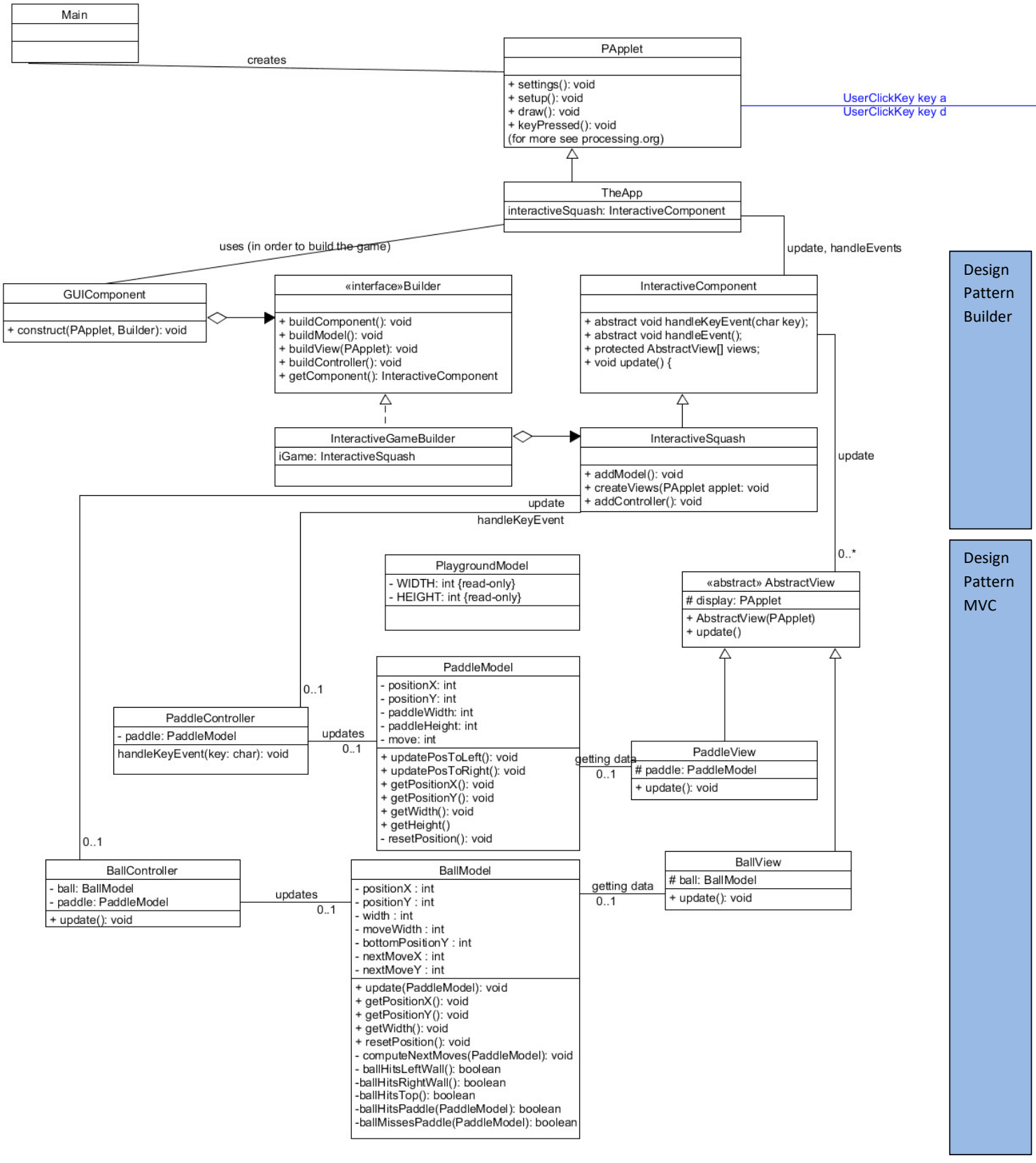
Documentation: Squash Game

UML diagram



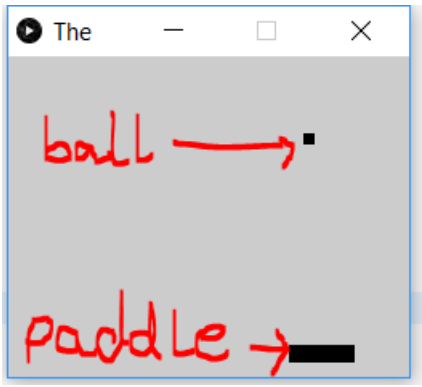
Design Pattern Builder

Design Pattern MVC

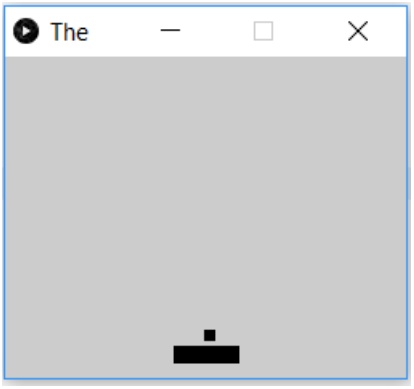
Screen Shots of the Running Program

Start / automatic restart of the program

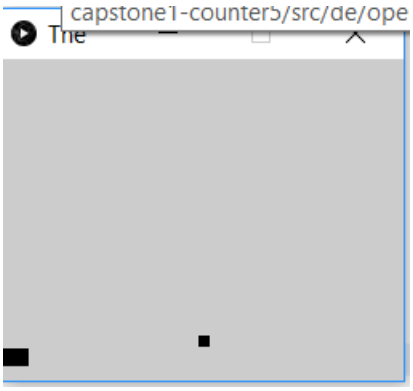
The game consists of a paddle and a ball.



When the program is started, the ball and the paddle are at the bottom, in the middle:



When the ball was missed by the paddle, it appears at once again at the bottom, in the middle and starts to move (this was the fastest solution and time was scarce ☹)



Playing the game

Well, it is the squash game. The goal of the game is that the ball never misses the paddle and gets lost at the lower boarder.

The ball bounces off the left, upper, and right border of the window. It also bounces off the paddle. But it gets “lost”, when it does not hit the paddle and goes to the lower border of the window. In this

case the ball reappears as described above und moves at once. It does start with a randomized direction, as the game should very from one start / restart to the next.

The way it bounces off is according to the law “angle of incidence is equal to angle of reflection”.

The paddle can be moved horizontally by the keyboard keys ‘a’ and ‘d’. The key ‘a’ moves the paddle to the right, the key ‘d’ to the left. If any other key is pressed, the following information is given:

```
press key a to move the paddle to the left
press key d to move the paddle to the right
```

LabReport: Realizing a Squash Game

Working alone it is good when I can finish the Squash Game. Still, I want to program it flexibly, so that I can advance to the other games after the course.

Preparation

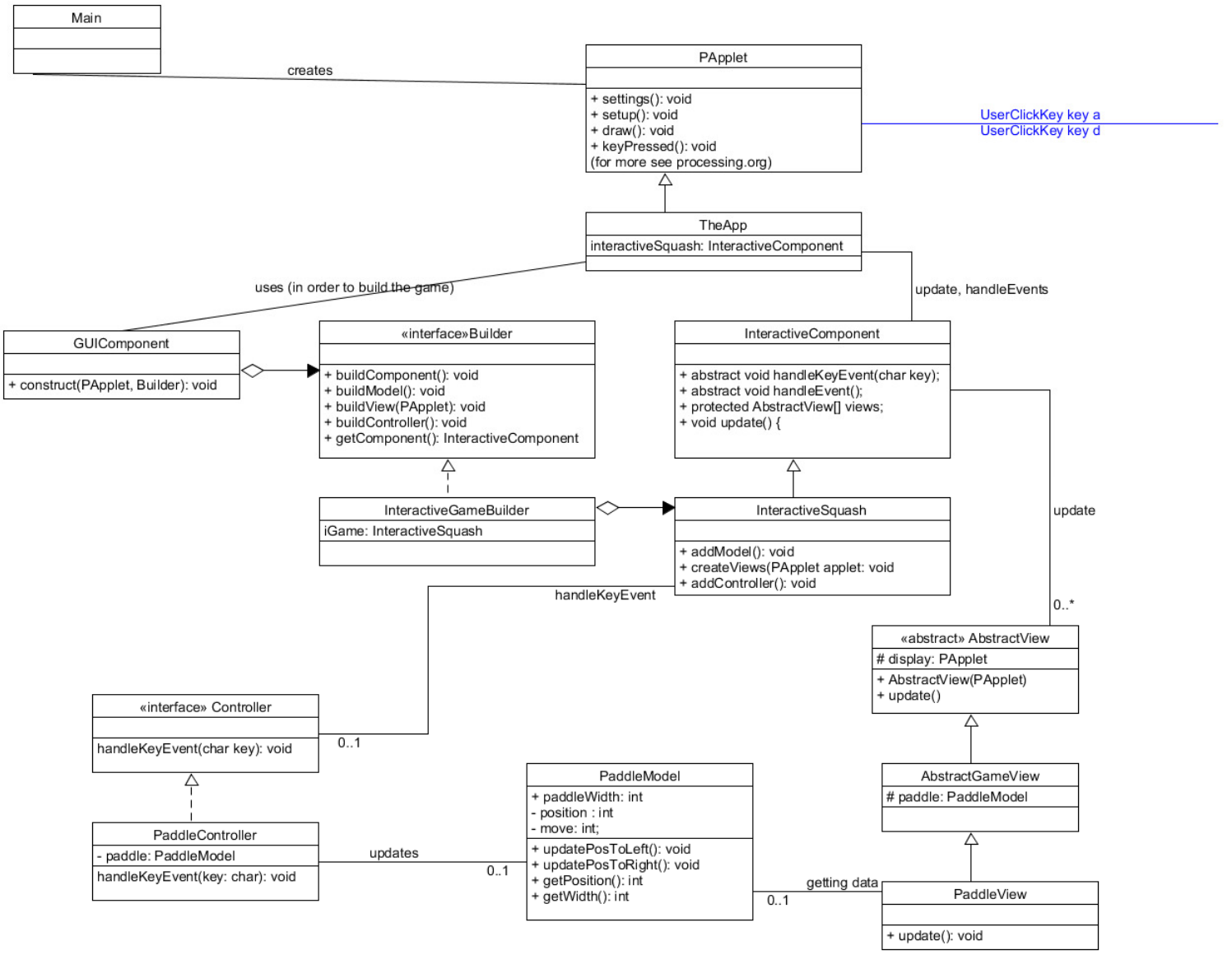
I started with the videos and info pages of the course. The shown MVC model and code was very interesting and I decided that I wanted to use it. I also liked the Builder Pattern of the project counter5 and wanted to use it too.

Most of the trouble I had during the preparation was related to a bad internet connection.

First Step: Builder Pattern and the Paddle

I found it difficult to start with the UML diagram, therefore I decided to break the task into parts. First I wanted to use the Builder Pattern and the MVC Pattern to build up a game that consisted only of the paddle.

After some time I came up with the following UML-diagram:

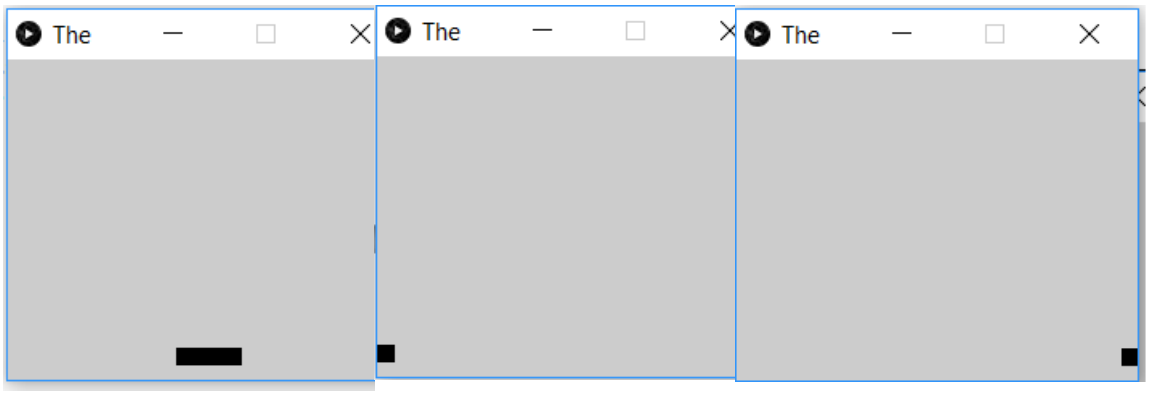


Important was to separate the concerns with Conroller/Model/View. In my program only the model of the paddle should be concerned with real positions. The controller operates on a higer level with “Update position to the right/left. The view pulls the data from the model. The observer pattern is not used as shown in the video about MVC and processing as GUI.

The model also takes care that the paddle is only moved to the left and the right until it can barely be seen. It cannot be moved out of sight.

The next table shows the result:

The paddle is in the middle, when starting the application. The paddle moved to the left as far as possible pressing the key ‘a’ and moved to the right as far as possible pressing the key ‘d’.



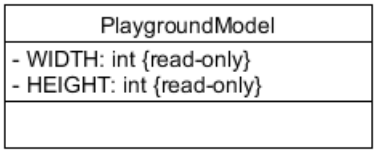
Whan another key than ‘a’ or ‘d’ is pressed the following is printed using `System.out.println()`:

```
press key a to move the paddle to the left
press key d to move the paddle to the right
```

Improving and adding the ball

Adding the playground

Until now I had several locations where I used the width and height of the playground. Not good, if I want to change it, I have to change all these locations. Therefore I introduced a new class with constants and used the constants instead of the current value:



Extending the builder part for the ball

In the builder part I did only have to adapt my Interactive Squash class to handle the ball. There I used the new ballController, ballModel and ballView.

Also the update()-method needed to be adapted. I decided that whenever the draw()-method of the class TheApp is called the ball must update its position. The draw-method already calls the update()-method of my InteractiveSquash class. This update()-method was programmed in the superclass and updates all the views. I added an update()-method to my InteractiveSquash class that still calls the method of the superclass but also the update-method() of my ball controller:

```
public void update() {
	ballController.update();
	super.update();
}
```

Adding the ball to the MVC-part

I deleted the interface controller and the class AbstractGameView. They did not fit the new classes. If I had had more time I would have thought more about whether I could still use them but a running game was more important to me.

I added the ballController, the ballModel and the ballView. The tricky thing was that the ball had to have access to the paddleModel to decide whether the paddle hit the ball or not. Therefore I extended the constructor of the ballController:

```
public BallController(BallModel ball, PaddleModel paddle)
```

When the update came, the BallController called the update()-Method of the class BallModel, passing the PaddleModel paddle.

In the BallModel the next position of the ball depended on the environment, therefore these methods were implemented:

```
boolean ballHitsLeftWall(){…}
boolean ballHitsRightWall(){…}
boolean ballHitsTop(){…}
boolean ballHitsPaddle(PaddleModel paddle) {…}
boolean ballMissesPaddle(PaddleModel paddle) {…}
```

Also the paddle classes were slightly modified.

The game

The squash game is implemented. Whenever the ball misses the paddle and is lost, it is reset to the playground. The start movement is to the right, but the movement is randomized, so that the game varies from one time to the next:

```
nextMoveX = 1 + (int)(Math.random() * moveWidth);
nextMoveY = - 1 -(int)(Math.random() * moveWidth);
```

When the ball hits the wall/paddle it changes the direction using the following law: angle of incidence is equal to angle of reflection. In programming that is easy: if the left wall is hit, the movement in x-direction is inverted from the next move onwards.

The last pitfall

Looking over my project I thought that the method handleEvent() is unnecessary. Actively I only worked with handleKeyEvent(). Therefore I deleted handleEvent() and surprisingly handleKeyÉvent did not work any more. I re-introduced handle Event() and everything worked fine again.

The final UML diagram