capstone1-TheProject LAB REPORT/ Document SQUASH GAME : TEAM 35

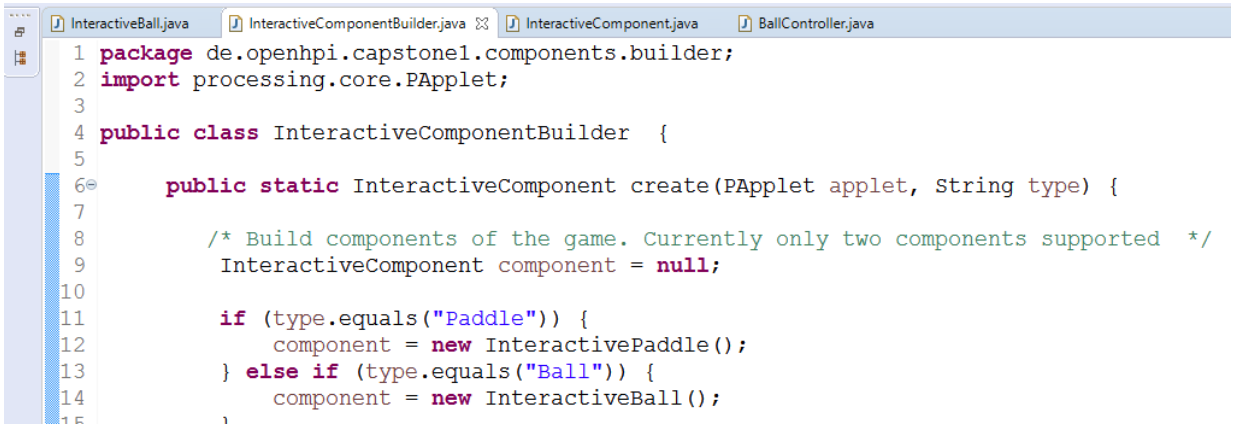
Contents

Introduction:

After going through all the course videos, it was decided that capstone1 counter5 would be more appropriate to be used as reference for the design of Squash game as it has all required details to implement the game based on Model-view-Controller pattern (MVC).

Initially, we found it hard to decide which classes needs to be implemented. The only concrete idea that was prevalent in the team was, that requirement of classes for both Ball and paddle.

Challenge / Issue-1:



The basic code scaffold was already provided as part of course. here a frame work was provided to build components as shown in above figure.

- The first question was, is this scalable? Suppose: an invalid string is passed. What happens? Exception?

```
@Override
public void setup() {  // setup() runs once
    rectMode(CENTER);
    //  frameRate(30);
    /* Currently the code supports creation of only two components: "Paddle"  and "Ball"   */
    interactivePaddle = InteractiveComponentBuilder.create(this, "MyPaddle");
    interactiveBall = InteractiveComponentBuilder.create(this, "Ball");
```

As shown in above figure, Mypaddle is used as argument instead of Paddle. This needs exception handling.

- Next, Suppose it is required to add new component. Then again this class must be updated / modified. So the code needs modification every time a new component is added. That is, at later stage if this squash game needs to be upgraded to some other new game like Pong game, then it needs modifications.

It was decided to address this at later stage and tiried some ideas but did not succeed.So decided to stick-on to the existing frame work.

Initially decided to add Null Exception handling but this was not feasible, since we had to add this at many places where methods of the object was involved .This resulted in a cluttered code.

Later concluded that an appropriate fix would be to add null handling as shown below.

```
/* Build components of the game. Currently only two components supported  */
 InteractiveComponent component = null;

 if (type.equals("Paddle")) {
     component = new InteractivePaddle();
 } else if (type.equals("Ball")) {
     component = new InteractiveBall();
 }
 /* Handle invalid component type */
 if(component == null) {
     System.out.println("Invalid component specified while building game components."
             + " Check argument to method: InteractiveComponent.create() ");
     System.exit(-1);
 }
```

Challenge / Issue-2:

How to handle the interaction between the ball and paddle?
We had no clue about this initially. How do you model such a behaviour? This was not discussed in any of the videos earlier.

Since both the Models Ball and Paddle needed to interact with each other. So which one will handle it. Since ball is component that is very much active in the game, we decided to add additional functionalities to Ball Model.

The connection between the component is done by the InteractiveComponent. connectComponents( ) of InteractiveComponentBuilder class.

This class also adds Model, Views and controllers for both the interactive components: InteractiveBall and InteractivePaddle.
These two are derived from `abstract class InteractiveComponent`

Similarly the `class BallController implements Controller` and `class PaddleController implements Controller.`

This interface `Controller` provides all the necessary methods that must be implemented. The `PaddleController` handles the keyPressed () event whereas the `BallController` handles the keyReleased()

Challenge / Issue-3:

Assigning the same key event was a problematic and was causing key bounce in the initial screen. i.e. when a key is pressed longer, then multiple events were logged and the two initial screens would move very fast in successive fashion.

As a solution, we used keyReleased() for the `BallController.`

Thus the issue was fixed because there can only event generated when a key is released .On the contrary multiple events can be generated when a key is pressed.

Challenge / Issue-4:

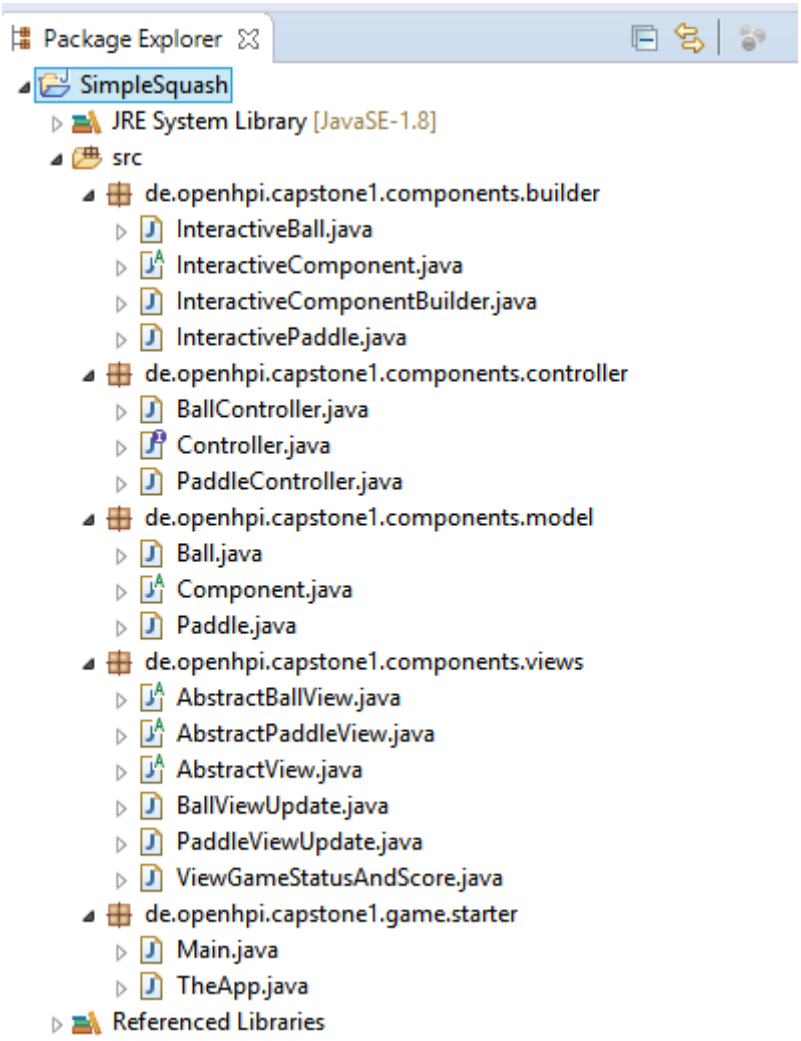
Once Ball and Paddle classes were completed, we found that there were many attributes were in common. This called for a design change and the result was a new Super class, `abstract class Component.`

This class handles all the data associated with ball and paddle like dimensions, color, vertical and horizontal speed, X and Y positions. Specific functionalities are handled by the Ball class.

Brief description of functionalities of important Classes:

The name of the proejct is : SimpleSquash

The final set of classes are shown in the screen shot below




```
public class Main
```

This calls the main function of PApplet class.

```
public class TheApp extends PApplet
```

Implements method setup() to do the initial settings of the game: like screen size, the components of the game, initialises the Interactive

Paddle and Ball for the game, sets the initial dimensions, color of paddle and Ball and finally velocities of ball and paddle.
Finally the draw () method (which loops continuously) calls the methods `interactivePaddle.update()` and `interactiveBall.update()`.These methods calls the move () methods of ball and Paddle and also checks the Collision between Paddle and Ball and takes necessary action.
Also they indirectly trigger the views of the respective models: paddle and Ball.

```
public abstract class InteractiveComponent
```

This class provides all the methods that must be implemented by `InteractiveBall and InteractivePaddle.`

```
public class InteractivePaddle extends
InteractiveComponent
```

This class create a Model by creating a new Paddle object , creates the corresponding controller and the views for Paddle.
It also passes the values of Dimensions, color and Initial velocities to the model. The Update () method calls controlComponent () to control the Model, like its position, Speed, direction. Then, it notifies or update all observers attached to the Paddle.

Similar functionality is done by the `public class InteractiveBall extends InteractiveComponent` for the Ball model
The interface `public interface Controller` provides all the necessary methods for Ball and Paddle controller.

```
public class PaddleController implements Controller
```

The two most important methods in this class are `handleEvent ()` and `controlComponent ().`The `handleEvent ()` processes all the key board events and moves the paddle object accordingly. The `controlComponent ()` ensures that the paddle does not move beyond screen edges.

```
public class BallController implements Controller
```

Similarly this class handles the keyboard events when the pressed Space bar key is released. This is used for detecting the game start and game end status and correspondingly set the speed, positions of the ball The `controlComponent()` is responsible for bouncing the ball back from the bottom part of screen.
The `checkPaddleBallColllison_n_TakeAction ()` is responsible for checking the collisions between paddle and ball and bouncing back the ball. It also handles the scoring and version of the game.

Display the Paddle, Ball, Score, version and game status on the Screen:

The `update ()` method of `class PaddleViewUpdate` is responsible for Displaying the Paddle on the Screen. It gets the data form the Paddle object and updates on the screen. This is done continuously in a loop.

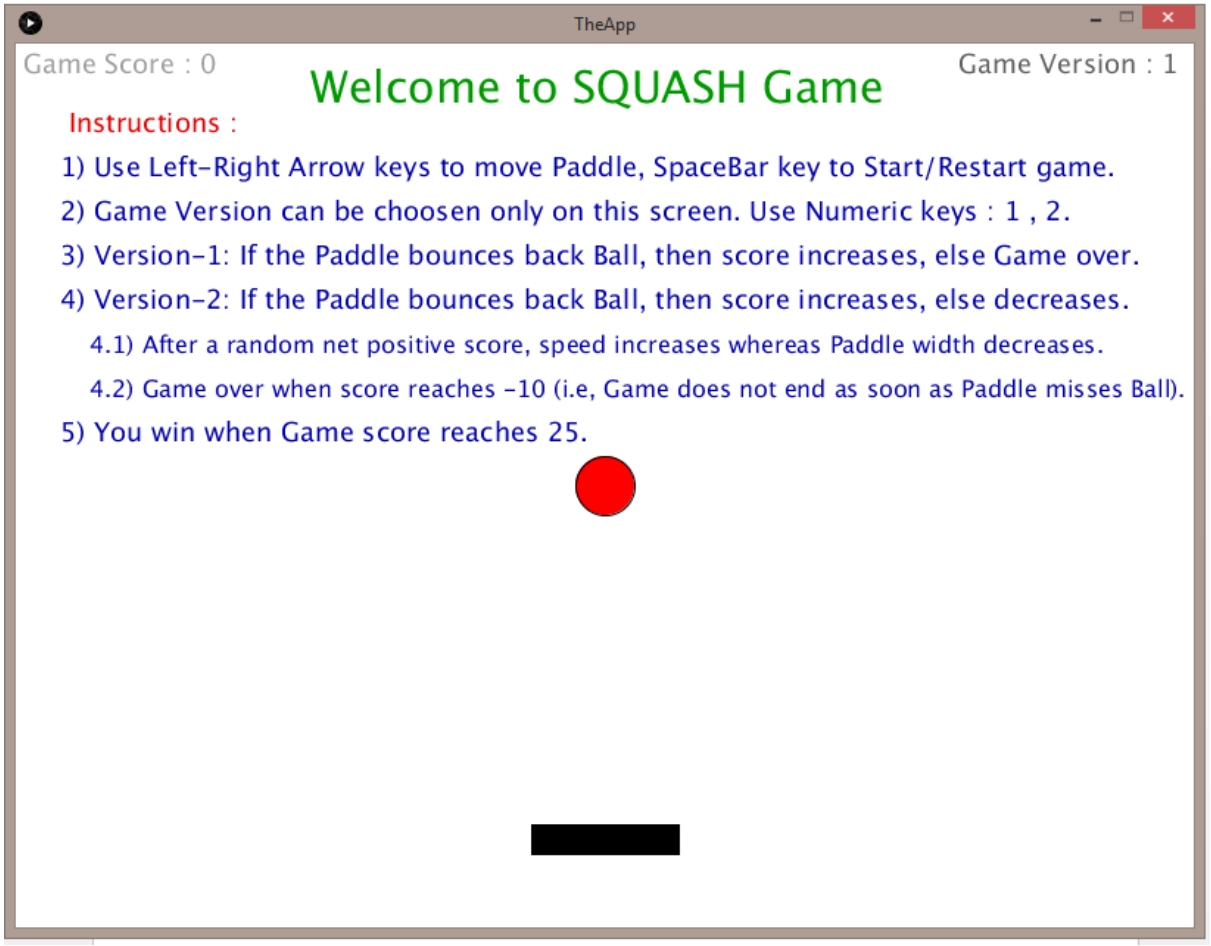
Similar functionality is performed by the update method of `class BallViewUpdate`

The `public class ViewGameStatusAndScore is` the view / observer for updating status of the game. It displays the home screen instructions, the current score of game, the selected version and the game over screen. The control information is obtained indirectly through `BallController.`

Screen shots shots of the game:

When java application is run the game starts with the below Screen shot-1.

Screen shot : 1

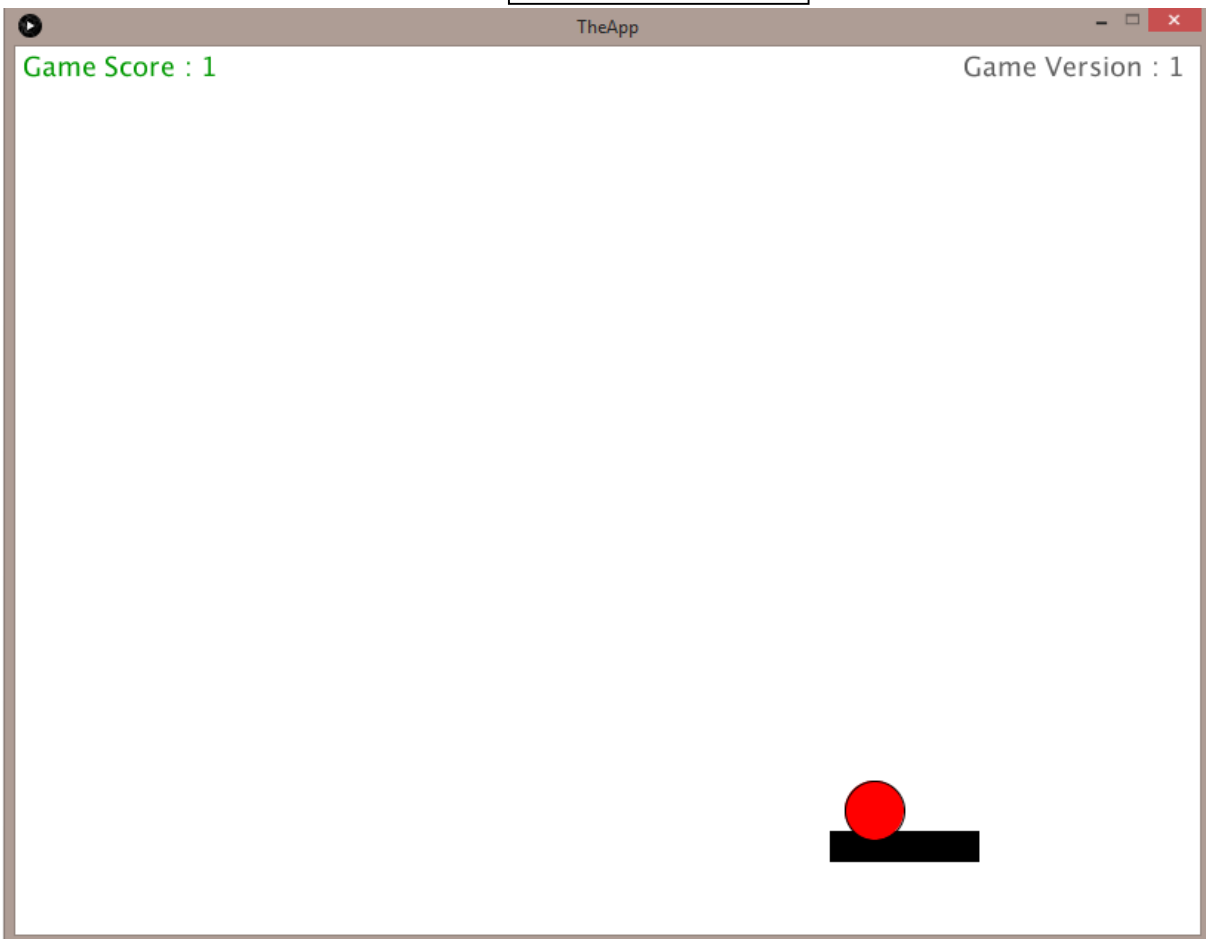


This Initial screen provides all the necessary instructions needed to paly the game. Thus no special manual or document is needed as an additional aid for the players. This screen also allows the player to select the Game versions 1 and 2. By default version -1 is selected.

Game Version-1:

When in Initial screen, the default version is 1 and shown on the top right corner of screen as shown in the Screen shot-2. Pressing the Spacekey bar will cause the movement of the ball.

Screen shot : 2



The player is expected to bounce back the back the ball by using left / right arrow keys of keyboard.

When the paddle bounces back the ball successfully, the Game Score turn green and increments when the the players succeeds in bouncing back the ball with paddle, every time the ball advances towards the bottom of screen.

The game winning score is set to 25. On reaching this score, we get winning message as shown in Screen shot-3.

Screen shot : 3

If in-case, the player misses the ball, then the game is over and ball returns to its initial position as shown in the Screen shot-4.

Screen shot : 4

The interesting feature of this Game is, the user does not have to re – run the applicationm, instead, pressing the space bar key once will lead to initial screen of Screen shot-1. So the player is again allowed to continue with Version -1 or version -2 of the game.

Game Version -2:

If the player selects Game Version -2, in the initial screen, as shown in Screen shot-5, this information is reflected on the top right corner of the screen. This version is more flexible and allows the player to adjust himself / herself to the technicalities of the game.

Screen shot : 5

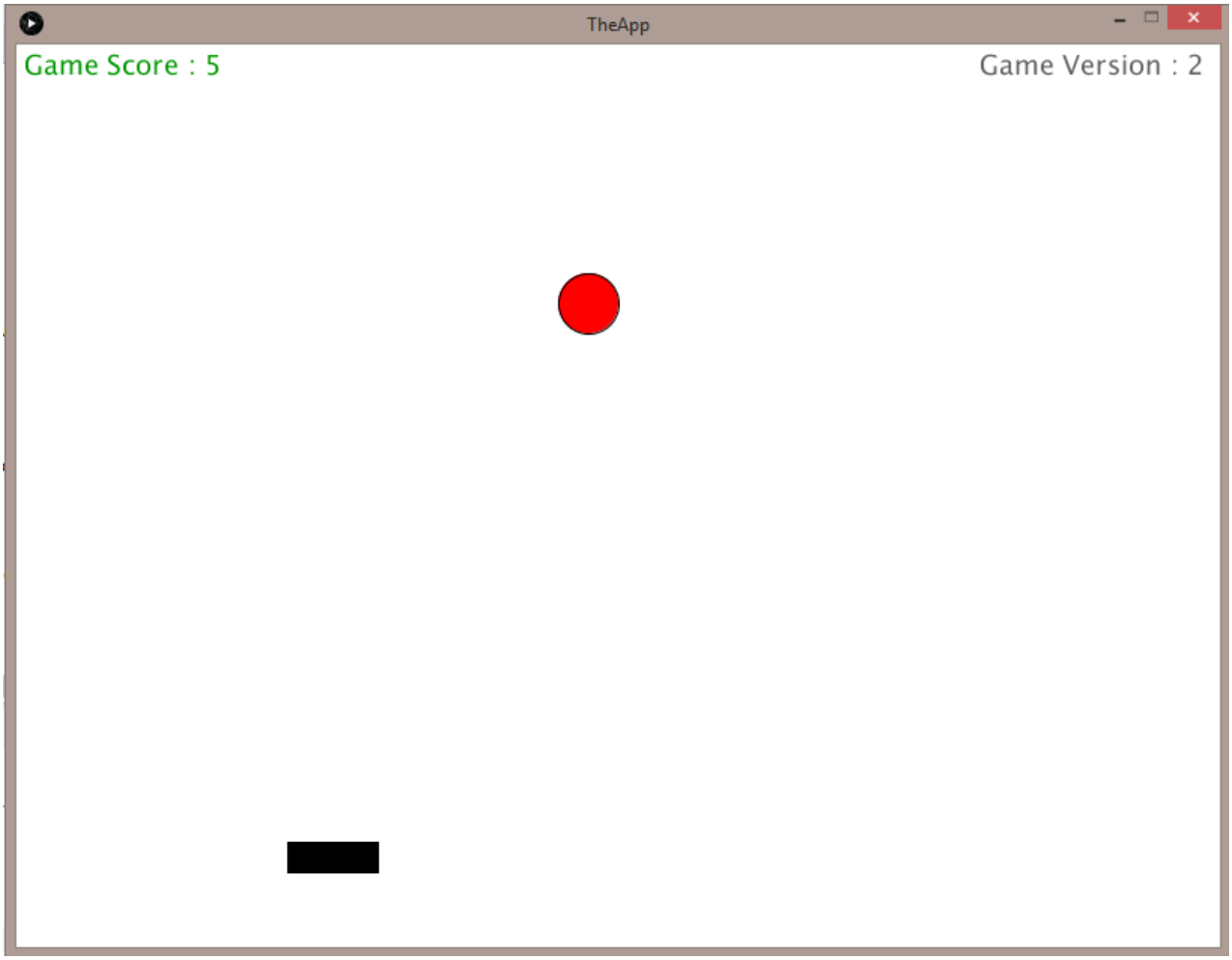
Again, pressing the SpaceBar key will cause the movement of ball and the player is expected to bounce back the back the ball by using left / right arrow keys of keyboard.

If in case the paddle misses the ball, the Game is **not over** but the score decrements. Every successful bounce back increments the score.

But when positive score, reaches a random value between 5 and 10, the game speed increases i.e. ball and paddle velocity increases whereas the paddle width decreases making the game harder for the player.

This is shown in Screen shot-6.

Screen shot : 6



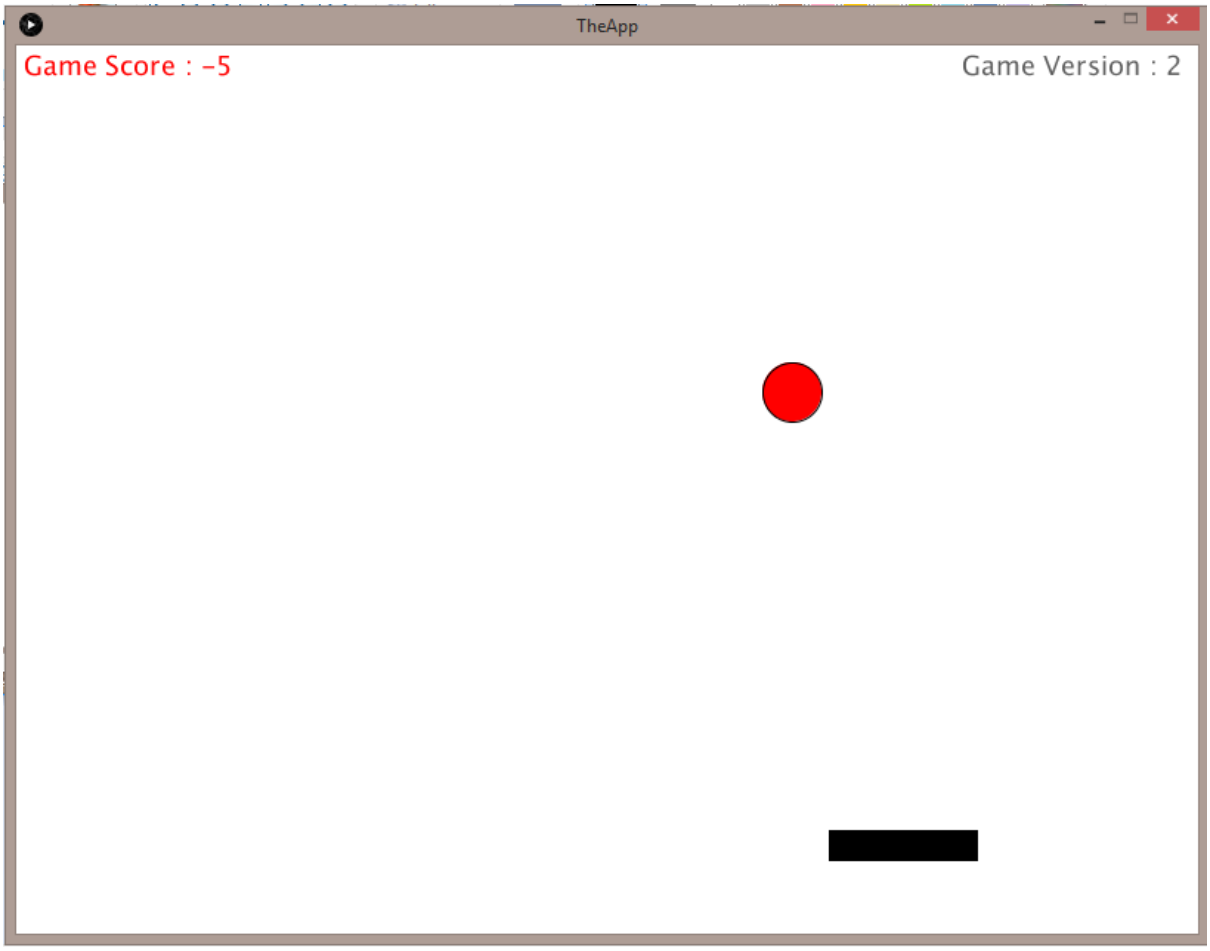
On reaching the winning score of 25, the Screen shot-7 is displayed

Screen shot : 7

The player also gets his / her game statistics like number of hits, misses and the Hit ratio. Pressing space bar again takes the player to Initial screen of Screen shot-1.

The Screen shot-8 shows the situation when the number of misses is greater than number of hits. i.e. the score goes negative.

Screen shot : 8



The game is lost or finished when the score reaches a value of -10. This is shown in Screen shot-9. Here too the player gets to view his / her game statistics like number of hits, misses and the Hit ratio.

Screen shot : 9

Pressing the spaceBar key takes the player to initial screen where the player is presented with all the options and instructions.

The UML Diagrams of Squash game

The figure 1 shows only Class level UML. It shows all classes involved in one Page.

The figure 2 (is same as figure-2) but it shows both the Class level and Instance level diagrams

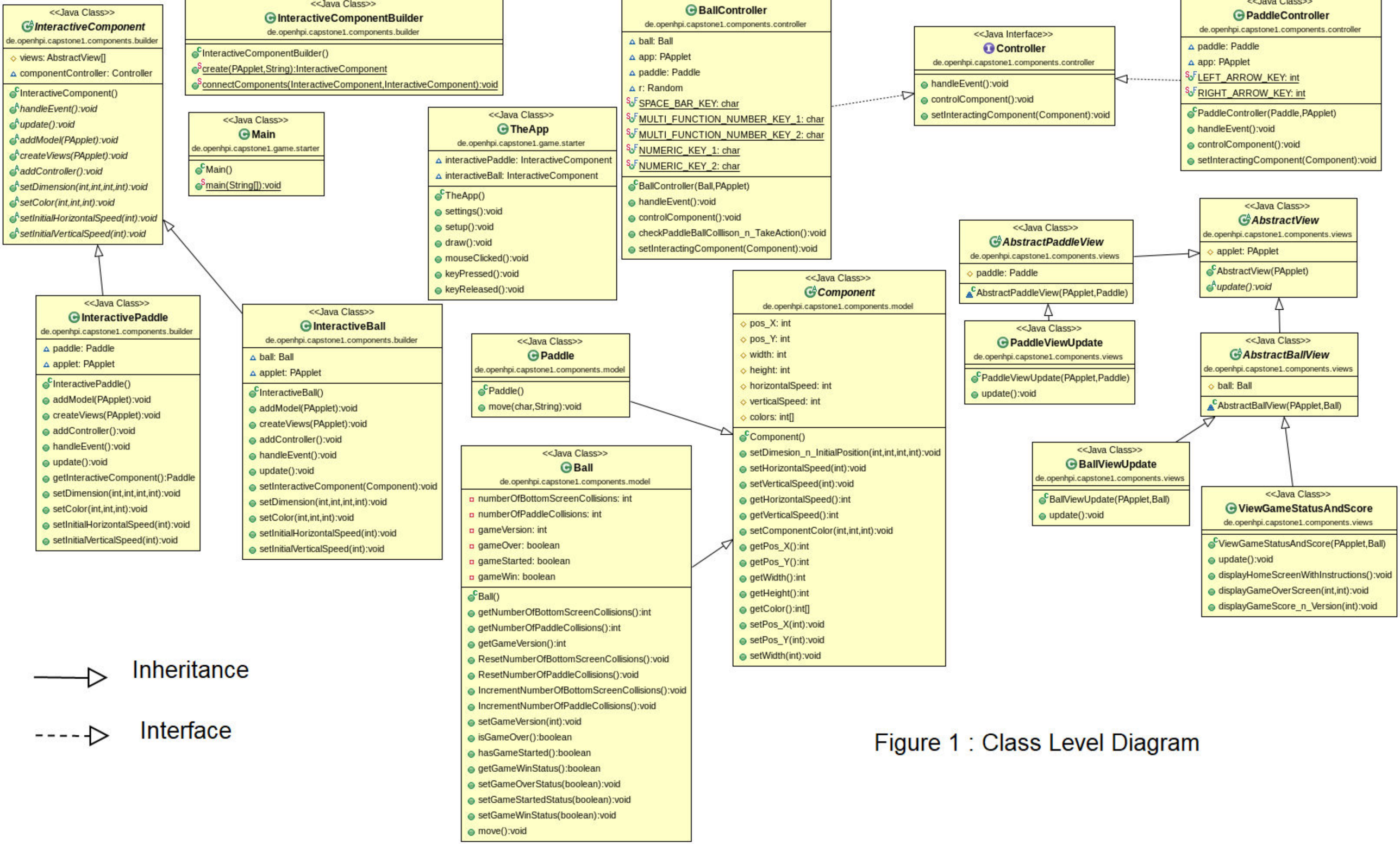


Figure 1 : Class Level Diagram

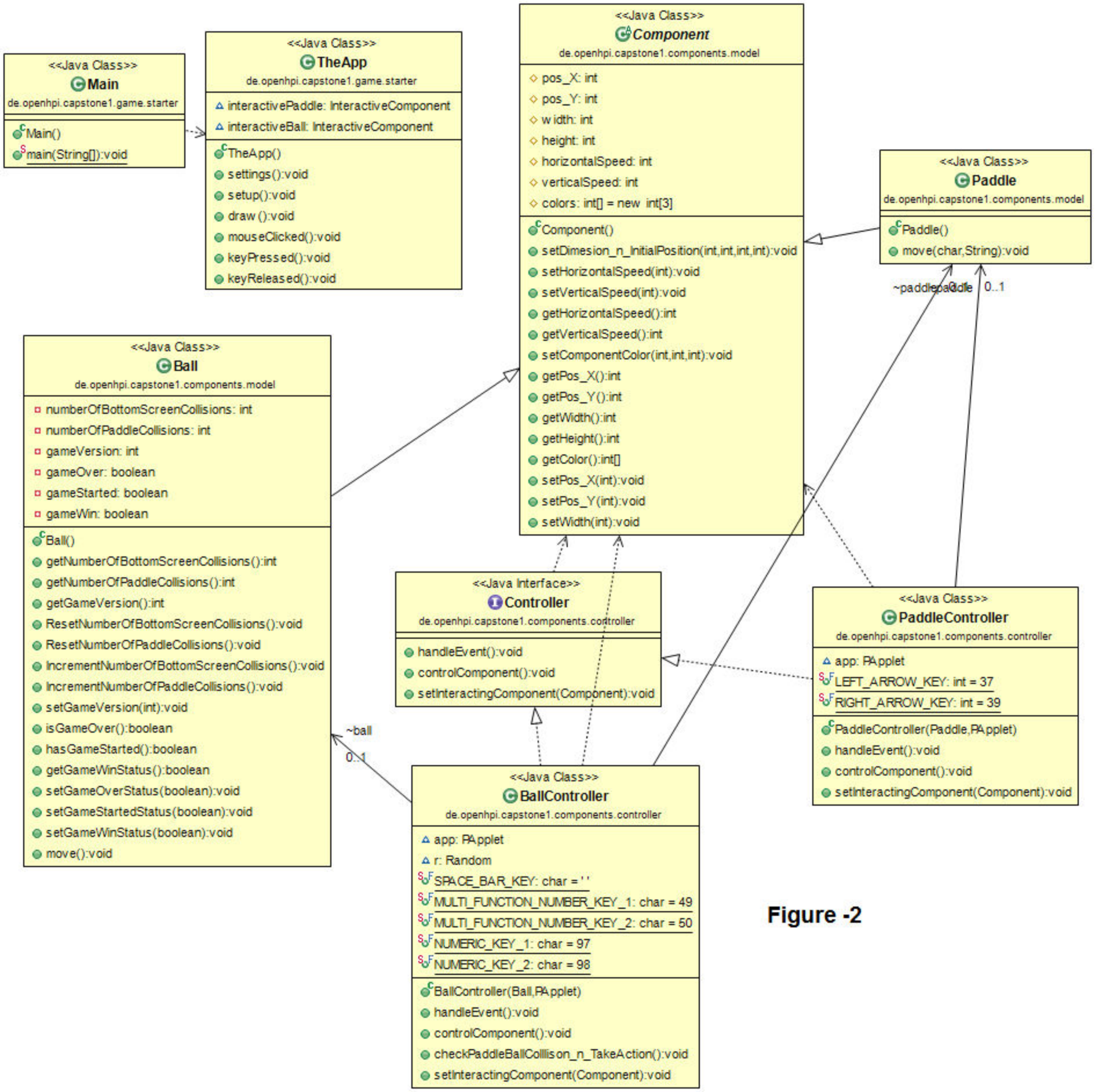

<<Java Class>>
Main
de.openhpi.capstone1.game.starter
Main()
main(String[]):void
<<Java Class>>
TheApp
de.openhpi.capstone1.game.starter
interactivePaddle: InteractiveComponent
interactiveBall: InteractiveComponent
TheApp()
settings():void
setup():void
draw():void
mouseClicked():void
keyPressed():void
keyReleased():void
<<Java Class>>
Component
de.openhpi.capstone1.components.model
pos_X: int
pos_Y: int
width: int
height: int
horizontalSpeed: int
verticalSpeed: int
colors: int[] = new int[3]
Component()
setDimesion_n_InitialPosition(int,int,int,int):void
setHorizontalSpeed(int):void
setVerticalSpeed(int):void
getHorizontalSpeed():int
getVerticalSpeed():int
setComponentColor(int,int,int):void
getPos_X():int
getPos_Y():int
getWidth():int
getHeight():int
getColor():int[]
setPos_X(int):void
setPos_Y(int):void
setWidth(int):void
<<Java Class>>
Paddle
de.openhpi.capstone1.components.model
Paddle()
move(char,String):void
~paddle
0..1
0..1
<<Java Class>>
Ball
de.openhpi.capstone1.components.model
numberOfBottomScreenCollisions: int
numberOfPaddleCollisions: int
gameVersion: int
gameOver: boolean
gameStarted: boolean
gameWin: boolean
Ball()
getNumberOfBottomScreenCollisions():int
getNumberOfPaddleCollisions():int
getGameVersion():int
ResetNumberOfBottomScreenCollisions():void
ResetNumberOfPaddleCollisions():void
IncrementNumberOfBottomScreenCollisions():void
IncrementNumberOfPaddleCollisions():void
setGameVersion(int):void
isGameOver():boolean
hasGameStarted():boolean
getGameWinStatus():boolean
setGameOverStatus(boolean):void
setGameStartedStatus(boolean):void
setGameWinStatus(boolean):void
move():void
<<Java Interface>>
Controller
de.openhpi.capstone1.components.controller
handleEvent():void
controlComponent():void
setInteractingComponent(Component):void
<<Java Class>>
PaddleController
de.openhpi.capstone1.components.controller
app: PApplet
LEFT_ARROW_KEY: int = 37
RIGHT_ARROW_KEY: int = 39
PaddleController(Paddle,PApplet)
handleEvent():void
controlComponent():void
setInteractingComponent(Component):void
~ball
0..1
<<Java Class>>
BallController
de.openhpi.capstone1.components.controller
app: PApplet
r: Random
SPACE_BAR_KEY: char = ' '
MULTI_FUNCTION_NUMBER_KEY_1: char = 49
MULTI_FUNCTION_NUMBER_KEY_2: char = 50
NUMERIC_KEY_1: char = 97
NUMERIC_KEY_2: char = 98
BallController(Ball,PApplet)
handleEvent():void
controlComponent():void
checkPaddleBallCollison_n_TakeAction():void
setInteractingComponent(Component):void

Figure -2

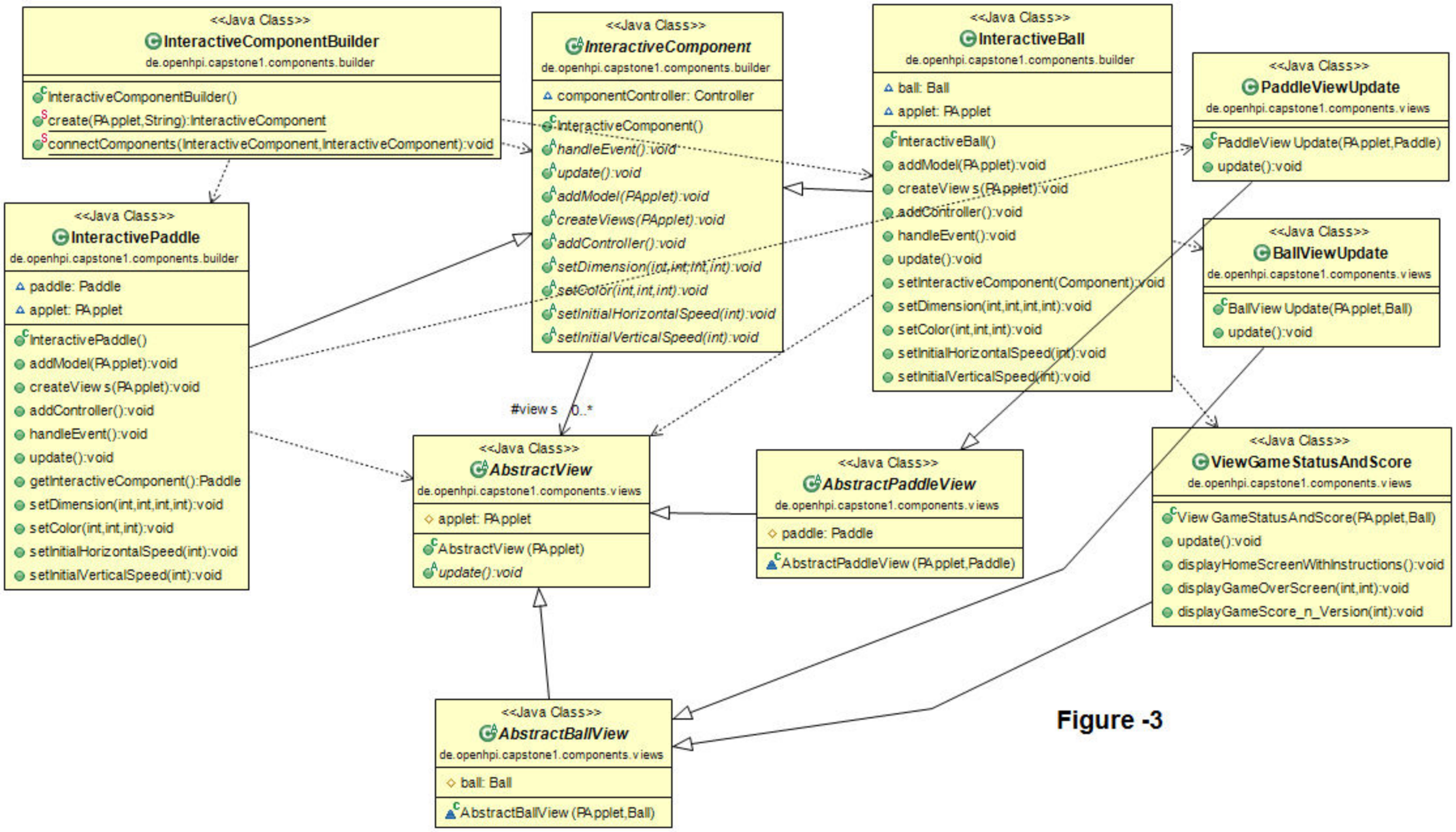

<<Java Class>>
InteractiveComponentBuilder
de.openhpi.capstone1.components.builder
InteractiveComponentBuilder()
create(PApplet,String):InteractiveComponent
connectComponents(InteractiveComponent,InteractiveComponent):void
<<Java Class>>
InteractiveComponent
de.openhpi.capstone1.components.builder
componentController: Controller
InteractiveComponent()
handleEvent():void
update():void
addModel(PApplet):void
createViews(PApplet):void
addController():void
setDimension(int,int,int,int):void
setColor(int,int,int):void
setInitialHorizontalSpeed(int):void
setInitialVerticalSpeed(int):void
<<Java Class>>
InteractiveBall
de.openhpi.capstone1.components.builder
ball: Ball
applet: PApplet
InteractiveBall()
addModel(PApplet):void
createViews(PApplet):void
addController():void
handleEvent():void
update():void
setInteractiveComponent(Component):void
setDimension(int,int,int,int):void
setColor(int,int,int):void
setInitialHorizontalSpeed(int):void
setInitialVerticalSpeed(int):void
<<Java Class>>
PaddleViewUpdate
de.openhpi.capstone1.components.views
PaddleView Update(PApplet,Paddle)
update():void
<<Java Class>>
InteractivePaddle
de.openhpi.capstone1.components.builder
paddle: Paddle
applet: PApplet
InteractivePaddle()
addModel(PApplet):void
createViews(PApplet):void
addController():void
handleEvent():void
update():void
getInteractiveComponent():Paddle
setDimension(int,int,int,int):void
setColor(int,int,int):void
setInitialHorizontalSpeed(int):void
setInitialVerticalSpeed(int):void
<<Java Class>>
BallViewUpdate
de.openhpi.capstone1.components.views
BallView Update(PApplet,Ball)
update():void
#views 0..*
<<Java Class>>
AbstractView
de.openhpi.capstone1.components.views
applet: PApplet
AbstractView (PApplet)
update():void
<<Java Class>>
AbstractPaddleView
de.openhpi.capstone1.components.views
paddle: Paddle
AbstractPaddleView (PApplet,Paddle)
<<Java Class>>
ViewGameStatusAndScore
de.openhpi.capstone1.components.views
View GameStatusAndScore(PApplet,Ball)
update():void
displayHomeScreenWithInstructions():void
displayGameOverScreen(int,int):void
displayGameScore_n_Version(int):void
<<Java Class>>
AbstractBallView
de.openhpi.capstone1.components.views
ball: Ball
AbstractBallView (PApplet,Ball)

Figure -3