

## **Tank and Lazarus Game (In Java)**

**a. Student's Name:** Ulises Martinez

**b. Class, Semester:** CSC413, Summer 2018

**c. repositories.**

1. <https://github.com/csc413-01-su18/csc413-tankgame-gasper94>
2. <https://github.com/csc413-01-su18/csc413-secondgame-gasper94>

## **2. Introduction**

### **a. Project Overview:**

For this term project, two games will be implemented using java and principles learned along semester. The first game will be a Tank Game and the second game will be any of the choices given in the TermProject2018 file. The whole idea of this project is to implement a game from scratch and eventually use such code to implement a second game (Lazarus was my choice).

### **b. Introduction of the Tank game (general idea):**

The Tank Game Wars will be a game consisted of two players which will perform and player vs player scenario. This game will have some features as shooting, colliding/destroying with walls, colliding/destroying with tank, pick in up power up (machine gun power up). More specifically, each tank will have to be able to move back and forward and rotate. In terms of game environment, we will have a map which is divided accordingly so each player has its own screen as well a mini-map which both will share. The objective of the game it's for both Tank to try to damage each other until one of them inflicts enough damage to bring each's other live counter to zero.

### **c. Introduction of the Second game (general Idea):**

There will be four different types of boxes, increasing in weight and strength: cardboard, wood, metal, and stone. Falling boxes will come to rest on boxes that are stronger than them, but will crush boxes that are lighter. The type of each box is chosen at random, but the next box will be shown in the bottom-left corner of the window just before it appears. There will be a number of increasingly difficult levels, with higher stairways to build, and boxes that fall faster. The objective of the game its to don't get squished and reach the game checkpoints.

## **3. Development environment.**

**a. Version of Java Used:** Java Version: 9.0.4

**b. IDE Used:** IDE: NetBeans 8.2

**c. Any special libraries used or special resources and where you got them from:**

All resources were obtained from ilearn on behalf of our teacher.

**4. How to build or import your game in the IDE you used.**

**a. Note saying things like hit the play button and/or click import project is not sufficient. You need to explain how to import and/or build the game.**

- First, you'll need to download the app code from Github.

You can do so by using the following commands:

>Using a command line:

1. Navigate to your repository's Code tab
2. Click Clone or Download
3. Copy the URL provided
4. Open your terminal application and enter directory where you would like to copy the repository. For example:

- cd ~/

5. Clone the repository by typing.

- git clone <URL>

OR

Using git (<https://git-scm.com/>)

1. Install program
2. Head to your destination folder and right click on git GUI here
3. Click on Clone Existing Repository.
4. On Source Location: paste your repository URL
5. On Target Directory: Browse for desired location to download files.

Import

---

Import downloaded files to NetBeans

1. Click on File > import project > go to where your files were downloaded and click finish

Note: Make sure Main's EvaluatorUI is been executed. You can right click on your project go to Properties, and then in categories click Run. In Run, select where is says Main Class and select EvaluatorUI as your main so you can see the calculator display.

Optional: In case you would like to run such application without the EvaluatorUI, you can select the class EvaluatorTester in Main Class and insert expressions manually.

**b. List what Commands were ran when building the JAR.**

```
java -jar myJar.jar
```

### **c. List commands needed to run the built jar**

```
java -jar <jar-file-name>.jar
```

## **5. How to run your game. As well as the rules and controls of the game.**

- First you would need to clean and building. You can do so by clicking on the button clean and build jar file on the tool bar.

- Then (depending on your operating system) head to your project's destination folder, open a dist folder and type CTRL + ALT +T to get a command line for that destination folder. Lastly just type the following command line to run your program:

```
java -jar <jar-file-name>.jar
```

## **6. Assumptions Made when designing and implementing Game**

**Tank Game:** Assumptions made when designing and implementing the Tank Game Wars was to complete all of the requirements listed below and make it completely as functional and independent, so you can re-use your code for implementing the second game.

Requirements:

The Tank Wars Game has the following requirements:

1. Tank Game must have 2 Players
2. Tank Game must have tanks that move forwards and backwards
3. Tank Game Must have tanks that rotate so they can move in all directions a. Note when only rotating left or right, the tank **MUST NOT** move forwards or backwards.
4. Tank Game must have a split screen.
5. Tank Game must have a mini-map
6. Tank Game must have health bars for each tank
7. Tank Game must have lives count (how many lives left before game over) for each tank
8. Tank Game must have power up (these are items that can be picked up to modify your tank. What these power ups are, is up to you).

9. Tank game must have unbreakable walls
10. Tank game must have breakable walls
11. Tank Game must have tanks that can shoot bullets that collide with walls
12. Tank Game must have tanks that can shoot bullets that collide with other tanks.

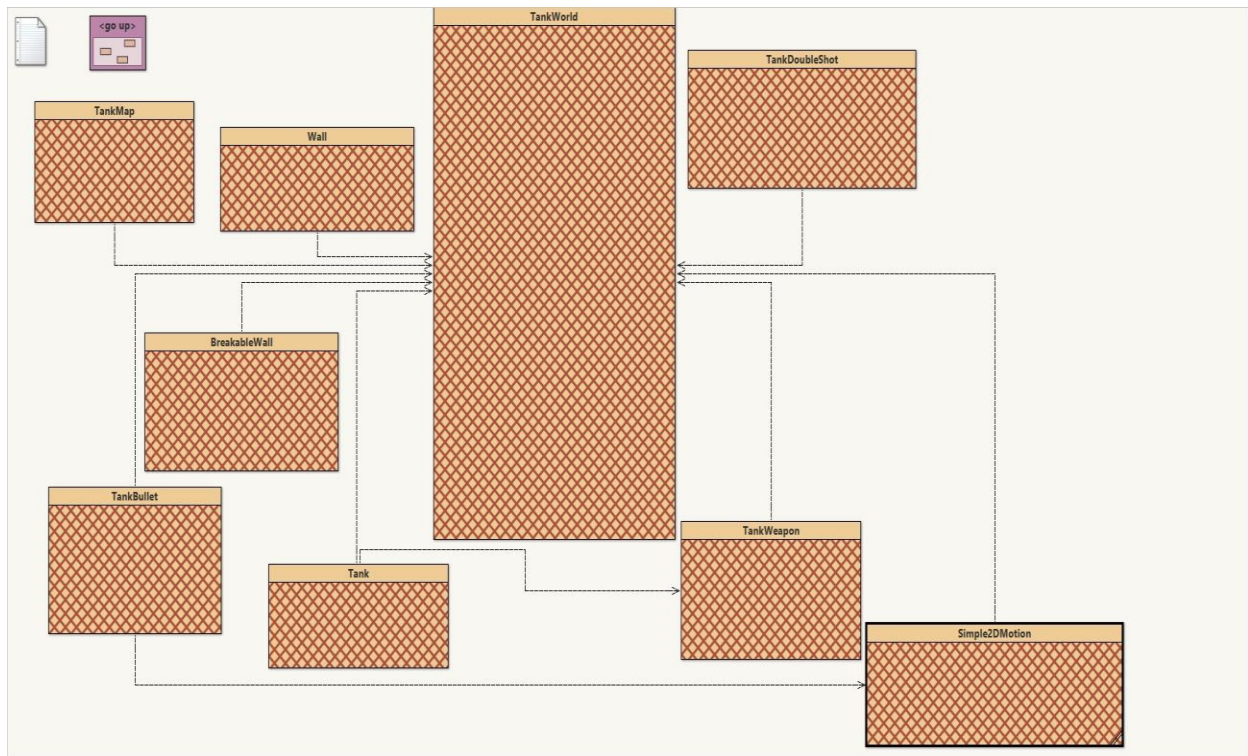
**Lazarus Game:** Assumptions made when designing and implementing the Lazarus Game was to use classes used in the previous game to implement a second game with the idea of “code reusing” to reduce the amount of time and also apply as many principles learned in class.

Requirements:

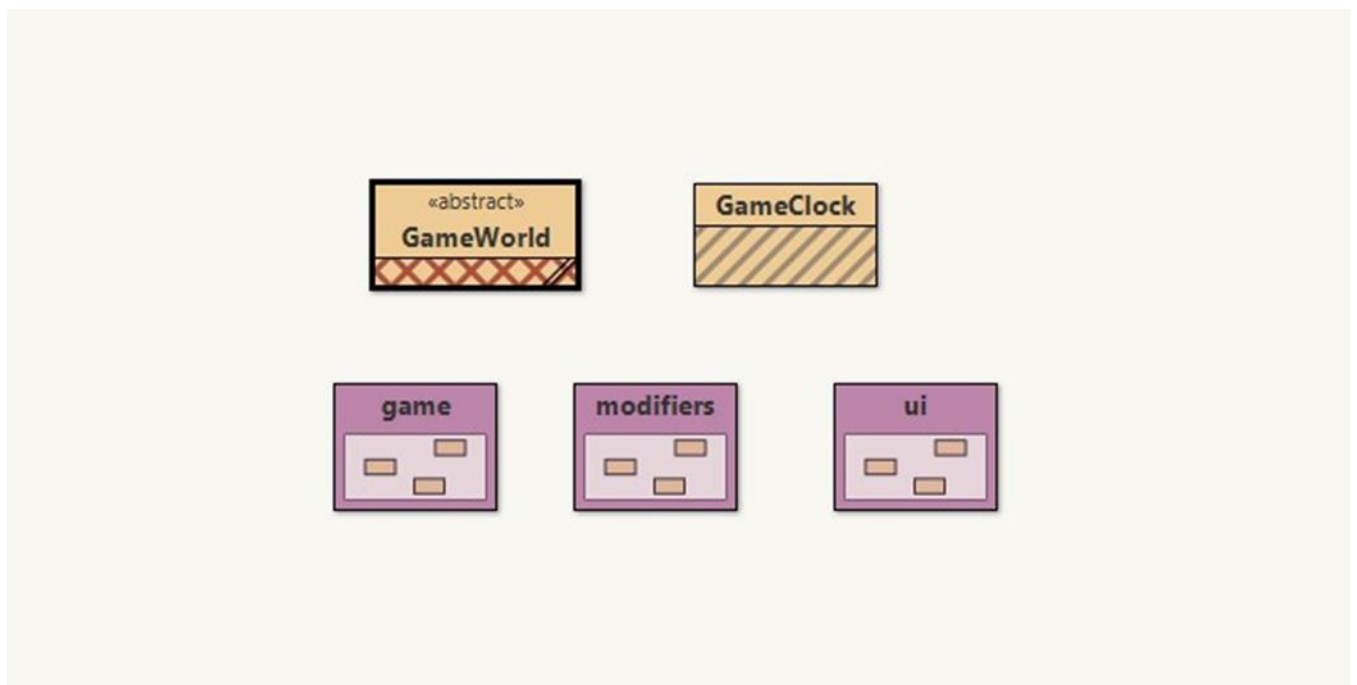
1. Type of boxes: cardboard, wood, metal, and stone
2. Falling boxes will come to rest on boxes that are stronger than them but will crush boxes that are lighter.
3. The type of each box is chosen at random, but the next box will be shown in the bottom-left corner of the window before it appears.
4. There will be several increasingly difficult levels (boxes fall faster)

## 7. Tank Game Class Diagram

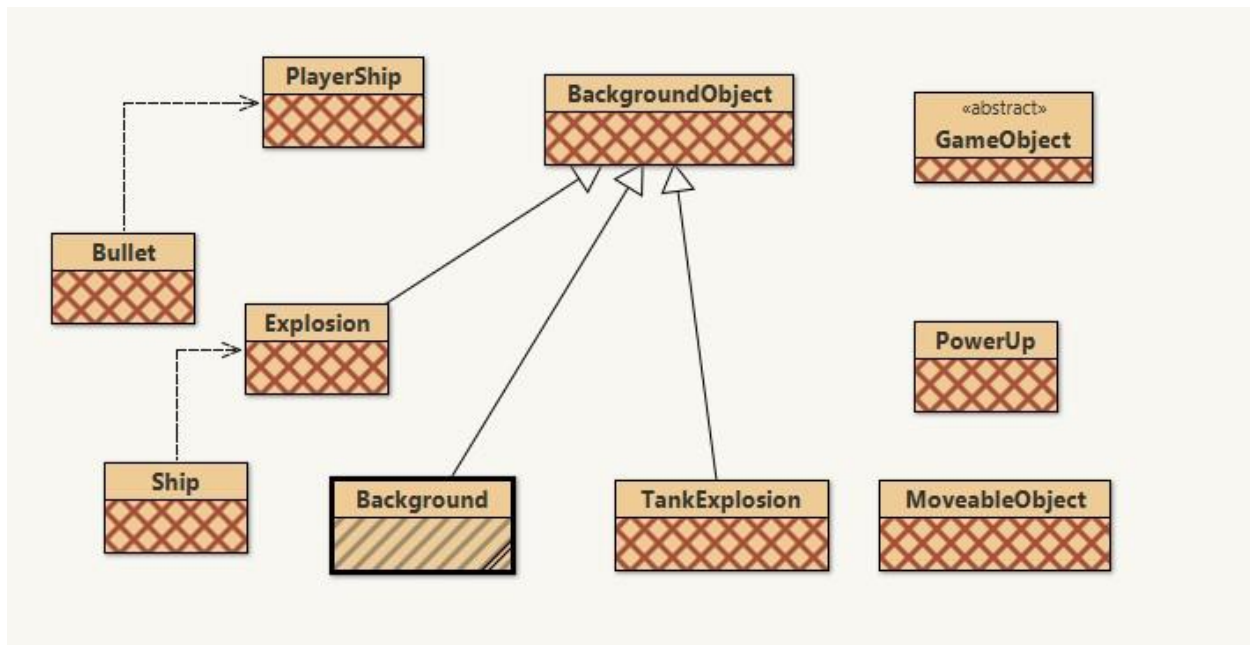
### Tank



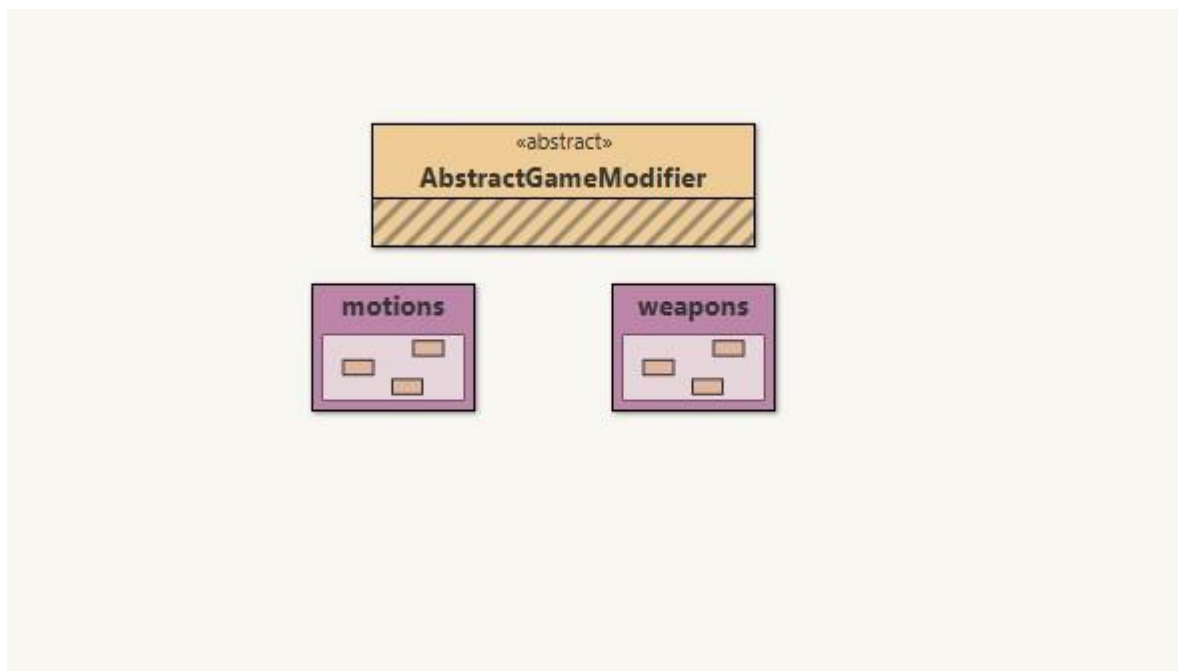
### OtherClasses



## OtherClasses.game

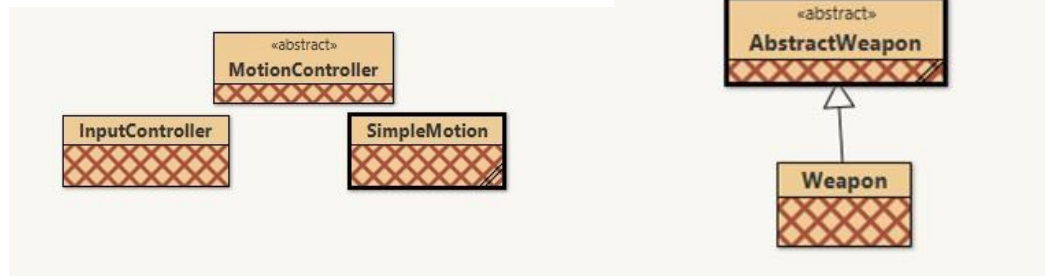


## OtherClasses.modifiers

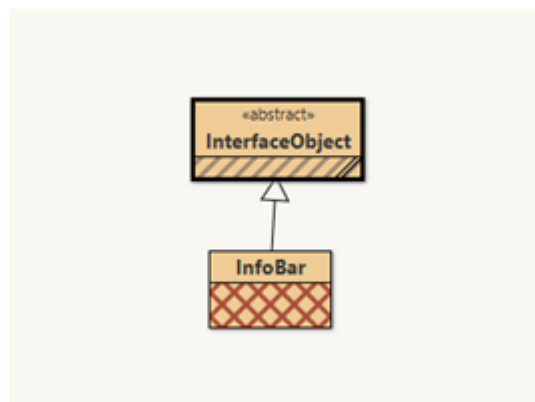


OtherClasses.modifiers.weapon

OtherClasses.modifiers.motion



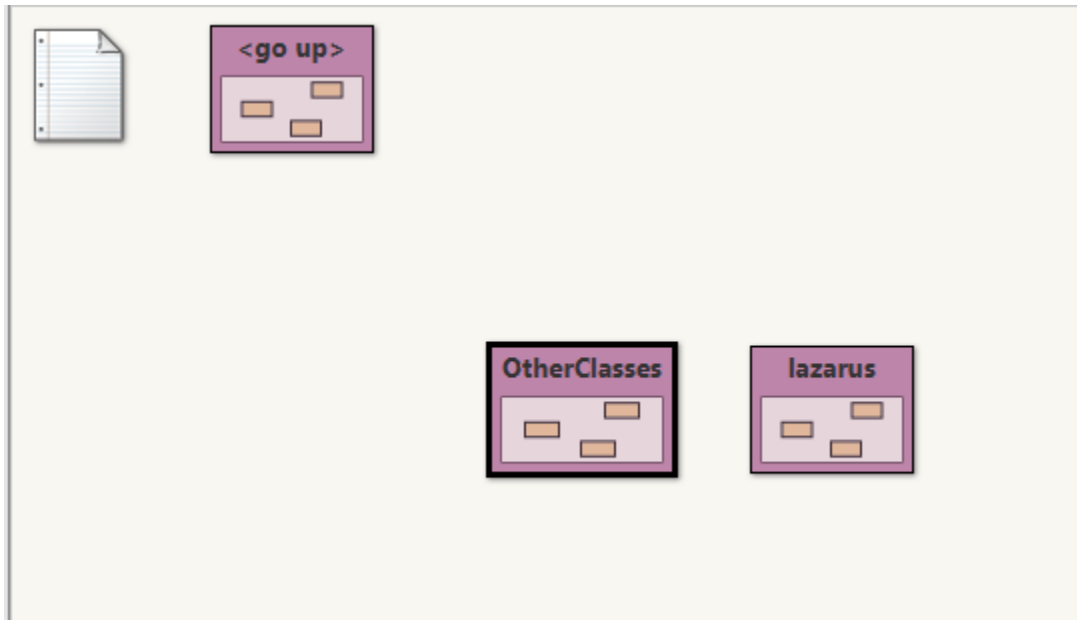
Otherclasses.ui



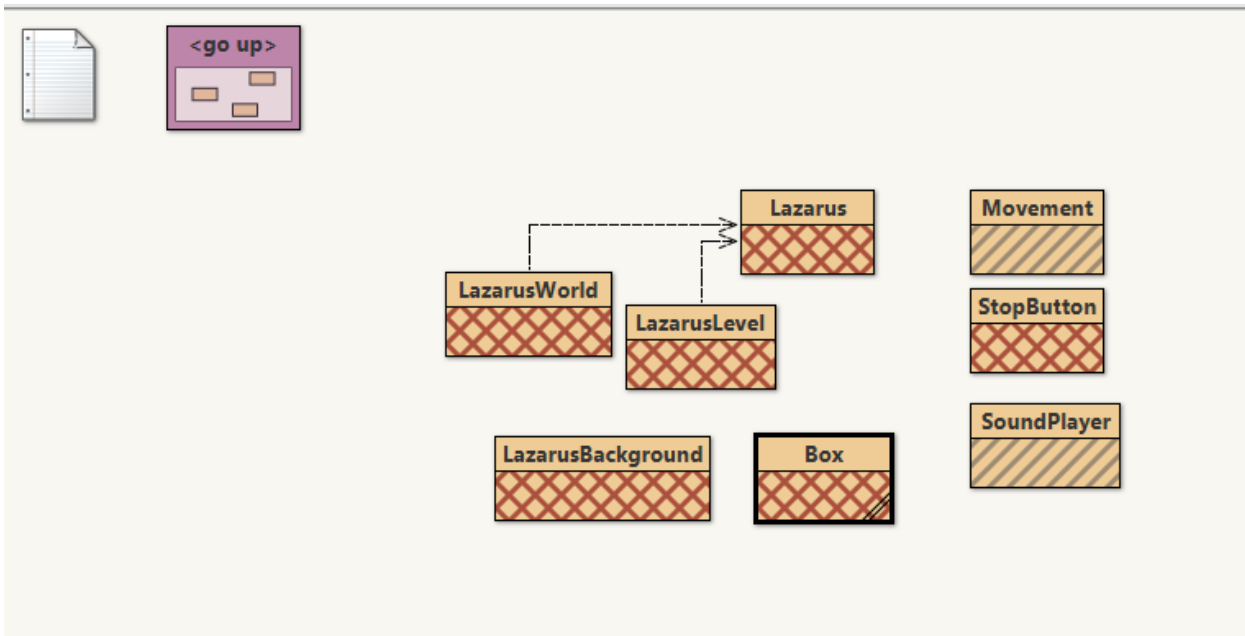


## 8. Second Game Class Diagram

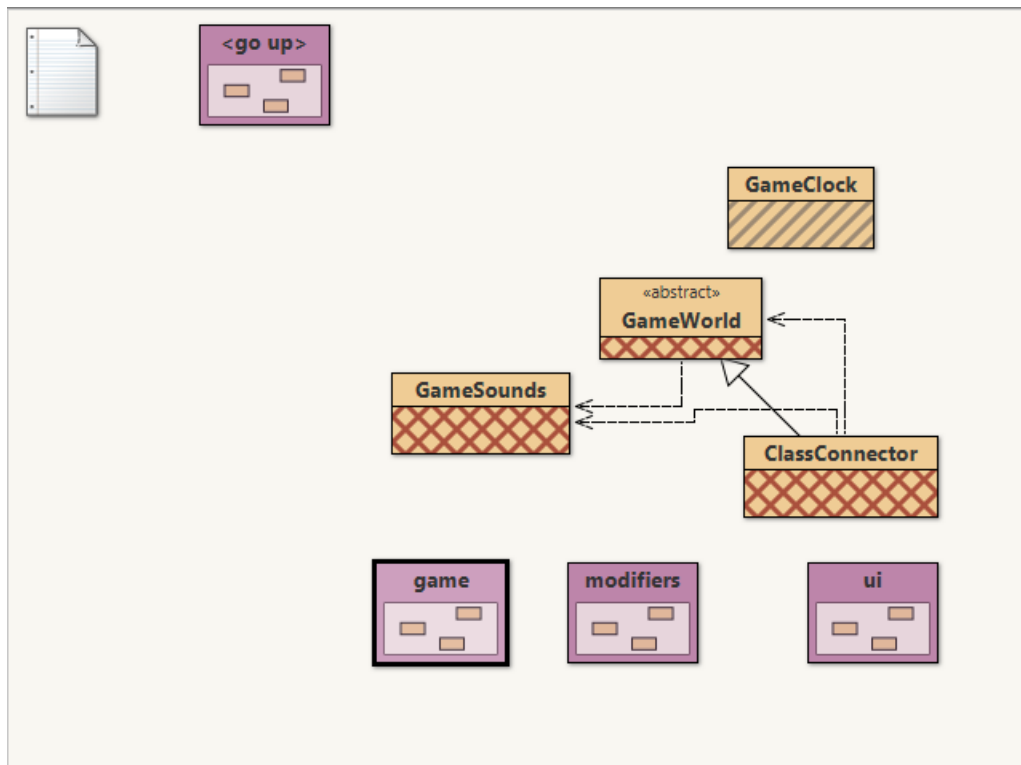
Src



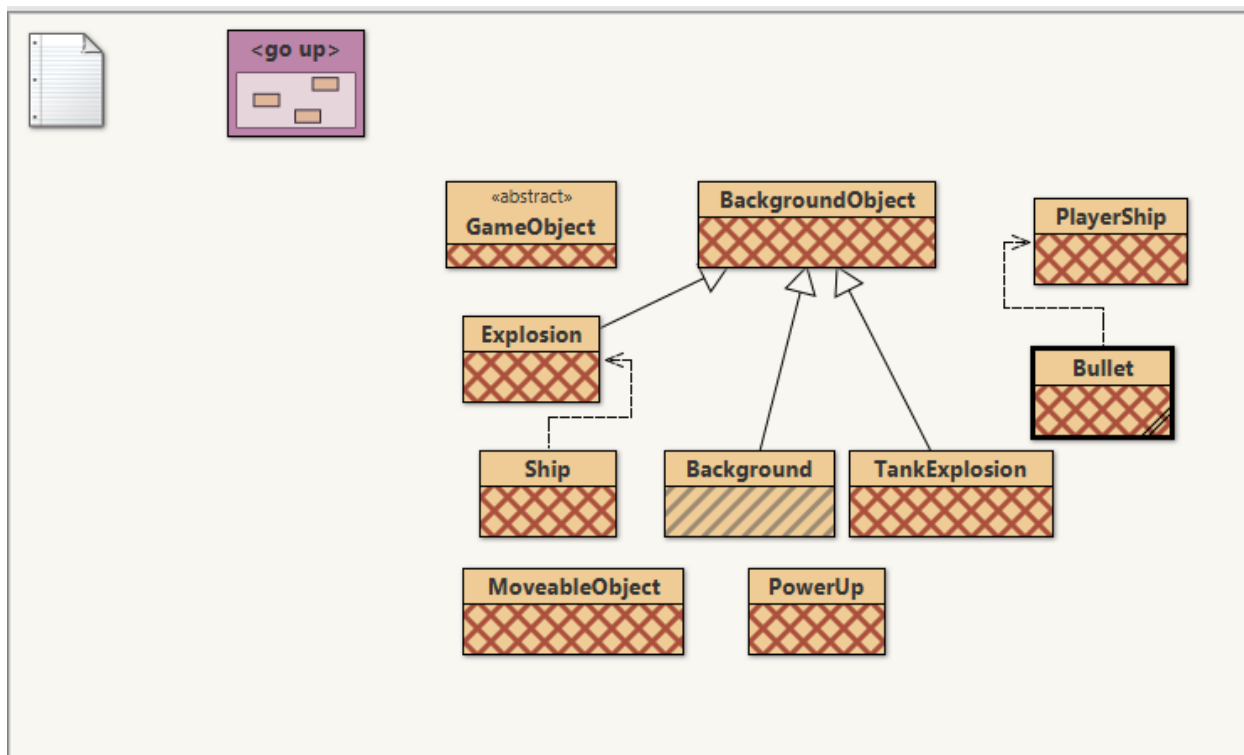
Lazarus



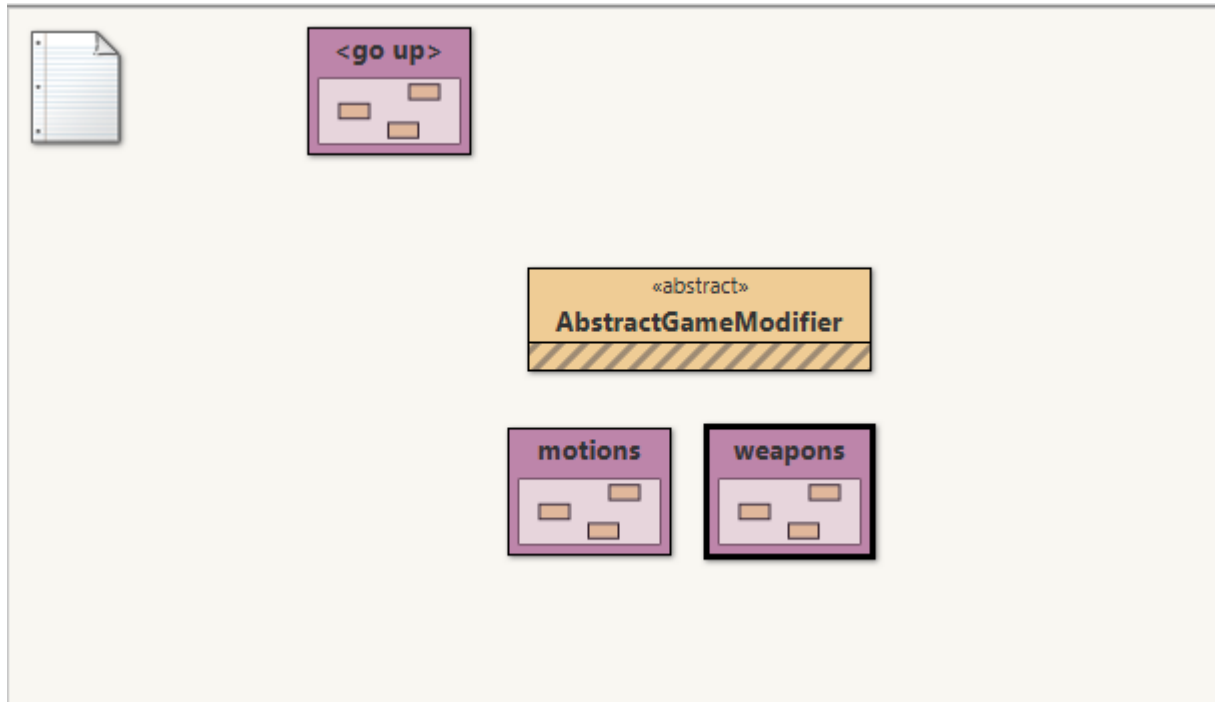
## OtherClasses



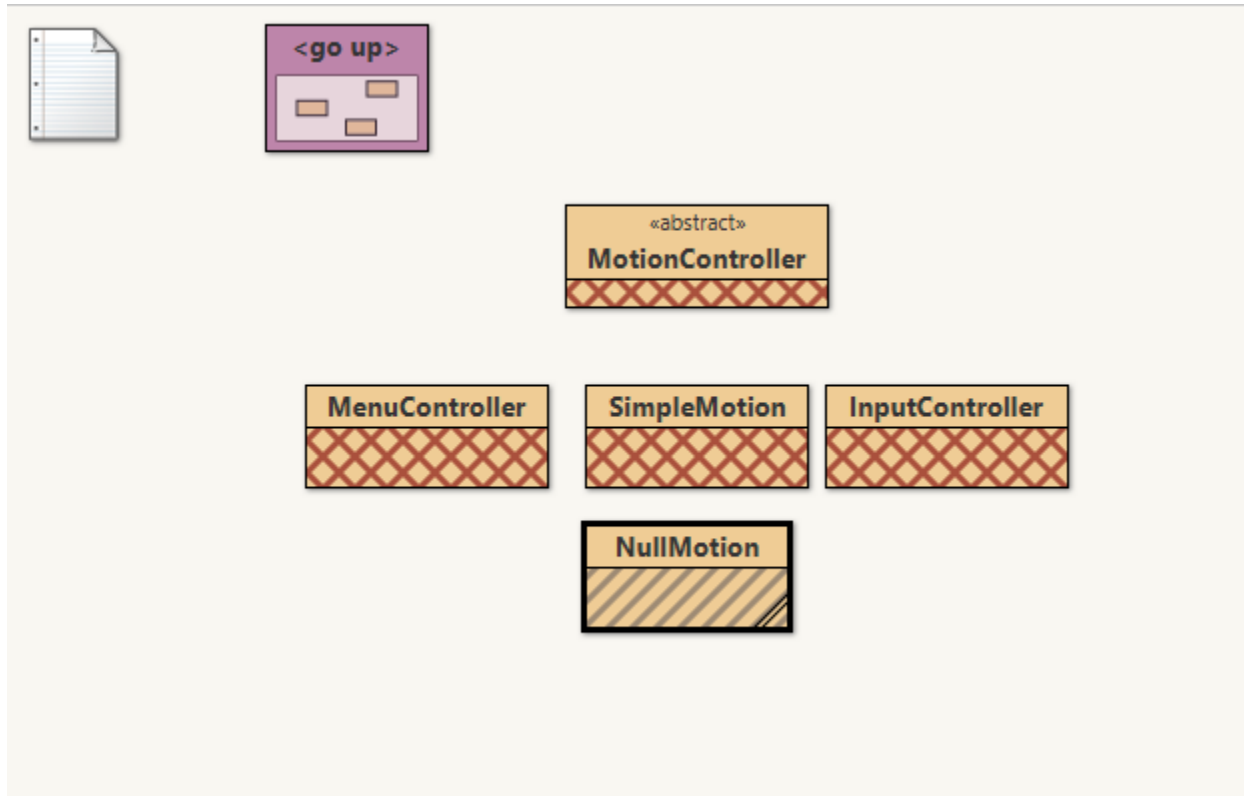
## OtherClasses.game



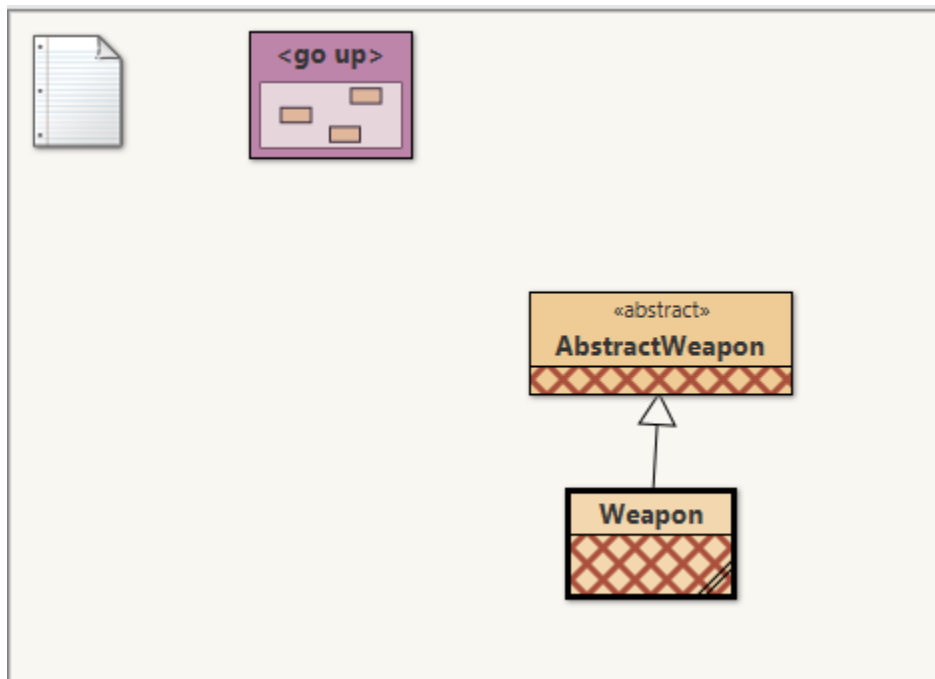
## OtherClasses.modifiers



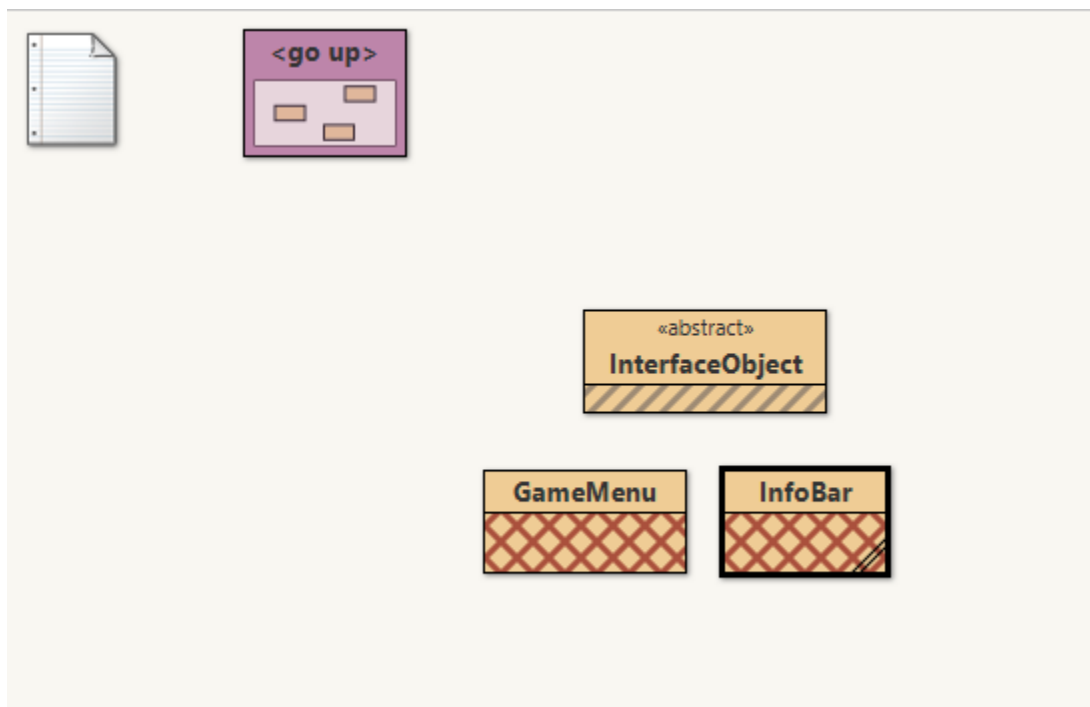
## OtherClasses.Modifiers.motions



OtherClasses.Modifiers.motions



OtherClasses.ui

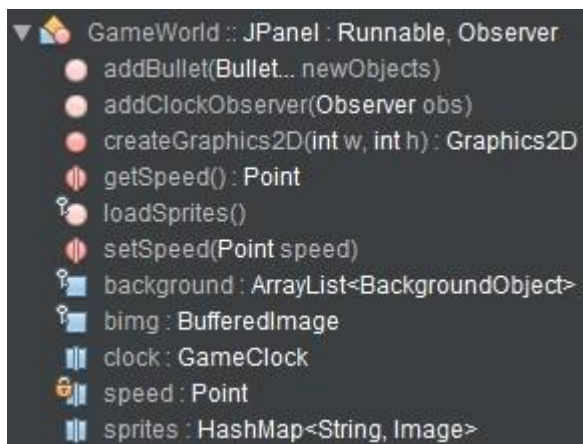


## 9. Class descriptions of all classes shared among both Games

1. **Game Clock:** Game clock ticks on every frame and notifies observers to update. It extends Observable.



2. **Game World:** This class extends JPanel and implements Runnable and Observer. It is in charge of highlighting key aspects of a "Game World".



1. **Background:** This is where the background is drawn. It extends Background Object



2. **Background Object:** Background Objects move at speed of 1 and are not collide able.

```

▼ BackgroundObject :: GameObject
  ♦ BackgroundObject(Point location, Image img)
  ♦ BackgroundObject(Point location, Point speed, Image img)

```

3. **Bullet:** Bullets fired by player one and player two

```

▼ Bullet :: MoveableObject
  ♦ Bullet(Point location, Point speed, int strength, MotionController motion, Gam
  ● getOwner() : PlayerShip
  ● isFriendly() : boolean
  ■ friendly : boolean
  ■ owner : PlayerShip

```

4. **Explosion:** explosions when bullets collide with enemy tank. It extends Background Object.

```

▼ Explosion :: BackgroundObject
  ♦ Explosion(Point location)
  ● collision(GameObject otherObject) : boolean ↑ GameObject
  ● update(int w, int h) ↑ GameObject
  ■ animation : Image[]
  ■ frame : int
  ■ timer : int

```

5. **Game Object:** its an Abstract class which will initialize many variables to create individual objects. Etc wall, tank, and bullets.

```

GameObject :: Observer
  GameObject()
  GameObject(Point location, Point speed, Image img)
  GameObject(Point location, Image img)
  collision(GameObject otherObject) : boolean
  draw(Graphics g, ImageObserver obs)
  getLocation() : Rectangle
  getLocationPoint() : Point
  getSizeX() : int
  getSizeY() : int
  getSpeed() : Point
  getX() : int
  getY() : int
  hide()
  move(int dx, int dy)
  move()
  setImage(Image img)
  setLocation(Point newLocation)
  show()
  update(int w, int h)
  update(Observable o, Object arg)
  height : int
  img : Image
  location : Rectangle
  observer : ImageObserver
  show : boolean
  speed : Point
  width : int

```

6. **PlayerShip**: It a class which extends ship and observer and it creates features of a player's controlled object. (ex: plane, train, Lazarus, tank)

```

▼ PlayerShip :: Ship : Observer
  ◆ PlayerShip(Point location, Point speed, Image img, int[] controls, String name)
  ● damage(int damageDone) ↑ Ship
  ● die() ↑ Ship
  ● draw(Graphics g, ImageObserver observer) ↑ GameObject
  ● fire() ↑ Ship
  ● getLives() : int
  ● getName() : String
  ● getScore() : int
  ● incrementScore(int increment)
  ● isDead() : boolean
  ● reset()
  ● startFiring()
  ● stopFiring()
  ● update(int w, int h) ↑ MoveableObject
  ● update(Observable o, Object arg) ↑ GameObject
  ■ down : int
  ? isFiring : boolean
  ? lastFired : int
  ■ left : int
  ? lives : int
  ? name : String
  ? resetPoint : Point
  ■ respawnCounter : int
  ■ right : int
  ? score : int
  ■ up : int

```

7. **Power Up:** power up it's a class that extends Ship and it creates objects that can be accessible to tanks. (ex: weapon, health and other power ups)

```

▼ PowerUp :: Ship
  ◆ PowerUp(Ship theShip)
  ◆ PowerUp(int location, int health, AbstractWeapon weapon)
  ◆ PowerUp(Point location, int health, AbstractWeapon weapon)
  ● die() ↑ Ship
  ● update(Observable o, Object arg) ↑ GameObject

```

8. **Ship:** The class ship extends Moveable Objects. It will be in charge of dealing with “game interactive objects” such as weapons, health bar, and gun location.



```

Ship :: MoveableObject
  Ship(Point location, Point speed, int strength, Image img)
  Ship(int x, Point speed, int strength, Image img)
  collide(GameObject otherObject)
  damage(int damageDone)
  die()
  fire()
  getGunLocation() : Point
  getHealth() : int
  getMotion() : MotionController
  getWeapon() : AbstractWeapon
  setHealth(int health)
  setMotion(MotionController motion)
  setWeapon(AbstractWeapon weapon)
  gunLocation : Point
  health : int
  weapon : AbstractWeapon

```

9. **Tank Explosion:** The Tank explosion class extends Background Object and its in charge of detecting collision between a tank and a bullet. It also its in charge of the animation that comes after destroying a tank.

```

TankExplosion :: BackgroundObject
  TankExplosion(Point location)
  collision(GameObject otherObject) : boolean ↑ GameObject
  update(int w, int h) ↑ GameObject
  animation : Image[]
  frame : int
  timer : int

```

---

1. **Abstract Game Modifier:** This class extends Observable and it's the root of modifiers such as weapons, movements, player input, and other events.

```

AbstractGameModifier :: Observable
  AbstractGameModifier()
  read(Object theObject)

```

---

1. **Input Controller:** This class extends Motion Controller and implements Key listener. Assign buttons to in-game actions.

```

InputController :: MotionController : KeyListener
  InputController(PlayerShip player, int[] keys, Component world)
  clearChanged() ↑ Observable
  keyPressed(KeyEvent e)
  keyReleased(KeyEvent e)
  keyTyped(KeyEvent e)
  read(Object theObject) ↑ MotionController
  setFire()
  setMove(String direction)
  signalKeyPress(KeyEvent e)
  unsetFire()
  unsetMove(String direction)
  update(Observable o, Object arg) ↑ MotionController
  action : Method
  field : Field
  keys : int[]
  moveState : int
  player : boolean

```

2. **Motion Controller:** This class extend Abstract Game Modifier and implements Observer. Its in charged on moving around objects.

```

MotionController :: AbstractGameModifier : Observer
  MotionController()
  MotionController(GameWorld world)
  delete(Observer theObject)
  read(Object theObject) ↑ AbstractGameModifier
  update(Observable o, Object arg)
  fireInterval : int

```

3. **Simple Motion:** Simple motion extends Motion Controller and it receives a object of the type Moveable Object then executes such movement.

```

SimpleMotion :: MotionController
  SimpleMotion()
  read(Object theObject) ↑ MotionController

```

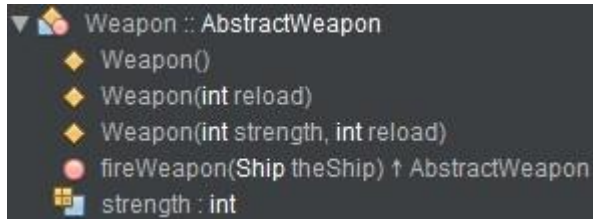
1. **Abstract Weapon:** This Class extends Abstract Game Modifier and highlights key features of weapons in the game.

```

AbstractWeapon :: AbstractGameModifier
  AbstractWeapon()
  AbstractWeapon(Observer world)
  fireWeapon(Ship theShip)
  read(Object theObject) ↑ AbstractGameModifier
  remove()
  bullets : Bullet[]
  direction : int
  friendly : boolean
  lastFired : int
  reload : int
  reloadTime : int

```

2. **Weapon:** This class extends Abstract Weapon and it implements the idea of firing a weapon. (location of such weapon, speed & direction as well as the bullets)



```
▼ Weapon :: AbstractWeapon
  Weapon()
  Weapon(int reload)
  Weapon(int strength, int reload)
  fireWeapon(Ship theShip) ↑ AbstractWeapon
  strength : int
```

---

1. **InfoBar:** it extends Interface Object and its in charged of creating the interface for health bar, so user can interact with the tank health.



```
▼ InfoBar :: InterfaceObject
  InfoBar(PlayerShip player, String name)
  draw(Graphics g2, int x, int y) ↑ InterfaceObject
  name : String
  player : PlayerShip
```

2. **Interface Object:** It implements Observer and its in charge of featuring key aspects of interface, so user can interact.

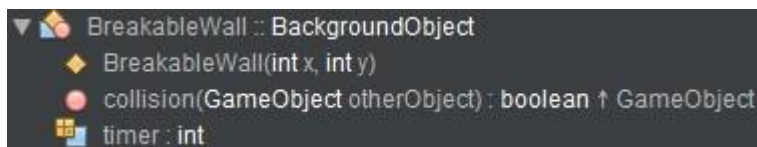


```
▼ InterfaceObject :: Observer
  draw(Graphics g, int x, int y)
  update(Observable o, Object arg)
  location : Point
  observer : ImageObserver
```

---

## 10. Class Descriptions of classes specific to Tank Game

1. **Breakable Wall:** The Breakable Wall class extends Background Object which implement the object wall that disappear when a bullet collides with them.



```
▼ BreakableWall :: BackgroundObject
  BreakableWall(int x, int y)
  collision(GameObject otherObject) : boolean ↑ GameObject
  timer : int
```

2. **Simple2DMotion:** The Simple2DMotion class extends Motion Controller and highlights the moving features for the tank objects. (Ex: moving back/forward and rotating.

```

Simple2DMotion :: MotionController
  Simple2DMotion(int direction)
  read(Object theObject) ↑ MotionController
  dx : int
  dy : int

```

3. **Tank:** The class Tank extends PlayerShip which assign specific features of a tank creating such movable object.

```

Tank :: PlayerShip
  Tank(Point location, Image img, int[] controls, String name)
  die() ↑ PlayerShip
  draw(Graphics g, ImageObserver obs) ↑ PlayerShip
  reset() ↑ PlayerShip
  turn(int angle)
  update(int w, int h) ↑ PlayerShip
  direction : int

```

4. **Tank Bullet:** The class Tank Bullet extends bullet assign specific features of a bullet that its made for a tank. Also contains the method Draw() which projects such on the screen.

```

TankBullet :: Bullet
  TankBullet(Point location, Point speed, int strength, Tank owner)
  TankBullet(Point location, Point speed, int strength, int offset, Tank owner)
  draw(Graphics g, ImageObserver obs) ↑ GameObject

```

5. **Tank Double Shot:** Tank Double Shot extends Abstract Weapon and has a new feature where two instances of bullet are created using the same logic as the tank Bullet class but the reloading time variates.

```

TankDoubleShot :: AbstractWeapon
  TankDoubleShot()
  fireWeapon(Ship theTank) ↑ AbstractWeapon

```

6. **Tank Map:** The class Tank Map extends Abstract Game Modifiers and implements Observer which Maps location of Tanks, power up, walls and unbreakable walls.

```

▼ TankMap :: AbstractGameModifier : Observer
  ♦ TankMap(String filename)
  ● load()
  ● read(Object theObject) ↑ AbstractGameModifier
  ● update(Observable o, Object arg)
  ■ endgameDelay : int
  ■ filename : String
  ■ h : int
  ■ level : BufferedReader
  ■ position : Integer
  ■ start : int
  ■ w : int

```

7. **Tank Weapon:** This class extends Abstract Weapon and its in charge the location of the weapon in relation to the tank.

```

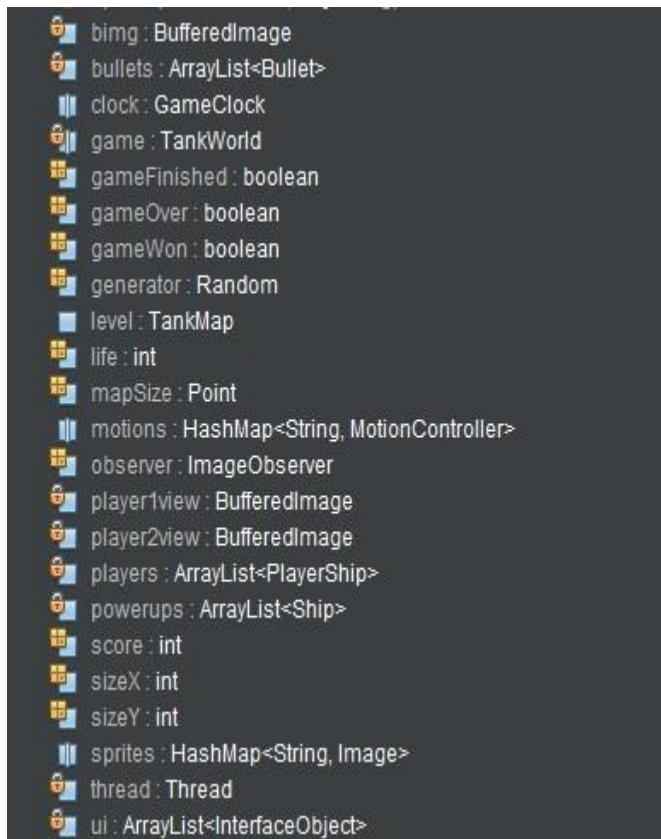
▼ TankWeapon :: AbstractWeapon
  ♦ TankWeapon()
  ● fireWeapon(Ship theTank) ↑ AbstractWeapon

```

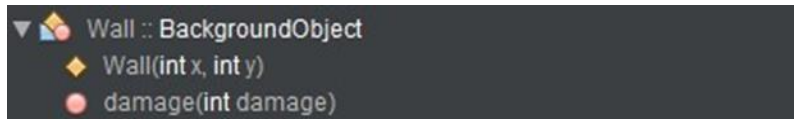
8. **Tank World:** it's the main class and extends GameWorld. It has all the game logic and initialization objects.

▼ TankWorld :: GameWorld

- TankWorld()
- addBackground(BackgroundObject... newObjects)
- addBullet(Bullet... newObjects) ↑ GameWorld
- addClockObserver(Observer theObject) ↑ GameWorld
- addPlayer(PlayerShip... newObjects)
- addPowerUp(Ship powerup)
- countPlayers() : int
- createGraphics2D(int w, int h) : Graphics2D ↑ GameWorld
- drawFrame(int w, int h, Graphics2D g2)
- endGame(boolean win)
- finishGame()
- getBackgroundObjects() : ListIterator<BackgroundObject>
- getBullets() : ListIterator<Bullet>
- getFrameNumber() : int
- getInstance() : TankWorld
- getPlayers() : ListIterator<PlayerShip>
- getSprite(String name) : Image
- getTime() : int
- init()
- isGameOver() : boolean
- loadSprites() ↑ GameWorld
- main(String[] argv)
- paint(Graphics g) ↑ JComponent
- removeClockObserver(Observer theObject)
- run()
- setDimensions(int w, int h)
- start()
- update(Observable o, Object arg)

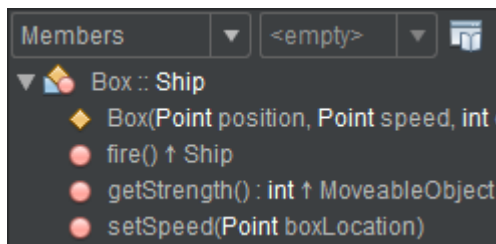


9. **Wall:** The class Wall extends Background Object and it describe unbreakable walls. The difference its that this wall have no damage to contact with bullets.



## 11. Class Descriptions of classes specific to Second Game

1. **Box:** It extends the ship Class and deals with speed of boxes and strength.



2. **Lazarus:** The Class Lazarus extends PlayerShip and describes Lazarus behavior such as Box, Stop, and wall Collision.



```

▼ Lazarus :: PlayerShip
  ◆ Lazarus(Point location, Image img, int[] controls, String name)
  ● die() ↑ PlayerShip
  ● isPlayerBoxCollision() : boolean
  ● isPlayerStopCollision() : boolean
  ● isPlayerWallCollision() : boolean
  ● update(int w, int h) ↑ PlayerShip
  ● oldLeft : int
  ● oldRight : int

```

3. **Lazarus Background:** The class Lazarus World will extend Game World and will be in charge of implementing surrounding behaviors such Fallen Boxes, 2D movement, and Wall boxes.

```

▼ LazarusBackground :: BackgroundObject
  ◆ LazarusBackground(int w, int h, Point speed, Image img)
  ● collision(GameObject otherObject) : boolean ↑ GameObject
  ● update(int w, int h) ↑ GameObject
  ● h : int
  ● w : int

```

4. **Lazarus Level:** This Class extends Abstract Game Modifiers and implements Observer when it will basically map the level (ex: Lazarus, level intensity(walls) and “stops” exits.


```


▼ LazarusLevel :: AbstractGameModifier : Observer
  ◆ LazarusLevel(String filename)
  ● getRandomBox(int x, int y) : Box
  ● load()
  ● read(Object theObject) ↑ AbstractGameModifier
  ● update(Observable o, Object arg)
  ● currentBox : Box
  ● filename : String
  ● h : int
  ● level : BufferedReader
  ● nextBox : Box
  ● w : int






























```

5. **Lazarus World:** it's the main class and extends Game World. It has all the game logic and initialization objects.



▼  LazarusWorld :: GameWorld

 LazarusWorld()

-  addBackground(BackgroundObject... newObjects)
-  addBckgrd(BackgroundObject... objects)
-  addBckgrd(Box... aBox)
-  addBullet(Bullet... newObjects) ↑ GameWorld
-  addClockObserver(Observer object) ↑ GameWorld
-  addFallenBox(int y, Box... fallenBox)
-  addNewPlayer(PlayerShip... objects)
-  createGraphics2D(int w, int h) : Graphics2D ↑ GameWorld
-  drawExplosions()
-  drawFallingBox(int w, int h)
-  drawNewBox()
-  drawStopButtons()
-  drawWallBoxes()
-  findBoxAtY(int y) : ListIterator<Box>
-  getBoxesComingDown() : ListIterator<Box>
-  getLazarus() : LazarusWorld
-  getObjs() : ListIterator<BackgroundObject>
-  getPlayer() : ListIterator<PlayerShip>
-  getSprite(String name) : Image ↑ GameWorld
-  getWallItems() : ListIterator<Box>
-  init()
-  loadSprites() ↑ GameWorld
-  main(String[] argv)
-  numberOfFallenBoxes(int y, int x) : int
-  paint(Graphics g) ↑ JComponent
-  run()
-  setNewDimensions(int w, int h)
-  start()
-  update(Observable object, Object arg)

```

bimg : BufferedImage
boxArray : ArrayList<Box>
boxesFalling : ArrayList<Box>
clock : GameClock
explosions : ArrayList<Movement>
fallenBoxes : ArrayList<ArrayList<Box>>
g2 : Graphics2D
gameEnded : boolean
gameOver : boolean
level : LazarusLevel
mapSize : Point
menu : GameMenu
myGame : LazarusWorld
observer : ImageObserver
players : ArrayList<PlayerShip>
sizeX : int
sizeY : int
sleepNumber : int
sp : SoundPlayer
sprites : HashMap<String, Image>
squished : Image[]
stopButtons : ArrayList<StopButton>
thread : Thread
wallArray : ArrayList<Box>
winner : boolean

```

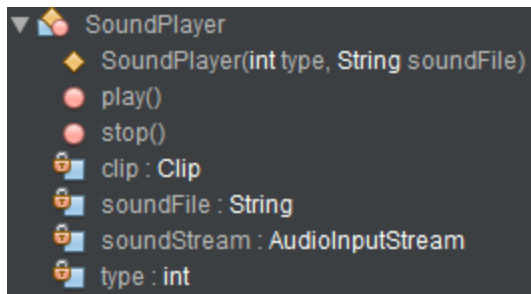
6. **Movement:** This class will be in charged of printing the box picture and updated once it has finished (touched ground).

```

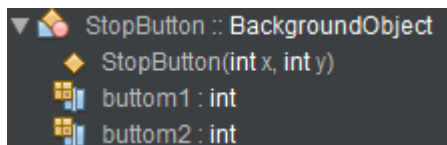
Movement
Movement(int x, int y, Image[] img)
draw(Graphics g, ImageObserver obs)
getFinished() : boolean
update()
count : int
finished : boolean
img : Image[]
type : int
x : int
y : int

```

7. **Sound Player:** This Class will be in charge of in-game sounds related to background music and Lazarus interaction to boxes and other behaviors.



**8. Stop Button:** The Class Stop Button will implement extends Background Object and will stop boxes at bottom limit, so they stay in place.



## 12. Self-reflection on Development process during the term project

This term project was really challenging, First I didn't know where to get started but with some tips from my professor I started to get the hang of it. I would say that as the project progressed the project started to feel more and more difficult. I would say the most helpful thing that I did was read the pdf on ilearn about Strategy pattern, because it made me feel that little by little I could accomplish the task. It might seem natural to write code but for this project I feel a bit lost. The Strategy Pattern it's the way to go. Later, after finishing the first game the second game became a bit easier time wise. There were many repetitive things about images, collision, movement. I also learn that if you do preparation before coding can be a time saver and your code will be ten times cleaner without many classes and coupling.

## 13. Project Conclusion.

This project made me realize what does Object-Oriented programming really is. Until now, I only did project that had a maximum of three classes. I have realized how big a project can get and the things I should do to make it better. I feel eager to do more projects like this. It is interesting how my programming world got even bigger taking this course. And as always, I wish I had more time to make improvements or do certain things in a certain manner. All I can say I'm ready to learn whatever its next and I'll much more effort to it.