OO Games

Authors:

- **Jason Cromer**
- **Ilya Nemtsov**

Introduction

The purpose of this assignment was to create object-oriented solutions for the **Tank War** and **KoalaBR8** games. The professional resources for both games were provided by Instructor; moreover, the example of games have been brought to life with graphics and illustrations. Both games have been developed, designed and written in java language. To complete both games, we used IDE IntelliJ IDEA 2016.3.4. Also, the games were tested on Windows and Mac operating systems with JDK version "1.8.0_121". The code was developed and implemented in such way that anyone can optimize and advance the current version Tank and Koala games.

Scope of work:

Objectives/What was completed:

We were challenged with the following requirements below, and they were successfully completed:

- ✓ create an object-oriented solution for a Tank game
  - follow the Tank game description
- ✓ clean up and reuse the Tank game's classes while creating the OO solution for Koala game
  - follow the Koala game description
- ✓ learn and use GUI
- ✓ learn and use java graphics
- ✓ use different OO design patterns
- ✓ work in a team
- ✓ follow given milestone requirements

Games' description

Tank Game

Tank game is a game for two players. Each player drives his or her own tank through the walled battle arena with the aim of obliterating the other's tank. Once a tank is destroyed, both tanks are respawned at their start position, and a point is awarded to the surviving player. Most walls provide permanent cover, but some can be temporarily demolished to create a way through. There is no ultimate goal to the game, and players simply play until one player concedes defeat. Each tank has a primary weapon that it can fire indefinitely. Pickups provide a repair some of the tank's damage. The tank game screen is divided into two sections: the left-hand side, which renders the area of the map immediately around player one, and the right-hand side, which renders the area of the map immediately around player two. The map is composed of a background tile, onto which various obstructions may be placed. As mentioned above there are two types of walls - indestructible wall that surround the map, creating a boundary past which tanks may not travel, and destructible wall that may be destroyed using the tank weapons. The game must contain a mini map which is rendered in the lower portion of the screen that shows the entire map, zoomed out, with markers for walls and the player locations. Both players will play from the same computer. Player one will move his/her tank with the A, D, W, and S keys and fire with the spacebar. Player two will control his/her tank with the arrow keys, and fire with the Enter key. Moreover, the tanks must be smoothly rendered as they move across the map. Animations is rendered where appropriate - explosions, bullet paths, etc. Audio is also used for game events such as explosions, tank firing, etc.

Koala Game

According to the book The Game Maker's Apprentice, the focus of the Koala game will be on puzzles rather than action. A colony of koala bears has been captured by the evil Dr. Bruce for use in his abominable experiments. The koalas manage to escape from their cages only to find that the doctor has implanted some kind of mind control device in their brains. The only way they can overpower the controlling effect is to combine their thoughts and all perform the same actions at once. The koalas must work together to find their way past the many dangers in the doctor's laboratory and escape to freedom. The arrow keys will simultaneously move all of the bears on a level, except bears whose paths are blocked by a wall or another bear. Each level will be a hazardous maze that is completed by getting all of the koalas to an exit. However, if a koala touches a dangerous hazard on the way, then he dies and the level must be replayed. The game will contain many of fatal and nonfatal hazards shown in the following feature list:

- Fatal hazards
  - Explosive TNT
  - Moving circular saws
- Nonfatal hazards
  - Red exits—Allow any number of koalas to exit the level
  - Blue exits—Allow a single koala to exit the level
  - Locks—Block the path of koalas (red, blue, and green)
  - Switches—Open locked passageways (normal, timed, and pressure)
  - Boulders—Can be pushed by koalas and destroy other hazards

Instruction to compile and execute using Terminal/Command Prompt:

Assumptions:

Input/output:

- Tank and Koala games read the text files (contained numbers and letters) that help to create a level map
- Once the text file has been read correctly, the game should start immediately with graphical representation of the map
- If the text file can't be read the game shouldn't start
  - It may notify a user about the problem
- Players input involves using keyboard keys

Possible features of Tank game:

- Tanks can move simultaneously
- Tanks can rotate 360 degrees
- Tanks can shoot bullets
- Tanks can't move through the walls

- Walls can be destructible and indestructible
- Game records score, lives and health damage for each tank
- Sound plays on background and during special game events such as explosion, shooting etc.

Possible features of Koala game:

- Koala must move simultaneously
- Koalas can only rotate 90 degrees
- Koalas can't move through the walls
- Koalas can move some objects: rock
- Game records saved Koalas
- Sound plays on background and during special game events such as TNT explosion, colliding saws, moving rocks etc.
- Menu with restart and quit buttons

Possible bugs in Tank game:

- Tank VS Tank and Tank VS Wall collision detection doesn't always work properly
- Simultaneous movements are not functioning correctly

Possible bugs in Koala game:

- Not all collision cases are considered
- Not all obstacles have the right starting positions

Possible bugs in Koala and Tank game implementation:

- Many inefficiencies
- Repetition of code
- Not meaningful variable names
- Indentation/spacing problems
- Many dependencies
- Classes are not design according to OOP
- Methods are bulky

Assumptions based on provided games' description:

- Project can be finished under provided requirements
- Classes should be reusable as much as possible

<u>Code organization (packages and classes exist within our project):</u>

Package game	Function of Package
GameBackground.java	This package contains the reusable classes that needed for creating Tank and Koala games
GameKeyAdapter.java	
GameboardPanel.java	
GameKeyEvent.java	
GameObject.java	
GameSounds.java	
LevelParser.java	

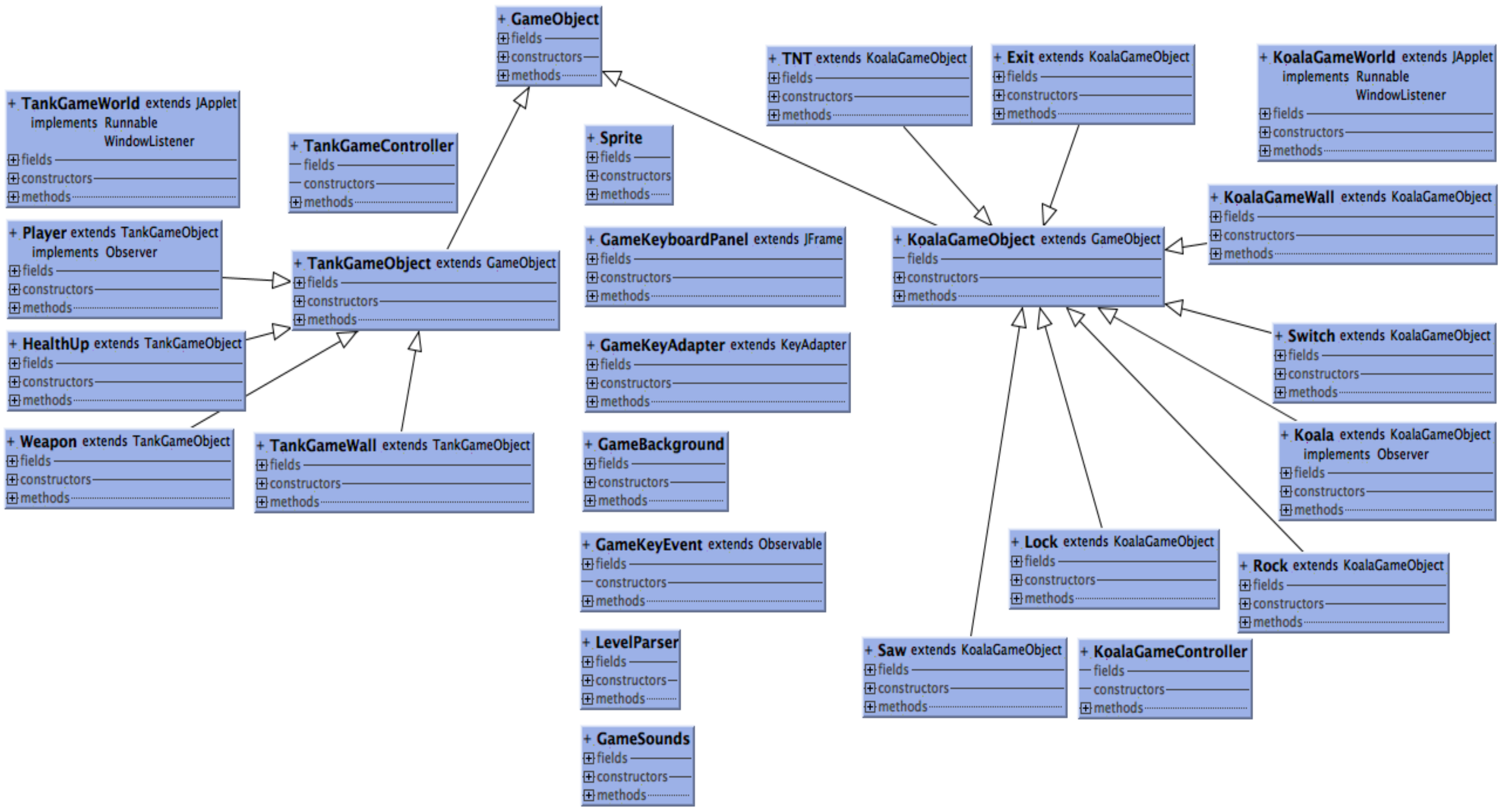
Package tankGame/game	Function of Package
TankGameObject.java	This package has all the logic and implementation needed for creating the Tank Game
TankGameWall.java	
TankGameWarld.java	
Player.java	
Weapon.java	
HealthUp.java	
TankGameController.java	

Package koalaGame/game	Function of Package
KoalaGameController.java	This package has all the logic and implementation needed for creating the Koala Game
KoalaGameWorld.java	
KoalaGameObject.java	
KoalaGameWorld.java	
Koala.java	
Exit.java	
Lock.java	
Saw.java	
Switch.java	
TNT.java	
Rock.java	

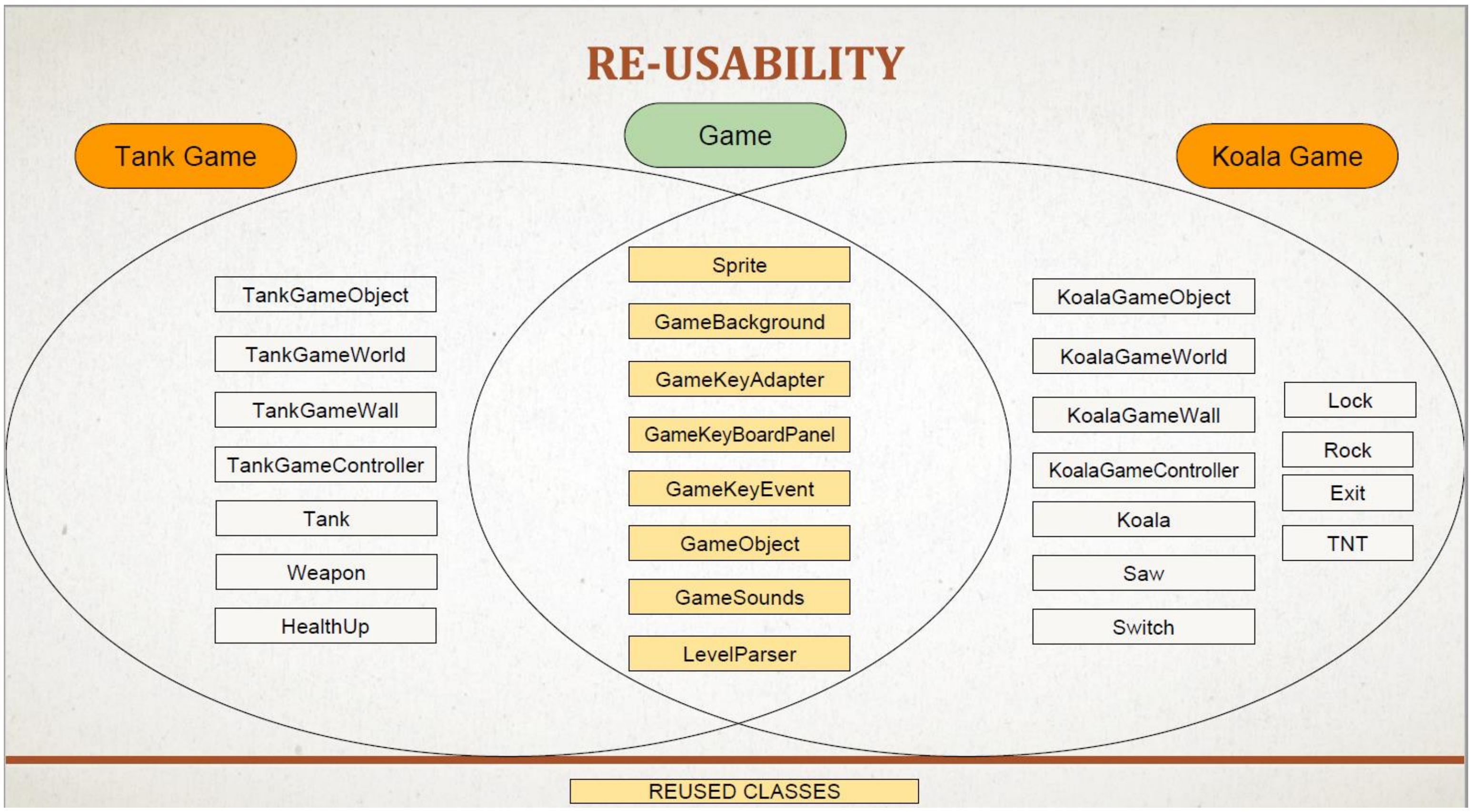
Package graphics	Function of Package
Sprite.java	This package helps to process the Sprites

Implementation of Tank and Koala games

The following figure illustrates how the various classes of **this project** interact:


GameObject
TankGameWorld extends JApplet implements Runnable WindowListener
TankGameController
Player extends TankGameObject implements Observer
TankGameObject extends GameObject
HealthUp extends TankGameObject
Weapon extends TankGameObject
TankGameWall extends TankGameObject
Sprite
GameKeyboardPanel extends JFrame
GameKeyAdapter extends KeyAdapter
GameBackground
GameKeyEvent extends Observable
LevelParser
GameSounds
TNT extends KoalaGameObject
Exit extends KoalaGameObject
KoalaGameWorld extends JApplet implements Runnable WindowListener
KoalaGameObject extends GameObject
KoalaGameWall extends KoalaGameObject
Switch extends KoalaGameObject
Koala extends KoalaGameObject implements Observer
Lock extends KoalaGameObject
Rock extends KoalaGameObject
Saw extends KoalaGameObject
KoalaGameController
fields
constructors
methods

The following figure illustrates how the classes were reused:

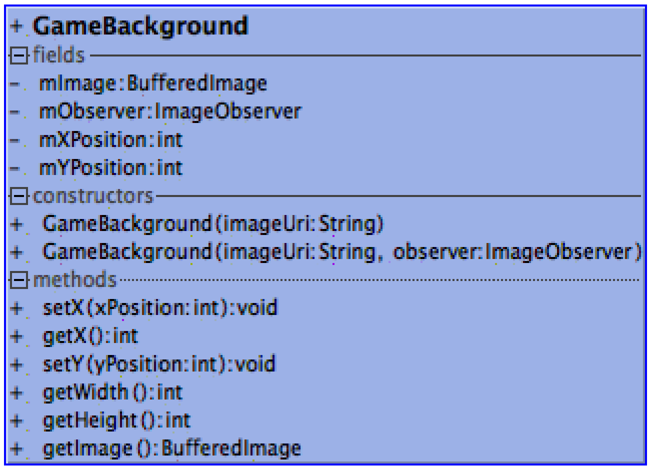


The implementation of both Games involved working on the following classes:

Classes that are intended for reuse:

<u>GameBackground.java</u>

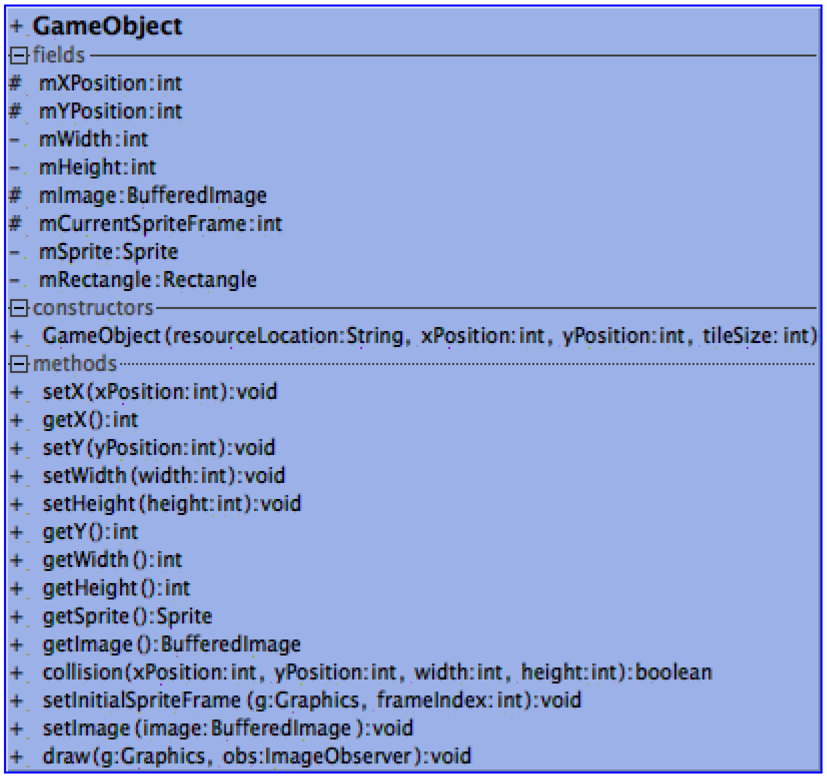
The GameBackground class is used to setup the game's background image.



```
+ GameBackground
⊟ fields
-  mImage:BufferedImage
-  mObserver:ImageObserver
-  mXPosition:int
-  mYPosition:int
⊟ constructors
+  GameBackground(imageUri:String)
+  GameBackground(imageUri:String, observer:ImageObserver)
⊟ methods
+  setX(xPosition:int):void
+  getX():int
+  setY(yPosition:int):void
+  getWidth():int
+  getHeight():int
+  getImage():BufferedImage
```

<u>GameObject.java</u>

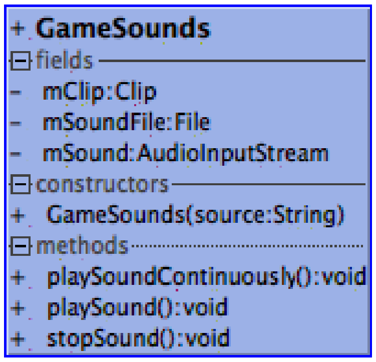
The GameObject class is used to record all the necessary info (image, position, size, bounds) about each object that extends it. Moreover, this class establishes some behavior that game's objects share. Also, it provides an ability to update objects' data through getters/setters.



```
+ GameObject
⊟ fields
#  mXPosition:int
#  mYPosition:int
-  mWidth:int
-  mHeight:int
#  mImage:BufferedImage
#  mCurrentSpriteFrame:int
-  mSprite:Sprite
-  mRectangle:Rectangle
⊟ constructors
+  GameObject(resourceLocation:String, xPosition:int, yPosition:int, tileSize: int)
⊟ methods
+  setX(xPosition:int):void
+  getX():int
+  setY(yPosition:int):void
+  setWidth(width:int):void
+  setHeight(height:int):void
+  getY():int
+  getWidth():int
+  getHeight():int
+  getSprite():Sprite
+  getImage():BufferedImage
+  collision(xPosition:int, yPosition:int, width:int, height:int):boolean
+  setInitialSpriteFrame(g:Graphics, frameIndex:int):void
+  setImage(image:BufferedImage):void
+  draw(g:Graphics, obs:ImageObserver):void
```

<u>GameSounds.java</u>

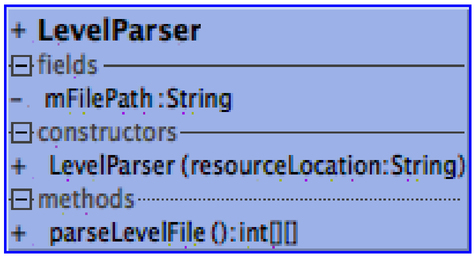
The GameSounds class is used to play background music, and play unique sound effects for any actions: tank/wall/TNT explosions, weapon fire, saw, game over, etc.



```
+ GameSounds
⊟ fields
-  mClip:Clip
-  mSoundFile:File
-  mSound:AudioInputStream
⊟ constructors
+  GameSounds(source:String)
⊟ methods
+  playSoundContinuously():void
+  playSound():void
+  stopSound():void
```

<u>Levelparser.java</u>

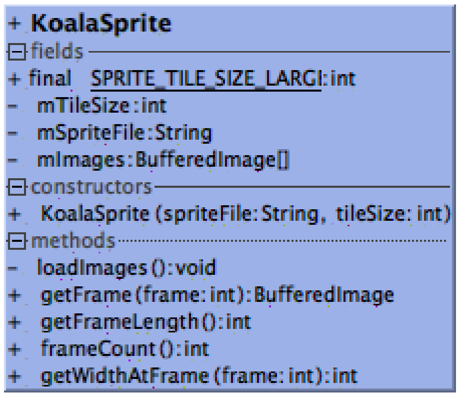
This class is used to read the file (game level) and parse the data from that file into two-dimensional array.



```
+ LevelParser
⊟ fields
-  mFilePath:String
⊟ constructors
+  LevelParser(resourceLocation:String)
⊟ methods
+  parseLevelFile():int[][]
```

<u>Sprite.java</u>

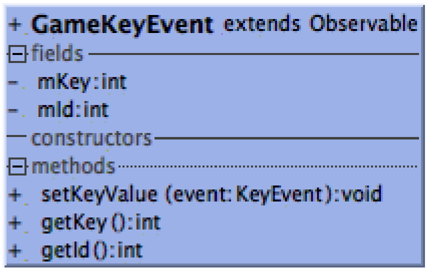
The Sprite class just creates a new sprite for use. This means we can create a new sprite, or list of sprites with different tile sizes. Also, this class helps us to get information about each frame that is loaded into bufferImage array (frame count, frame length, width and etc).



```
+ KoalaSprite
⊟fields
+ final  SPRITE_TILE_SIZE_LARGI:int
-  mTileSize:int
-  mSpriteFile:String
-  mImages:BufferedImage[]
⊟constructors
+  KoalaSprite (spriteFile:String, tileSize: int)
⊟methods
-  loadImages ():void
+  getFrame (frame: int):BufferedImage
+  getFrameLength ():int
+  frameCount ():int
+  getWidthAtFrame (frame: int):int
```

GameKeyEvent.java

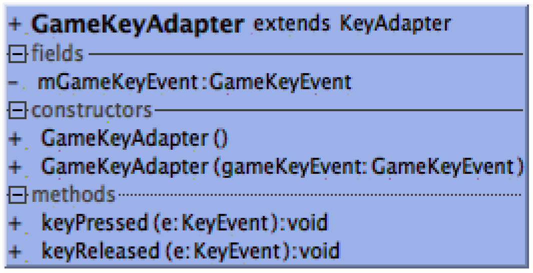
This class extends Observable class; therefore, it has ability notify the players(observers) when event happened (keyboard event). This class keeps track of the state that each player (if there are more than 1 player) is in via their current key press. An instance of this class can be created per player to allow simultaneous movement.



```
+ GameKeyEvent extends Observable
⊟fields
-  mKey:int
-  mId:int
— constructors
⊟methods
+  setKeyValue (event: KeyEvent):void
+  getKey ():int
+  getId ():int
```

GameKeyAdapter.java

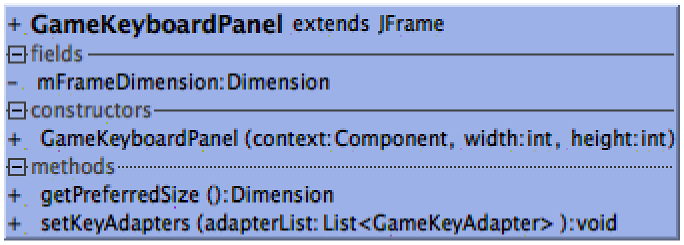
This class extends the KeyAdapter class, allowing it to override the `keyPressed()` and `keyReleased` methods, so that we can notify the `GameKeyEvent` class of any state changes.



```
+ GameKeyAdapter extends KeyAdapter
⊟fields
-  mGameKeyEvent:GameKeyEvent
⊟constructors
+  GameKeyAdapter ()
+  GameKeyAdapter (gameKeyEvent: GameKeyEvent)
⊟methods
+  keyPressed (e:KeyEvent):void
+  keyReleased (e:KeyEvent):void
```

GameKeyboardPanel.java

This class serves as the JFrame for the game. In the constructor, it sets all of the initial bounds and information, such as title, and width/height of the frame. Since the JFrame retains focus of the game, it is also in charge of setting the KeyListeners on each of the adapter instances.

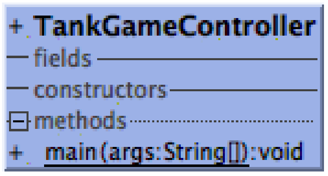


```
+ GameKeyboardPanel extends JFrame
⊟ fields
- mFrameDimension:Dimension
⊟ constructors
+ GameKeyboardPanel (context:Component, width:int, height:int)
⊟ methods
+ getPreferredSize ():Dimension
+ setKeyAdapters (adapterList: List<GameKeyAdapter> ):void
```

Classes that were created only for Tank Game:

<u>TankGameController.java</u>

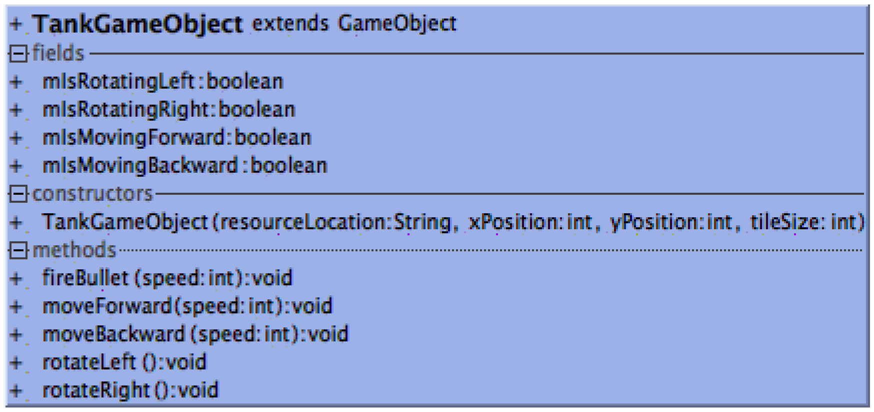
The TankGameController class contains the main method that starts the game.



```
+ TankGameController
— fields
— constructors
⊟ methods
+ main(args:String[]):void
```

<u>TankGameObject.java</u>

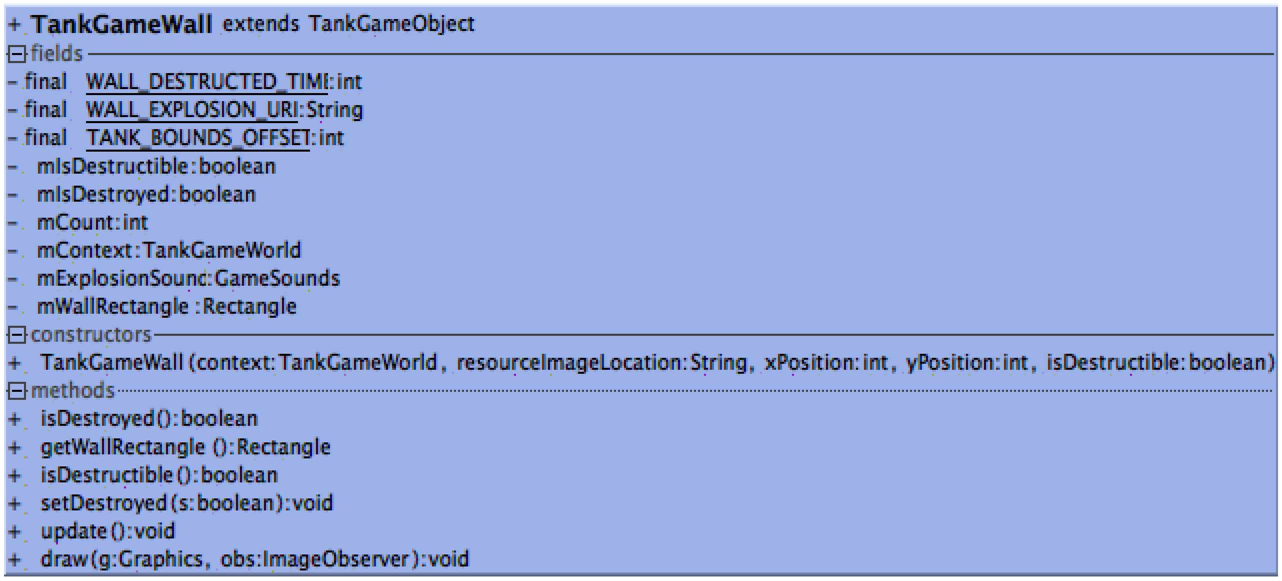
The TankGameObject class extends GameObject class and it helps with the tanks rotation and movements.



```
+ TankGameObject extends GameObject
⊟ fields
+ mIsRotatingLeft:boolean
+ mIsRotatingRight:boolean
+ mIsMovingForward:boolean
+ mIsMovingBackward:boolean
⊟ constructors
+ TankGameObject (resourceLocation:String, xPosition:int, yPosition:int, tileSize: int)
⊟ methods
+ fireBullet (speed:int):void
+ moveForward(speed:int):void
+ moveBackward (speed:int):void
+ rotateLeft ():void
+ rotateRight ():void
```

<u>TankGameWall.java</u>

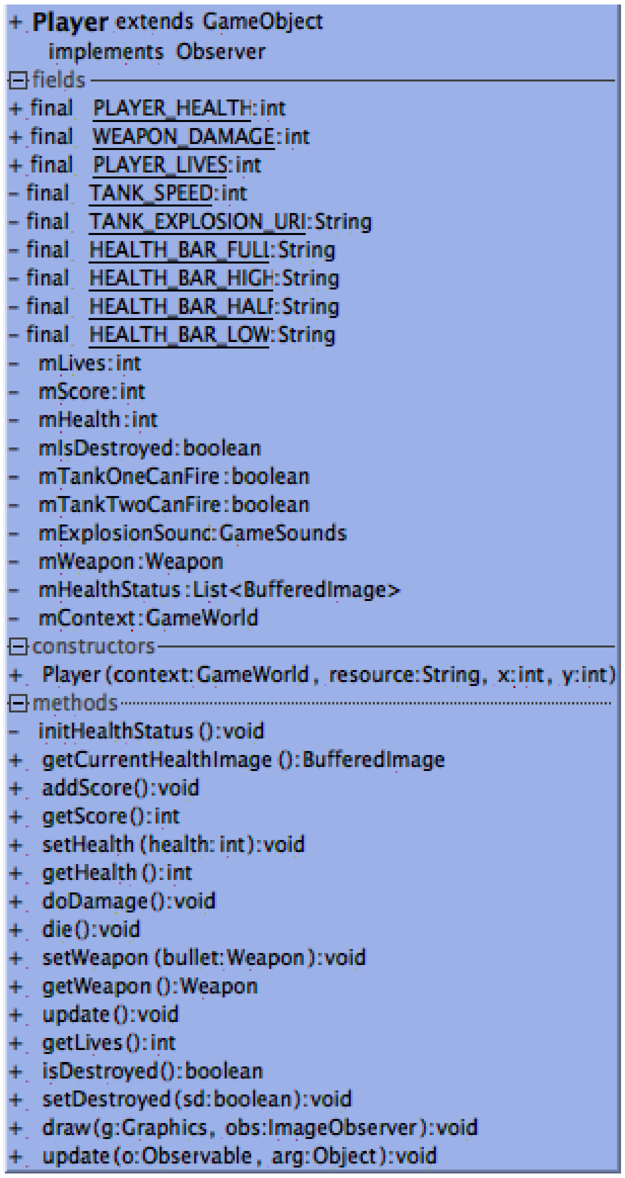
The TankGameWall class extends TankGameObject and it is used to construct the two type of walls: destructible and indestructible walls. This class checks if collision happened between tank and wall, preventing movement of tanks through the wall boundaries. Some of the wall blocks can be destroyed using the tank weapons, but these blocks will reappear after certain time.



```
+ TankGameWall extends TankGameObject
⊟ fields
- final  WALL_DESTRUCTED_TIMI:int
- final  WALL_EXPLOSION_URI:String
- final  TANK_BOUNDS_OFFSET:int
-  mIsDestructible:boolean
-  mIsDestroyed:boolean
-  mCount:int
-  mContext:TankGameWorld
-  mExplosionSounc:GameSounds
-  mWallRectangle :Rectangle
⊟ constructors
+  TankGameWall (context:TankGameWorld, resourceImageLocation:String, xPosition:int, yPosition:int, isDestructible:boolean)
⊟ methods
+  isDestroyed():boolean
+  getWallRectangle ():Rectangle
+  isDestructible():boolean
+  setDestroyed(s:boolean):void
+  update():void
+  draw(g:Graphics, obs:ImageObserver):void
```

Player.java (Tank)

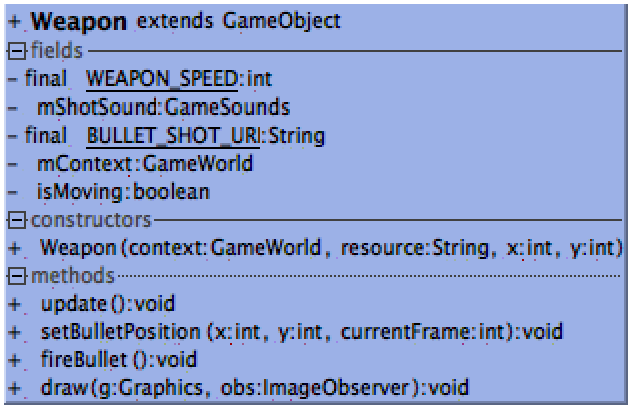
The Player class extends TankGameObject and implements Observer. This class is used to create instances for each tank, so each tank has its own data field: health, lives, type of bullets, score and etc. Since this class overrides the update method of Observer class, it helps to update each tank - player properly.



```
+ Player extends GameObject
    implements Observer
⊟fields
+ final  PLAYER_HEALTH:int
+ final  WEAPON_DAMAGE:int
+ final  PLAYER_LIVES:int
- final  TANK_SPEED:int
- final  TANK_EXPLOSION_URI:String
- final  HEALTH_BAR_FULL:String
- final  HEALTH_BAR_HIGH:String
- final  HEALTH_BAR_HALF:String
- final  HEALTH_BAR_LOW:String
-   mLives:int
-   mScore:int
-   mHealth:int
-   mIsDestroyed:boolean
-   mTankOneCanFire:boolean
-   mTankTwoCanFire:boolean
-   mExplosionSound:GameSounds
-   mWeapon:Weapon
-   mHealthStatus:List<BufferedImage>
-   mContext:GameWorld
⊟constructors
+  Player (context:GameWorld, resource:String, x:int, y:int)
⊟methods
-  initHealthStatus ():void
+  getCurrentHealthImage ():BufferedImage
+  addScore():void
+  getScore():int
+  setHealth (health: int):void
+  getHealth ():int
+  doDamage():void
+  die():void
+  setWeapon (bullet:Weapon):void
+  getWeapon ():Weapon
+  update ():void
+  getLives ():int
+  isDestroyed():boolean
+  setDestroyed(sd:boolean):void
+  draw(g:Graphics, obs:ImageObserver):void
+  update(o:Observable, arg:Object):void
```

Weapon.java

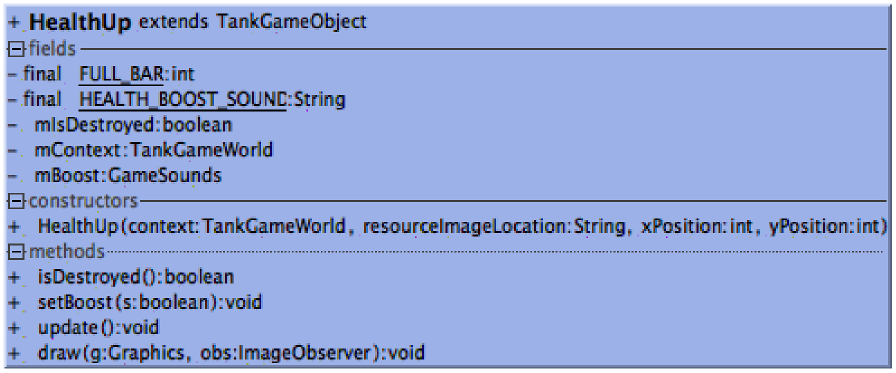
This class extends TankGameObject and it's used to construct the two type of bullets for tanks. The bullets can move at the different angles, do damage, and destroy the destructible walls and tanks.



```
+ Weapon extends GameObject
⊟fields
- final  WEAPON_SPEED:int
-   mShotSound:GameSounds
- final  BULLET_SHOT_URI:String
-   mContext:GameWorld
-   isMoving:boolean
⊟constructors
+  Weapon (context:GameWorld, resource:String, x:int, y:int)
⊟methods
+  update ():void
+  setBulletPosition (x:int, y:int, currentFrame:int):void
+  fireBullet ():void
+  draw(g:Graphics, obs:ImageObserver):void
```

HealthUp.java

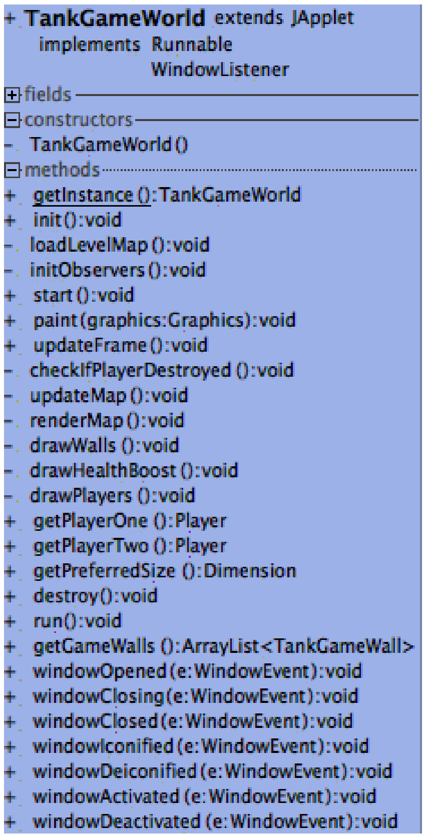
This class extends TankGameObject and it's used to give health boost to the tanks. This feature only works if the tank's health is not full.



```
+ HealthUp extends TankGameObject
⊟fields
- final  FULL_BAR:int
- final  HEALTH_BOOST_SOUND:String
-  mIsDestroyed:boolean
-  mContext:TankGameWorld
-  mBoost:GameSounds
⊟constructors
+  HealthUp(context:TankGameWorld, resourceImageLocation:String, xPosition:int, yPosition:int)
⊟methods
+  isDestroyed():boolean
+  setBoost(s:boolean):void
+  update():void
+  draw(g:Graphics, obs:ImageObserver):void
```

<u>TankGameWorld.java</u>

This class serves as the main loop of the game, and operates as the center of nearly all of the interactions that take place during the game. Using JApplet's lifecycle: init(), start() and destroy(); we can moderate appropriate and expected behavior, such as initializing objects and the map in init(), creating and starting a new Thread in start(), and removing references, stop the background music, and interrupting the main Thread in destroy(). In TankGameWorld's paint() method, we go through the process of drawing and updating every object each frame. This is done via the updateFrame() method. In this method, we handle the drawing of the health bars and any text on the screen, then, if the game hasn't ended, we update the map, render the map, draw the walls, draw the players, calculate the sub-screens and where they should be displayed, as well as the mini-map, and finally we draw them all using the Graphics2D object.

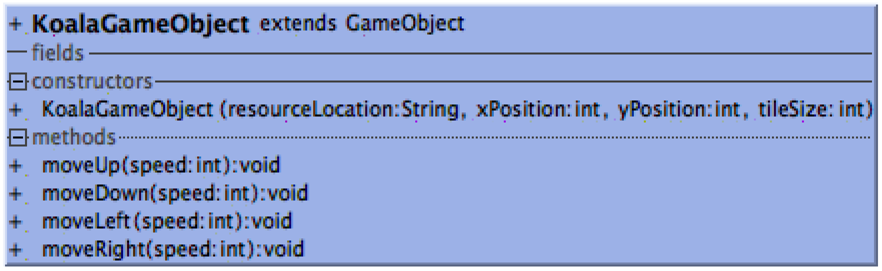


```
+ TankGameWorld extends JApplet
    implements Runnable
               WindowListener
⊞fields
⊟constructors
-  TankGameWorld()
⊟methods
+  getInstance():TankGameWorld
+  init():void
-  loadLevelMap():void
-  initObservers():void
+  start():void
+  paint(graphics:Graphics):void
+  updateFrame():void
-  checkIfPlayerDestroyed():void
-  updateMap():void
-  renderMap():void
-  drawWalls():void
-  drawHealthBoost():void
-  drawPlayers():void
+  getPlayerOne():Player
+  getPlayerTwo():Player
+  getPreferredSize():Dimension
+  destroy():void
+  run():void
+  getGameWalls():ArrayList<TankGameWall>
+  windowOpened(e:WindowEvent):void
+  windowClosing(e:WindowEvent):void
+  windowClosed(e:WindowEvent):void
+  windowIconified(e:WindowEvent):void
+  windowDeiconified(e:WindowEvent):void
+  windowActivated(e:WindowEvent):void
+  windowDeactivated(e:WindowEvent):void
```

Classes that were created only for Koala Game:

KoalaGameObject.java

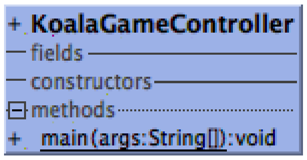
The KoalaGameObject class extends GameObject class and it helps with the koala movements.



```
+ KoalaGameObject extends GameObject
— fields
⊟ constructors
+  KoalaGameObject (resourceLocation:String, xPosition:int, yPosition:int, tileSize: int)
⊟ methods
+  moveUp(speed:int):void
+  moveDown(speed:int):void
+  moveLeft(speed:int):void
+  moveRight(speed:int):void
```

KoalaGameController.java

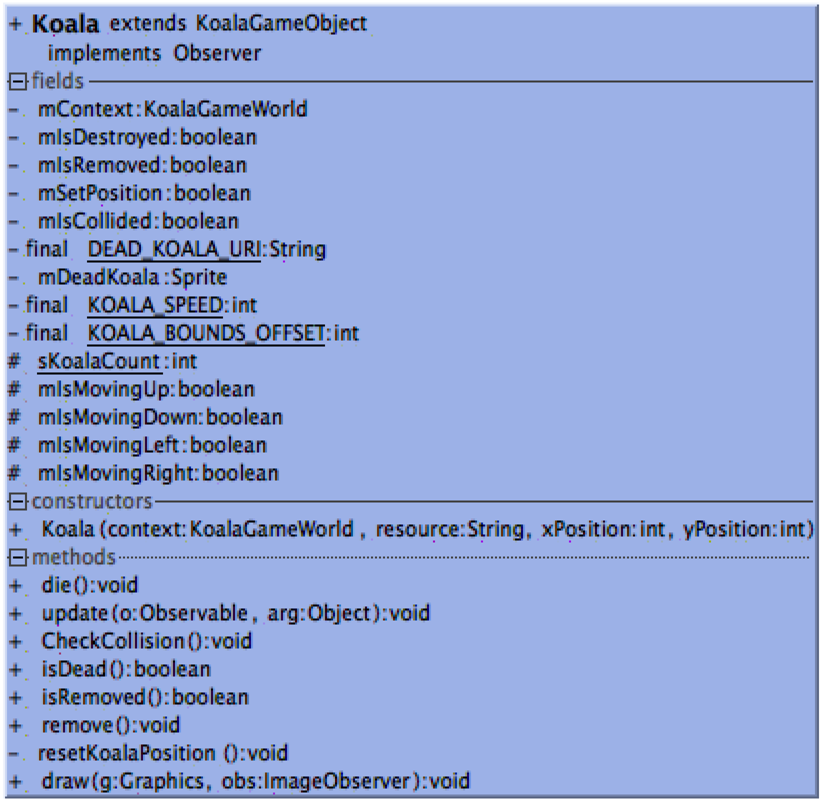
The KoalaGameController class contains the main method that starts the game.



```
+ KoalaGameController
— fields
— constructors
⊟ methods
+  main(args:String[]):void
```

Koala.java

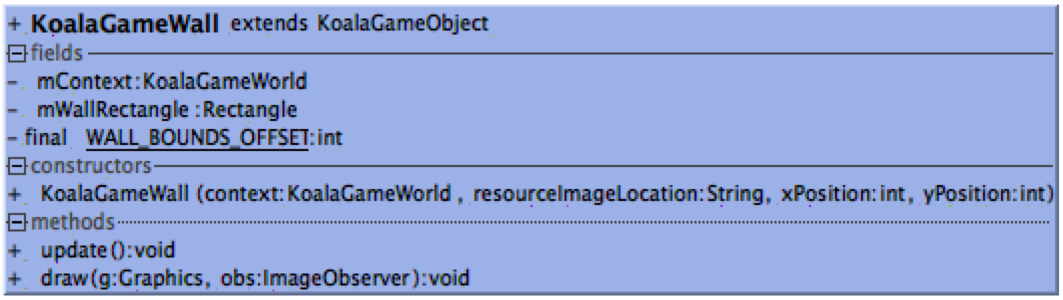
The Koala class extends KoalaGameObject and implements Observer. This class is used to create instances for each koala, so each koala has its own data field. Since this class overrides the update method of Observer class, it helps to update each koala properly.



```
+ Koala extends KoalaGameObject
    implements Observer
⊟ fields
-  mContext:KoalaGameWorld
-  mIsDestroyed:boolean
-  mIsRemoved:boolean
-  mSetPosition:boolean
-  mIsCollided:boolean
- final  DEAD_KOALA_URI:String
-  mDeadKoala:Sprite
- final  KOALA_SPEED:int
- final  KOALA_BOUNDS_OFFSET:int
#  sKoalaCount:int
#  mIsMovingUp:boolean
#  mIsMovingDown:boolean
#  mIsMovingLeft:boolean
#  mIsMovingRight:boolean
⊟ constructors
+  Koala (context:KoalaGameWorld, resource:String, xPosition:int, yPosition:int)
⊟ methods
+  die():void
+  update(o:Observable, arg:Object):void
+  CheckCollision():void
+  isDead():boolean
+  isRemoved():boolean
+  remove():void
-  resetKoalaPosition ():void
+  draw(g:Graphics, obs:ImageObserver):void
```

KoalaGameWall.java

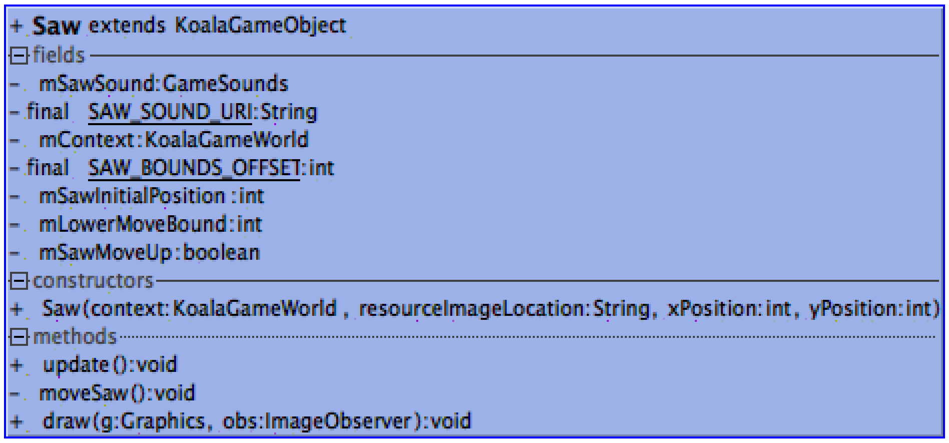
The KoalaGameWall class extends KoalaGameObject and it is used to construct the indestructible walls. This class checks if collision happened between koala and wall, preventing movement of koala through the wall boundaries.



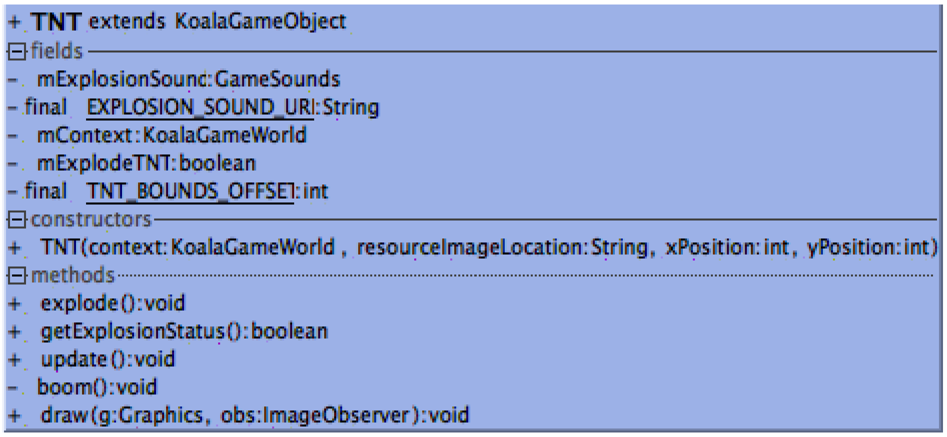
```
+ KoalaGameWall extends KoalaGameObject
⊟ fields
-  mContext:KoalaGameWorld
-  mWallRectangle :Rectangle
- final  WALL_BOUNDS_OFFSET:int
⊟ constructors
+  KoalaGameWall (context:KoalaGameWorld , resourceImageLocation:String, xPosition:int, yPosition:int)
⊟ methods
+  update():void
+  draw(g:Graphics, obs:ImageObserver):void
```

Saw.java and ***TNT.java***

Saw and TNT classes extend KoalaGameObject and they used to create the fatal hazards that kill the koala once he collides with the saw or TNT.



```
+ Saw extends KoalaGameObject
⊟ fields
-  mSawSound:GameSounds
- final  SAW_SOUND_URI:String
-  mContext:KoalaGameWorld
- final  SAW_BOUNDS_OFFSET:int
-  mSawInitialPosition :int
-  mLowerMoveBound:int
-  mSawMoveUp:boolean
⊟ constructors
+  Saw(context:KoalaGameWorld , resourceImageLocation:String, xPosition:int, yPosition:int)
⊟ methods
+  update():void
-  moveSaw():void
+  draw(g:Graphics, obs:ImageObserver):void
```

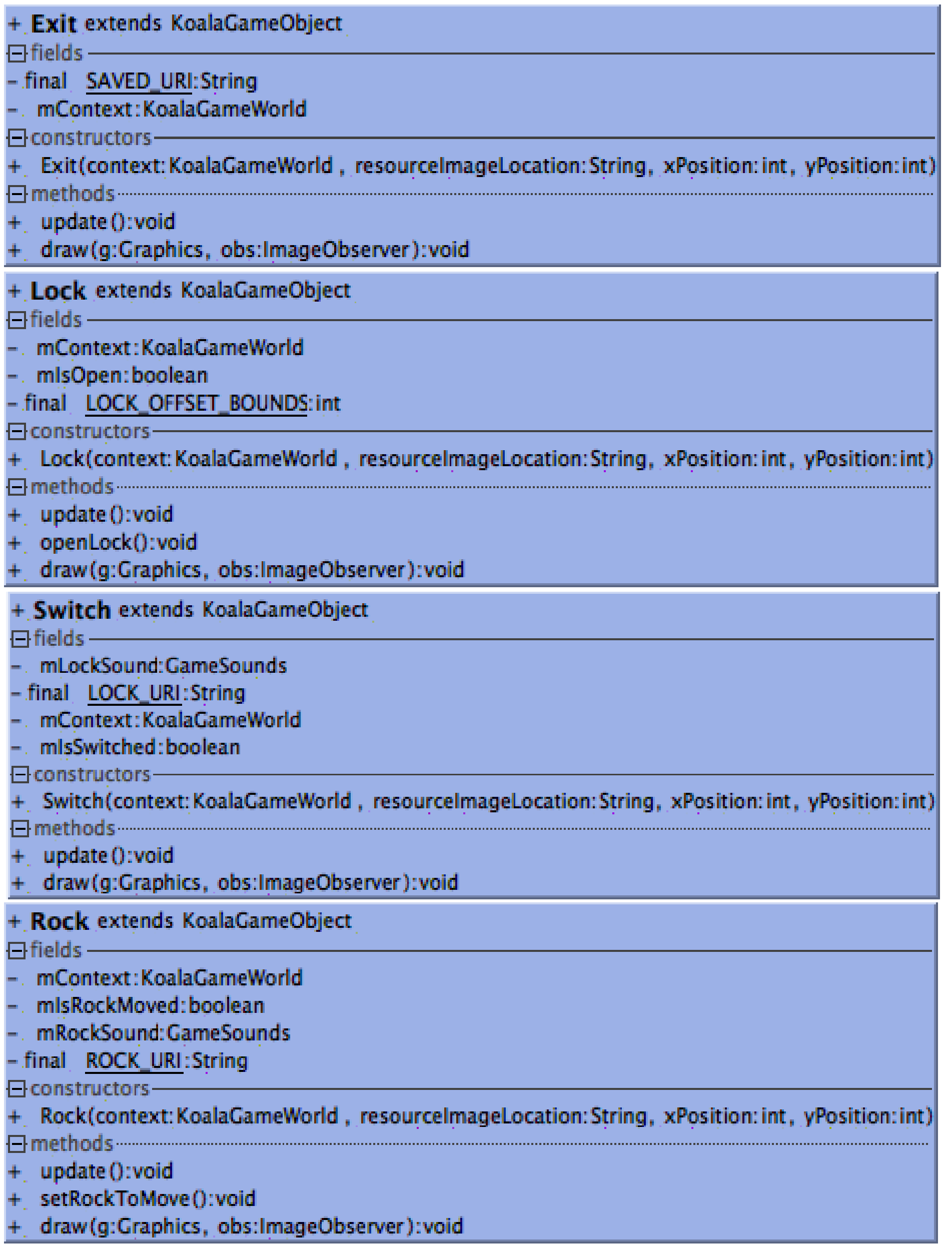


```
+ TNT extends KoalaGameObject
⊟ fields
-  mExplosionSound:GameSounds
- final  EXPLOSION_SOUND_URI:String
-  mContext:KoalaGameWorld
-  mExplodeTNT:boolean
- final  TNT_BOUNDS_OFFSET:int
⊟ constructors
+  TNT(context:KoalaGameWorld , resourceImageLocation:String, xPosition:int, yPosition:int)
⊟ methods
+  explode():void
+  getExplosionStatus():boolean
+  update():void
-  boom():void
+  draw(g:Graphics, obs:ImageObserver):void
```

Lock.java, Switch.java, Rock.java, and ***Exit.java***

Lock, Switch, Rock, and Exit classes extend KoalaGameObject and they used to create non-fatal hazards.

- Red exits—Allow any number of koalas to exit the level
- Locks and Switches— Block and Open locked passageways
- Rock—Can be pushed by koalas and destroy other hazards



```
+ Exit extends KoalaGameObject
⊟ fields
- final  SAVED_URI:String
-  mContext:KoalaGameWorld
⊟ constructors
+  Exit(context:KoalaGameWorld , resourceImageLocation:String, xPosition:int, yPosition:int)
⊟ methods
+  update():void
+  draw(g:Graphics, obs:ImageObserver):void
```

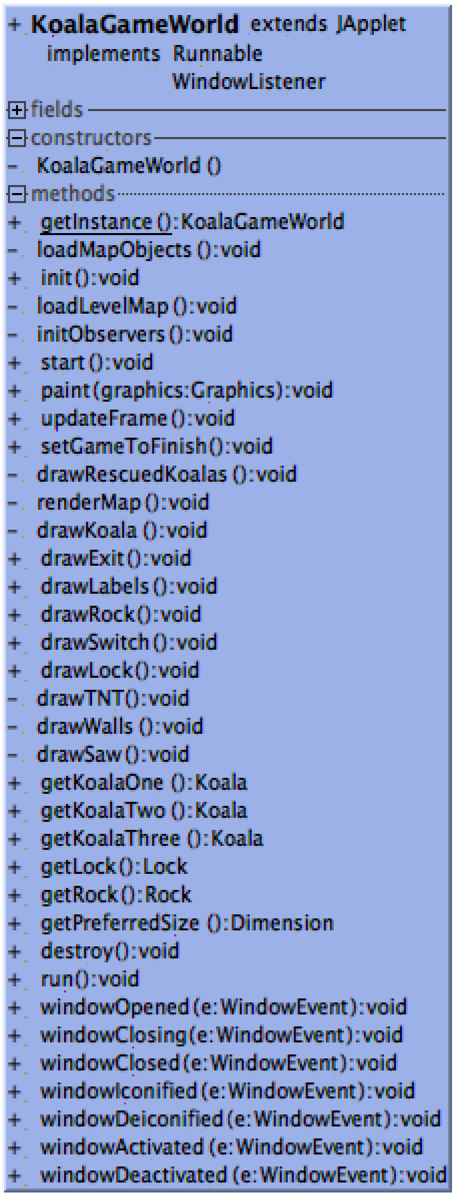
```
+ Lock extends KoalaGameObject
⊟ fields
-  mContext:KoalaGameWorld
-  mIsOpen:boolean
- final  LOCK_OFFSET_BOUNDS:int
⊟ constructors
+  Lock(context:KoalaGameWorld , resourceImageLocation:String, xPosition:int, yPosition:int)
⊟ methods
+  update():void
+  openLock():void
+  draw(g:Graphics, obs:ImageObserver):void
```

```
+ Switch extends KoalaGameObject
⊟ fields
-  mLockSound:GameSounds
- final  LOCK_URI:String
-  mContext:KoalaGameWorld
-  mIsSwitched:boolean
⊟ constructors
+  Switch(context:KoalaGameWorld , resourceImageLocation:String, xPosition:int, yPosition:int)
⊟ methods
+  update():void
+  draw(g:Graphics, obs:ImageObserver):void
```

```
+ Rock extends KoalaGameObject
⊟ fields
-  mContext:KoalaGameWorld
-  mIsRockMoved:boolean
-  mRockSound:GameSounds
- final  ROCK_URI:String
⊟ constructors
+  Rock(context:KoalaGameWorld , resourceImageLocation:String, xPosition:int, yPosition:int)
⊟ methods
+  update():void
+  setRockToMove():void
+  draw(g:Graphics, obs:ImageObserver):void
```

KoalaGameWorld.java

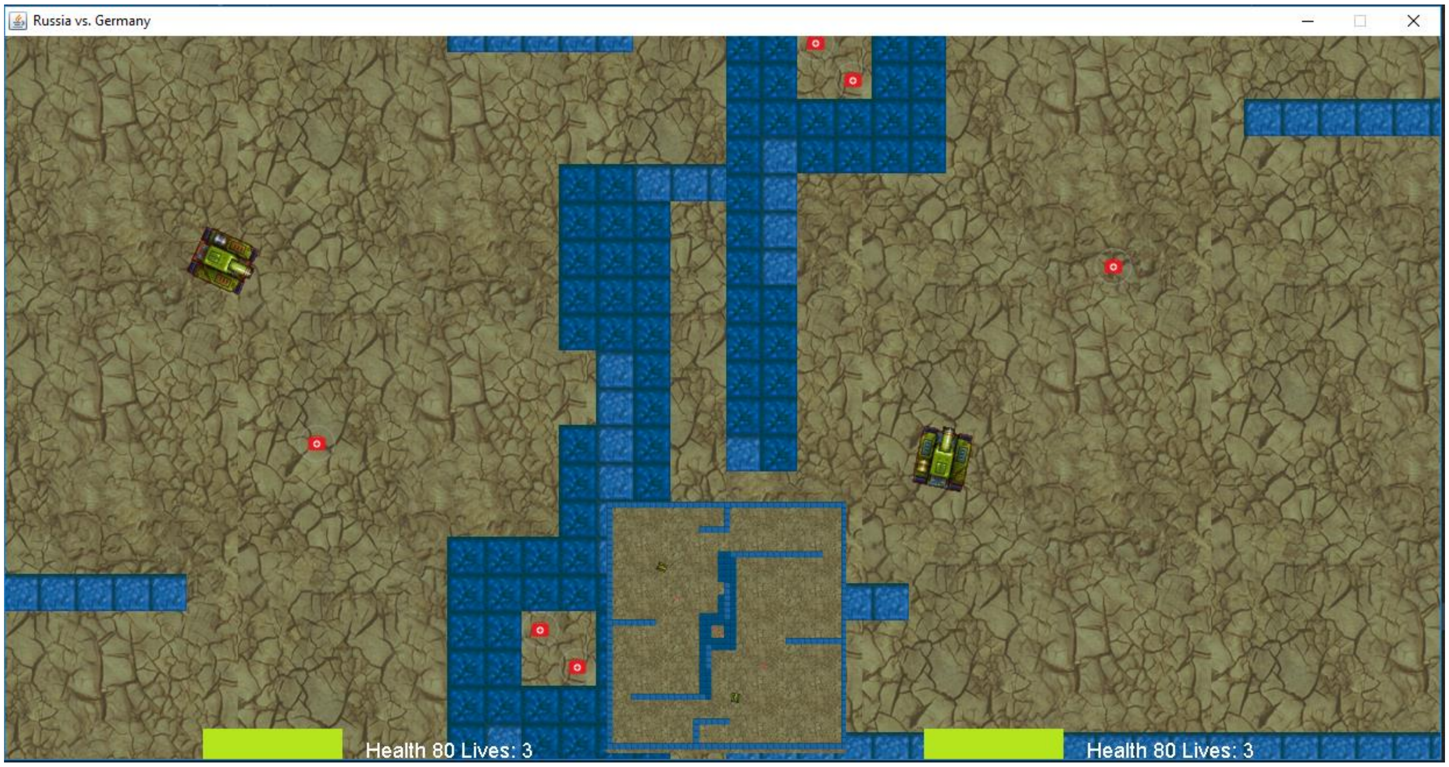
This class serves as the main loop of the game, and operates as the center of nearly all of the interactions that take place during the game. Using JApplet's lifecycle: init(), start() and destroy(); we can moderate appropriate and expected behavior, such as initializing objects and the map in init(), creating and starting a new Thread in start(), and removing references, stop the background music, and interrupting the main Thread in destroy(). In KoalaGameWorld's paint() method, we go through the process of drawing and updating every object each frame. This is done via the updateFrame() method. In this method, we handle the drawing of the saws, TNT, exit, rock and any text on the screen, then, if the game hasn't ended, we update the map, render the map, draw the walls, draw the koalas, different obstacles, and finally we draw them all using the Graphics2D object.



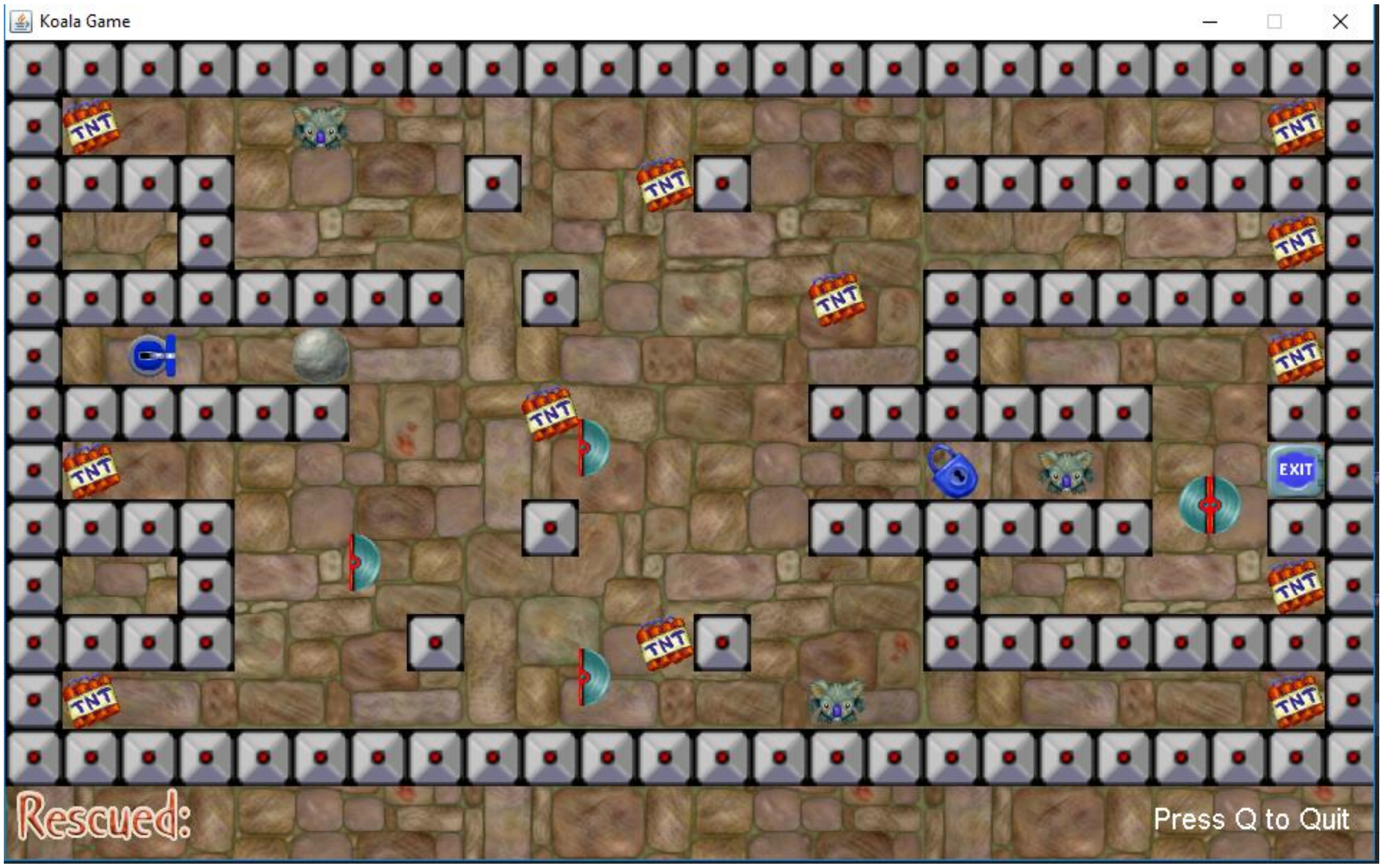
```
+ KoalaGameWorld extends JApplet
    implements Runnable
               WindowListener
⊞ fields
⊟ constructors
-  KoalaGameWorld ()
⊟ methods
+  getInstance ():KoalaGameWorld
-  loadMapObjects ():void
+  init ():void
-  loadLevelMap ():void
-  initObservers ():void
+  start ():void
+  paint (graphics:Graphics):void
+  updateFrame ():void
+  setGameToFinish ():void
-  drawRescuedKoalas ():void
-  renderMap ():void
-  drawKoala ():void
+  drawExit ():void
+  drawLabels ():void
+  drawRock ():void
+  drawSwitch ():void
+  drawLock ():void
-  drawTNT ():void
-  drawWalls ():void
-  drawSaw ():void
+  getKoalaOne ():Koala
+  getKoalaTwo ():Koala
+  getKoalaThree ():Koala
+  getLock ():Lock
+  getRock ():Rock
+  getPreferredSize ():Dimension
+  destroy ():void
+  run ():void
+  windowOpened (e:WindowEvent):void
+  windowClosing (e:WindowEvent):void
+  windowClosed (e:WindowEvent):void
+  windowIconified (e:WindowEvent):void
+  windowDeiconified (e:WindowEvent):void
+  windowActivated (e:WindowEvent):void
+  windowDeactivated (e:WindowEvent):void
```

<u>Sample Run</u>

Here is an example that shows a snapshot of the Tank game the user is expected to see after he/she runs a game:



Here is an example that shows a snapshot of the Koala game the user is expected to see after he/she runs a game:


Koala Game
TNT
EXIT
Rescued:
Press Q to Quit