is used for drawing status bar using `DrawingUtility.drawStatusBar`.

3.3 Class Gun

It represents a basic gun with unlimited bullets. The gun can be drawn as an icon thus is renderable (`IRenderableObject`).

3.3.1 Field

- int attack; The attack power of each gun shot
- All fields can be accessed only from subclasses.

3.3.2 Constructor

- Gun(int attack); It initializes attack power of the gun based on input value.

3.3.3 Method

- int getAttack(); Getter of `attack`
- boolean canShoot(); A basic gun has unlimited number of bullet. So, this method always returns `true`.
- void shoot(); It plays "shoot" sound using `AudioUtility.playSound`. *[provided]*
- boolean isVisible(); It will not be used, so, returning any value is acceptable.
- int getZ(); It will not be used, so, returning any value is acceptable.
- void render(); This method is used for drawing gun icon using `DrawingUtility.drawIconGun`.

3.4 Class SpecialGun

It represents a special gun with higher attack power and limited bullets. This class is a subclass of class `Gun`.

3.4.1 Field

- int bulletQuantity; The remaining bullets
- int maxBullet; The maximum number of bullets that special gun can have.
- All fields can be accessed only from subclasses.

3.4.2 Constructor

- SpecialGun(int bulletQuantity, int maxBullet, int attack); It initializes bullet quantity, maximum number of bullets and attack power of the gun based on input values. Note that the maximum number of bullets should be defined by method `setBulletQuantity`.

3.4.3 Method

- int getBulletQuantity(); Getter of the remaining bullets
- void setBulletQuantity(int bulletQuantity); Setter of the remaining bullets. The value must be capped at zero or maximum if it falls out of lower or upper bound respectively.
- boolean canShoot(); It returns `true` if there is at least one bullet left.
- void shoot(); It is the same as in class `Gun` but also decreases the remaining bullets.
- void render(); This method is used for drawing gun icon using `DrawingUtility.drawIconGun`.

3.5 Class TargetObject

It is an abstract class which represents a circular object on the game screen. This class is renderable (`IRenderableObject`).

3.5.1 Field

- int x; The position of the object in x-axis
- int y; The position of the object in y-axis
- int z; The position of the object in z-axis which is used to sort its drawing order
- int radius; The radius of the object
- boolean isDestroyed; It is `true` when the object was destroyed.
- int[] movingParameter; It is 8-element array which represents object's movement path.

Index	0	1	2	3	4	5	6	7
Meaning	Path's starting point (x,y)		Path's end point (x,y)		Path's midway point 1 (x,y)		Path's midway point 2 (x,y)	

- int movingDuration; The maximum period of time which the object can remain on the screen
- int movingDurationCounter; The time that the object has been on the screen
- int movingType; It represents type of movement path
  - 0 = Linear path.
  - 1 = Curve path.
  - 2 = Cubic curve path.
- boolean isPointerOver; It is true when the mouse pointer is over.

3.5.2 Constructor

- TargetObject(int radius, int movingDuration, int z) The constructor assigns values to all field. `movingType` and all `movingParameter`'s values is randomly assigned. *[provided]*

3.5.3 Method

- boolean contain(int x, int y); It returns true when (x,y) is in the object's area. *[provided]*

- boolean setPointerOver(boolean isPointerOver); Setter of `isPointerOver` field. *[provided]*
- void move(); The method moves the object based on `movingType` and `movingParameter`. It also destroys the object if the movement has ended. *[provided]*
    - `movingType=0`: Use `MotionUtility.linearMotion` with `movingParameter[0-3]`
    - `movingType=1`: Use `MotionUtility.curveMotion` with `movingParameter[0-5]`
    - `movingType=2`: Use `MotionUtility.cubicCurveMotion` with `movingParameter[0-7]`
- int getZ(); Getter of object's z-position
- boolean isVisible(); It returns `true` when the object is visible (not yet destroyed).
- The method `render` is not needed to be implemented here.

3.6 Class ShootableObject

It is abstract class which represents a shootable object. This class is a subclass of class `TargetObject`.

3.6.1 Field

- int reward; The score of the object which player will receive after the object has been destroyed by player.
- int life; The life point of the object

3.6.2 Constructor

- ShootableObject(int radius, int movingDuration, int z, int reward) It assigns values to fields based on parameters.

3.6.3 Method

- void setLife(int life); It is setter of `life`. The value is capped at zero if it goes below zero. The field `isDestroyed` will be set to `true` if `life` reaches zero.
- void hit(PlayerStatus player); It is called when the object is hit. The life of the object will be decreased by attack rate of current gun that player is using. If the object is destroyed, the player also gets score equals to this object `reward`.

3.7 Class CollectibleObject

It is an abstract class which represents an object that can be collected by holding mouse pointer over it for a specific period. This class is a subclass of class `TargetObject`.

3.7.1 Field

- int requiredGrabbingTime; The period of time (in tick) that player must hold mouse pointer over to collect it.
- int grabbingTimeCount; The period of time (in tick) that player has held mouse over the object

3.7.2 Constructor

- CollectibleObject(int radius, int movingDuration, int z, int requiredGrabbingTime) It assigns values to fields based on parameters.

3.7.3 Method

- void ungrab(); It is called when mouse pointer is out of the object's area. It sets `grabbingTimeCount` to zero.
- void grab(PlayerStatus player); It is called once every tick while mouse pointer is on the object. It increases `grabbingTimeCount` until the value reaches the required time. The item is then collected afterward. *[partially provided]*
- void collect(PlayerStatus player); It is abstract method which is called when the object is collected. The result differs in each type of collected object.

3.8 Class SimpleTarget

It represents a blue object that can be shot. This class is a subclass of class `ShootableObject`.

3.8.1 Constructor

- SimpleTarget(int radius, int movingDuration, int z) It assigns values to fields based on parameters. This object has 3 life points and gives 3 score points if it is destroyed.

3.8.2 Method

- void render(); This method is used for drawing the target using `DrawingUtility.drawShootableObject`. *[provided]*

3.9 Class SmallTarget

It represents a yellow object that can be shot. This class is a subclass of class `ShootableObject`.

3.9.1 Constructor

- SmallTarget(int radius, int movingDuration, int z) It assigns values to fields based on parameters. This object has 2 life points and gives 5 points upon its destruction.
- SmallTarget(int radius, int movingDuration, int z, int startX, int startY) It assigns values to fields based on parameters and also set its `life` and `reward`. Moreover, the `movingType` is set to zero and the `movingParameter`'s values are randomly generated. *[partially provided]*
- void render(); This method is used for drawing the target using `DrawingUtility.drawShootableObject`. *[provided]*

3.10 Class SplitterTarget

It represents a red object that can be shot and splits into multiple small targets after it has been destroyed. This class is a subclass of class `ShootableObject`.

3.10.1 Field

- List<TargetObject> onScreenObject; List of all objects that are currently on the screen. The object reference will be passed to the constructor by `MainLogic` class which instantiating this class' object.

3.10.2 Constructor

- SplitterTarget(int radius, int movingDuration, int z, List<TargetObject> onScreenObject) It assigns values to fields based on parameters. This target has 5 life points and gives 5 points when destroyed. *[partially provided]*

3.10.3 Method

- void hit(PlayerStatus player); It is called when the object is hit. The life of the object will be decreased by attack power of current gun that player is using. If the object is destroyed, the player gets score equals to this object reward. Also, this object randomly creates 3 to 6 small targets of half its size at its position. `movingDuration` and z-position of these small targets are copied from this splitter target directly.
- void render(); This method is used for drawing the target using `DrawingUtility.drawShootableObject`. *[provided]*

3.11 Class ItemSpecialGun

It represents a special gun that can be collected. This class is a subclass of class `CollectibleObject`.

3.11.1 Constructor

- ItemSpecialGun(int movingDuration, int z) It assigns values to fields based on parameters. It sets field `radius` to 50 and `requiredGrabbingTime` to 50 ticks.

3.11.2 Method

- void collect(PlayerStatus player); It is called when player collects this object. It changes current gun of player to special gun with maximum bullets = 20 and attack power = 3.
- void render(); This method is used for drawing the item using `DrawingUtility.drawItemGun`.

3.12 Class ItemBullet

It represents a pack of bullets that can be collected. This class is a subclass of class `CollectibleObject`.

3.12.1 Constructor

- ItemBullet(int movingDuration, int z) It assigns values to fields based on parameters. It sets field `radius` to 50 and `requiredGrabbingTime` to 30.

3.12.2 Method

- void collect(PlayerStatus player); It is called when player collects this object. If player holds special gun, it will add bullets to maximum capacity.
- void render(); This method is used for drawing the item using `DrawingUtility.drawItemBullet`. *[provided]*

3.13 Class MainLogic

It represents a main object that controls the entire game.

3.13.1 Field

- GameBackground background; Background of the game
- PlayerStatus player; Status of the player, also represents the player itself
- List<TargetObject> onScreenObject; List of all objects on the screen
- int zCounter; Z value for newly created object
- int nextObjectCreationDelay; Delay time (in tick) to create next object
- List<GameAnimation> onScreenAnimation; It is related to drawing process.
- boolean readyToRender; It is related to drawing process.

3.13.2 Method

- void onStart(); It is called when game starts. *[provided]*
- void onExit(); It is called when game ends. *[provided]*
- List<IRenderableObject> getSortedRenderableObject(); It returns list of all objects on the screen sorted by z-axis position. *[provided]*
- TargetObject getTopMostTargetAt(int x,int y); It returns target object that is at (x,y) which has the highest z-axis value. If no object is at (x,y), it will return null. *[provided]*
- void createTarget(); students are subject to completing the code. It creates new random object and adds to `onScreenObject`. You MUST use `RandomUtility.random` only once and no other random method in this randomization process. *[partially provided]*
    - 15% : If current gun is simple, it will create `ItemSpecialGun`. If current gun is special, it will create `ItemBullet`.
    - 20% : It creates `SplitterTarget` with radius = 40.
    - 35% : It creates `SmallTarget` with radius = 15.
    - 30% : it creates `SimpleTarget` with radius = 30.
- void logicUpdate(); students are subject to completing the code. It is the method that is used to update status of the game. This method is called once per tick. *[partially provided]*
    - Fill Code1: If player presses “enter” key, `player.isPause` will be toggled.
    - Fill Code2: If player presses “spacebar” key or does “left click”, it will shoot with current gun.
    - Fill Code3: Implement after shooting event. Local variable `target` is the top most object at current mouse pointer position. The action depends on type of `target`. If the type is `CollectibleObject`, it will be grabbed. If the type is

`ShootableObject`, it will be hit if player shoots. Note that local variable `shoot` will be `true` if player shoots during this tick.
- Fill Code4: If the number of bullets reaches zero, the player will be forced to equip basic gun with 1 attack power.

3.14 Class DrawingUtility (Library)

It is in "lab3_lib_0.jar" which is used to draw object.

3.14.1 Method

- void drawItemGun(int x, int y, int radius, String name, boolean isPointerOver); It is used to draw object that player receives gun when collects.
  - "x" is the position of object in x-axis.
  - "y" is the position of object in y-axis.
  - "radius" is the radius of object.
  - "name" is the name of object.
    - Use "gun" for special gun object.
  - "isPointerOver" is `true` if the mouse pointer is over the drawing object.
- void drawIconGun(int bulletQuantity, int maxBullet, boolean isInfiniteBullet); It is used to draw gun icon.
  - "bulletQuantity" is the number of bullets to draw special gun icon. In case of simple gun, any value is acceptable.
  - "maxBullet" is the maximum bullets of special gun. In case of simple gun, any value is acceptable.
  - If "isInfiniteBullet" is `true`, it draws simple gun icon without bullet quantity. Otherwise, it draws special gun icon with specified bullet quantity.
- void drawStatusBar(int remainingSecond, int score, Object gun, boolean pause); It is used to draw status bar at the top of the screen.
  - "remainingSecond" is the remaining time in second.
  - "score" is current score of player.
  - "gun" is current gun that player is using.
  - If "pause" is `true`, it draws the text "PAUSE" at the screen center.

3.15 Class RandomUtility (Library)

It is in "lab3_lib_0.jar" which is used to randomize a number.

3.15.1 Method

- int random(int start, int end); return a random number within range [start,end].

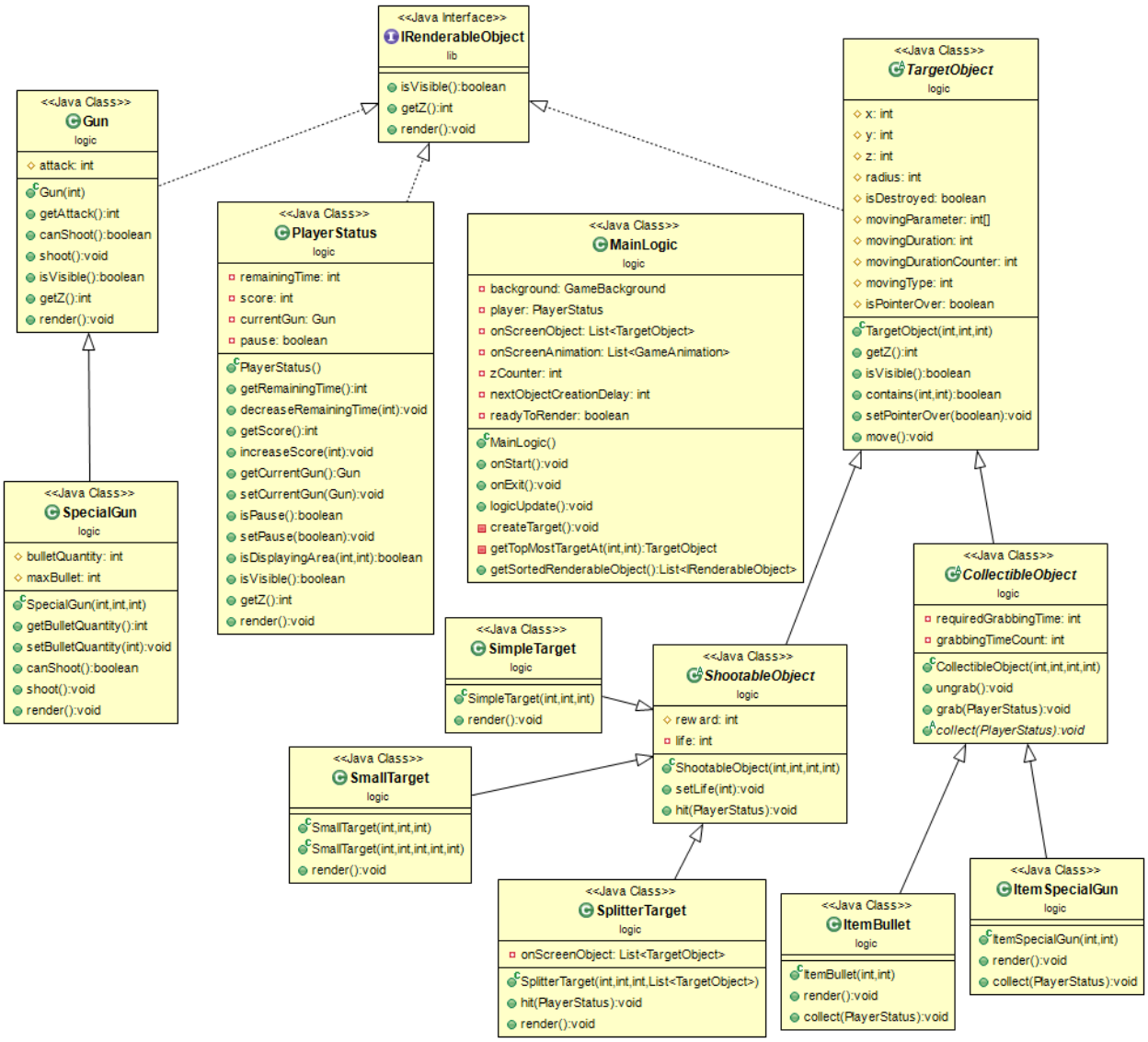

Figure 3. Class diagram of Shooting Game