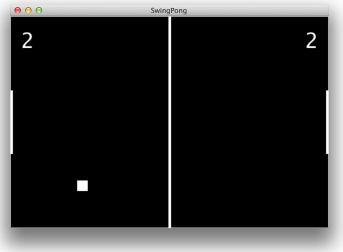
Pong



Pong is one of the oldest computer games – it was released in 1972 by Atari Inc. Unfortunately, there aren't many Ataris around these days, so your assignment is to implement Pong in Java. Since making a game is lots of work (and not all of the required features of Java are covered by the course!), you will get the graphical user interface from us. Feel free to look at our code though: either you're learning something – or we are!

Developing Software in Modules

As you have learned in this course, software is often split up in modules. This allows simpler reasoning about code by keeping the individual concerns separated. Dividing projects up in small(er) modules has many advantages: for one, there can be specialists working on the modules: in this course you won't learn how to build Graphical User Interfaces (GUIs) in Java. Therefore, we brought in the specialists. They proposed an interface that you, the backend-developers have to develop. While you were still busy with the previous course assignments, the specialists sat down and developed the GUI for the Pong application. Now it's your turn to implement the game logic.

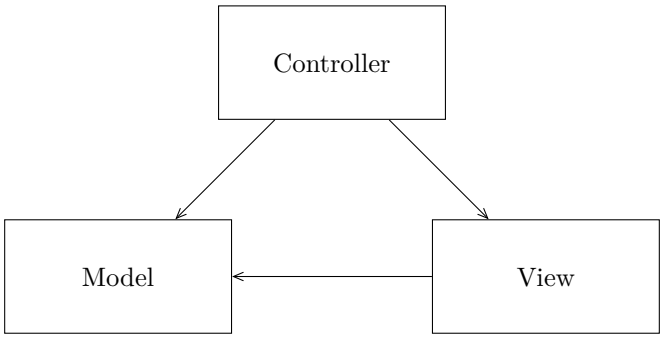
Model-View-Controller

The Model-View-Controller design pattern (short: MVC) is a common way to build graphical user interfaces. There are several flavors of MVC, the one we are using for this assignment looks like this:

The *Model* describes the data of an application. In the case of this assignment, the Model implements the "physics" of the game: where the two bars are, where the ball is, how the ball moves, and so on.

The *View* does nothing but visualizing the Model: it uses the getters of the Model to get all the necessary information – where the two bars are, where the ball is, and so on, and shows this information to the user.

The *Controller* is responsible for user input: it parses the command line arguments, it registers the buttons that the players are pressing and tells the Model when to update its state. Each time the Model is updated, the Controller also tells the View to update the graphics accordingly.



Figur 1: Model-View-Controller.

So, concluding, the Controller first tells the Model to react to the user input and then tells the View to get and draw the updated state from the Model. This allows the programmer to write a model that does not at all depend in the graphics – it would, for instance, be quite easy to develop different Views for different platforms.

Assignment

Implement a model for the game Pong. For this task, read the descriptions in the file `PongModel.java`. This assignment is all about implementing a larger interface on your own; this will involve reading some

of the Java API. You will at least need to use the classes Set, Point and Dimension, – although we recommend looking at how Maps are used in the View – you can use a similar approach to avoid much code duplication in your implementation[1].

We are providing a Makefile and a startup-script that will take care of the rest for you: assuming that you have an implementation of a model, you can use

```
> make
> ./run --framerate=10
```

to execute the program with a target frame rate of 10 frames per second. Even though this might seem low, please use low frame rates when running on the university computers.

Execute

```
> make
> ./run --help
```

to find out about the command line arguments the program will accept and what the default values are.

Features

A full featured game will be playable and fun.

Some of the expected features are:

- There is some way to "aim": if a ball bounces off the top of the bar, its angle will be modified such that a player can – with enough practice – aim for certain directions.
- The game keeps track of scores.
- Losing a point makes the player's bar smaller (or larger, depending on whether you want to be nice or not! :)
- The ball becomes faster each time it touches a bar.
- As soon as one player reaches 10 points, she is declared winner through the message label in the interface. After a couple of seconds, the game resets and a new game with the same players starts.
- Your code runs with the View and Controller we give you! Feel free to tweak our code however you like; as long as your model also works with the original code, we're happy :)

Have Fun!

[1] But also take care: avoiding code duplication alone does not make good code! Sometimes, readability is more important than avoiding duplication – there's no rule without an exception and you should train your instincts: good programmers know which rule to break when!