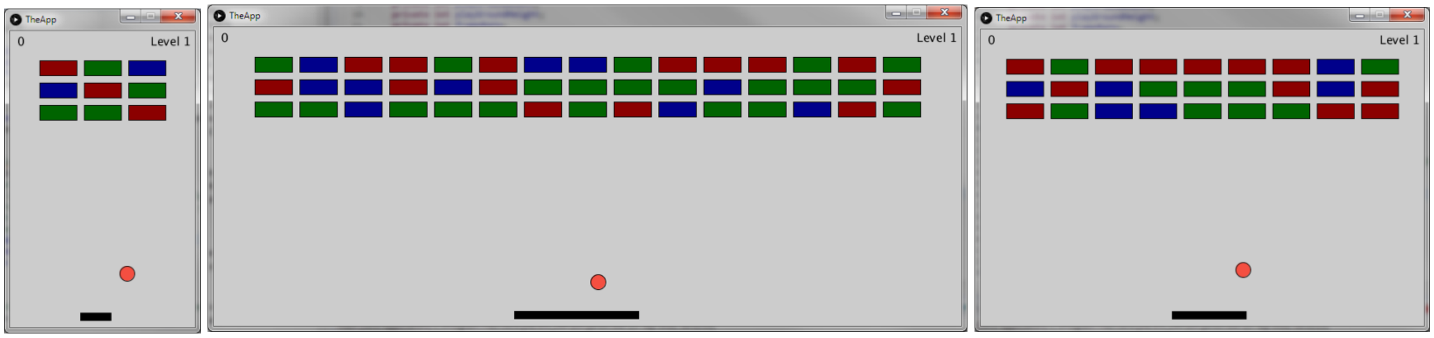
Breakout Game

About the game

Elements: A Ball, a Paddle, and a Wall of Bricks

- Ball: once inserted to the playground, it moves on its own.
  - Ball is released by clicking the Space key.
  - Ball bounces (changes direction) when it hits the left, right or upper bound of the playground.
  - Ball bounces when it hits the paddle
  - Ball leaves the playground when it "hits" the lower bound of the playground
- Paddle: moved by user interaction, Keyboard Left and Right arrow keys or Mouse drag.
- Bricks: they are destroyed when they're hit by the ball.
  - The Red Bricks require 1 hit to be destroyed, the Blue ones require 2 hits and the Green Bricks require 3 hits.
  - The first hit of a brick brings 10 points. Every other hit brings 30 points.
  - Once all the bricks are destroyed a next level starts.
- Level: The Ball speed increases with each level. The score from the previous level is transferred to the next.
- Screen size and frame rate can be adjusted. Go to Model class -> InitializeGameSettings()



->playGroundWidth, playGroundHeight, frameRate. For best game experience we suggest 600/400 screen size.

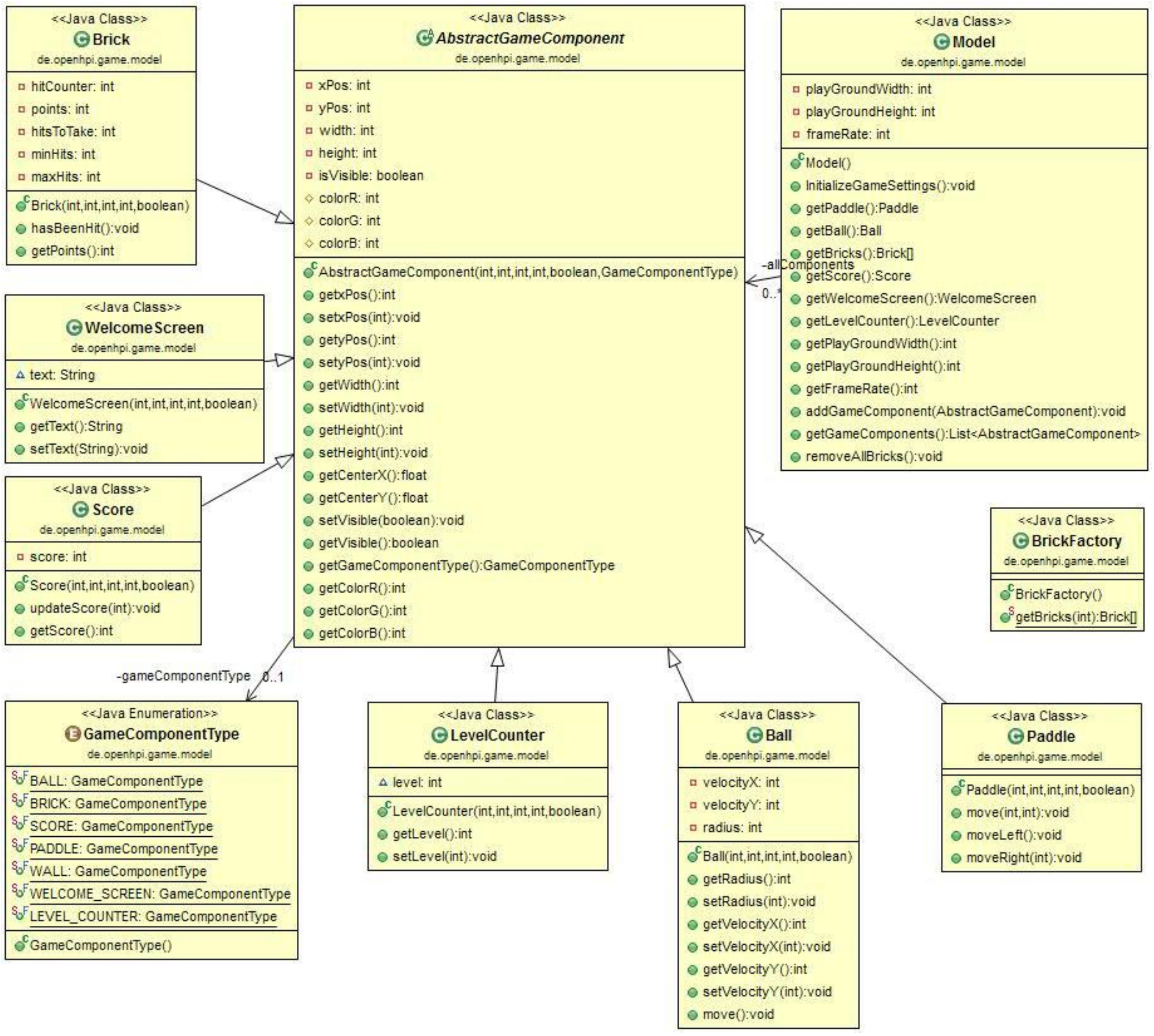
Source code explanation

- The game was created using the Processing library
- As we've decided to go with Processing library, we haven't had the need to implement any "IObservables", because the events we're interested in MouseDragged, KeyPressed and "next iteration in the game" are covered by the mentioned library and already available as overridable methods of the PApplet class. The "next iteration in the game" event normally would be triggered by a timer (java.util.Timer). We've made the decision to use the draw method for that. This method is triggered for each frame and so we were able to handle the application speed by using PApplets framerate() method (Higher frame rate, higher application speed).

- The design of the source code follows the Model-View-Controller pattern and divides the application into three interconnected parts.
  - The model is the central component of the pattern. It is the application's dynamic data structure, independent of the user interface. It directly manages the data, logic and rules of the application.
  - The view is the output representation of information. Multiple views of the same information are possible.
  - The controller, accepts input (key clicks, mouse drag, etc..) and converts it to commands for the model or the view.

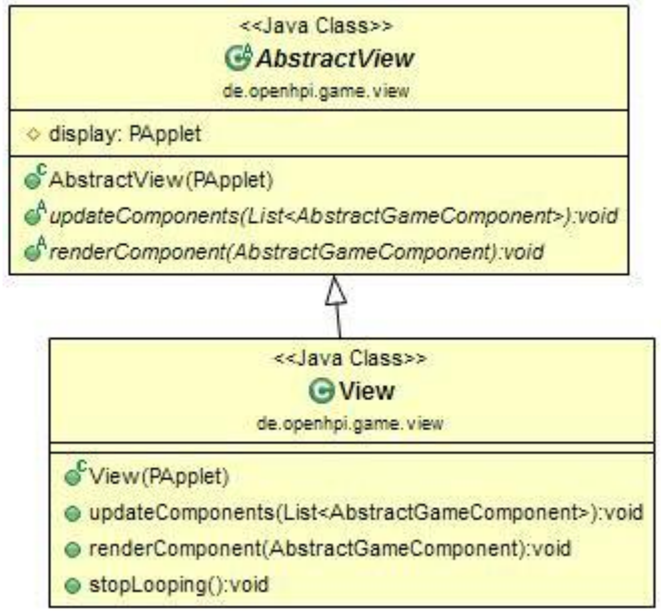
The Model

  - Our model (package de.openhpi.game.model) consists of a number of classes deriving from abstract class “AbstractGameComponent”. For each of the game components (ball, paddle, bricks..) a subclass of AbstractGameComponent exists in the model.
  - The main component in this package is a class named “Model”. The instance of this class helds all instances of AbstractGameComponent. The model can be asked to provide the entire list to a consumer (for rendering the objects for example). If a specific component, like the ball, is required a getter method (e. g. getBall()), that method exists for exposed components.This method identifies the requested type and returns the instance contained in the list already casted to subclasses type.



The View

- The View (package de.openhpi.game.view) is responsible for rendering the information. It has two key methods updateComponent and renderComponent.
- Our view receives a reference to the PApplet instance, when instantiated. It mainly consists of the updateComponents() method, which gets handed over a list of components kept in the model. The method iterates through the list received and renders each component based on the x, y position, width, height, color information and type into the PApplet instance.
- The reason, why we've derived view class from an abstract class "AbstractView" is, that if we had to support an older version of PApplet, or a version for another graphic subsystem (e.g. Android) which for example doesn't have render functions on the level of the current, we could create another view whose updateComponents() method would take in concern to work with some less comfortable/different built-in PApplet rendering. This case clearly is not implemented in our current version.



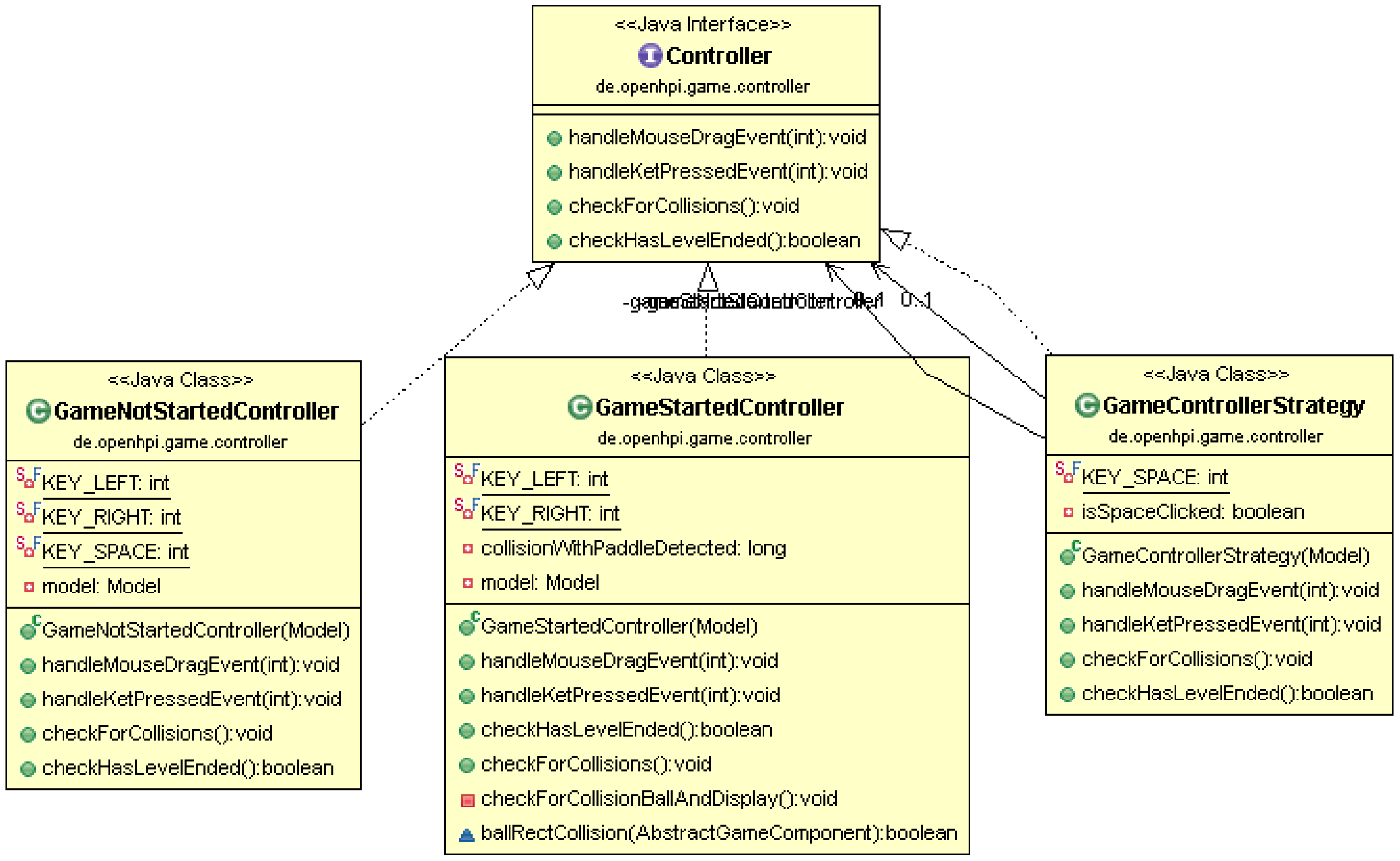
<<Java Class>>
AbstractView
de.openhpi.game.view
display: PApplet
AbstractView(PApplet)
updateComponents(List<AbstractGameComponent>):void
renderComponent(AbstractGameComponent):void
<<Java Class>>
View
de.openhpi.game.view
View(PApplet)
updateComponents(List<AbstractGameComponent>):void
renderComponent(AbstractGameComponent):void
stopLooping():void

The Controller

- The Controller controls the data flow into the Model object. It is also responsible for the collision detection between the objects.
- The Controller implements the Strategy Design Pattern which means that the class/game behavior can change at run time.
- Due to our decision to work with Processing library the classic model view controller roles aren't fully applicable normally the controller gets notified by the view about a user activity (key stroke, mouse drag…). When using PApplet class, the instance of PApplet already handles the mouse- or key events. For this reason we have our BreakOutGame class (which technically is a part of the controller too) being notified by the PApplet event handler methods. The BreakOutGame class instance simply "forwards" the notification to the appropriate controller method. These methods contact the model to update the position of the paddle object for example. In the other direction, a change of state in the model (e.g. paddle has got a new x-position value) would result in a notification of the controller, who itself would trigger an update of the view(s). When using

Processing lib, a redraw is done for each frame, so we had the view to pull data from model for each redraw. This is done, even if no change has happened at all.

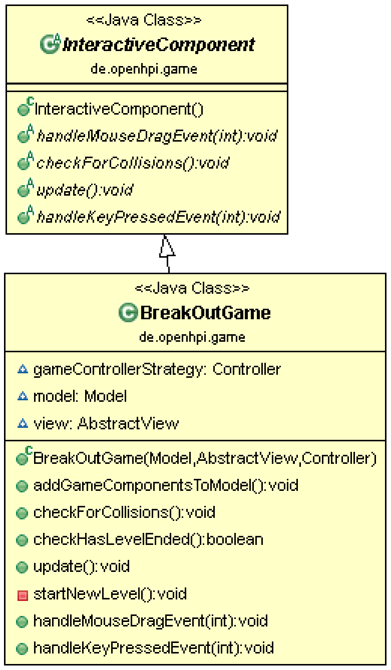
- When PApplets draw method is called, it calls BreakOutGame class' "update()" method. This method tells the model to let have the ball its next move, tells the view to redraw based on the models new values and uses the Controller classes to do the collision checking.



<<Java Interface>>
Controller
de.openhpi.game.controller
handleMouseDragEvent(int):void
handleKetPressedEvent(int):void
checkForCollisions():void
checkHasLevelEnded():boolean
0..1
0..1
<<Java Class>>
GameNotStartedController
de.openhpi.game.controller
KEY_LEFT: int
KEY_RIGHT: int
KEY_SPACE: int
model: Model
GameNotStartedController(Model)
handleMouseDragEvent(int):void
handleKetPressedEvent(int):void
checkForCollisions():void
checkHasLevelEnded():boolean
<<Java Class>>
GameStartedController
de.openhpi.game.controller
KEY_LEFT: int
KEY_RIGHT: int
collisionWithPaddleDetected: long
model: Model
GameStartedController(Model)
handleMouseDragEvent(int):void
handleKetPressedEvent(int):void
checkHasLevelEnded():boolean
checkForCollisions():void
checkForCollisionBallAndDisplay():void
ballRectCollision(AbstractGameComponent):boolean
<<Java Class>>
GameControllerStrategy
de.openhpi.game.controller
KEY_SPACE: int
isSpaceClicked: boolean
GameControllerStrategy(Model)
handleMouseDragEvent(int):void
handleKetPressedEvent(int):void
checkForCollisions():void
checkHasLevelEnded():boolean

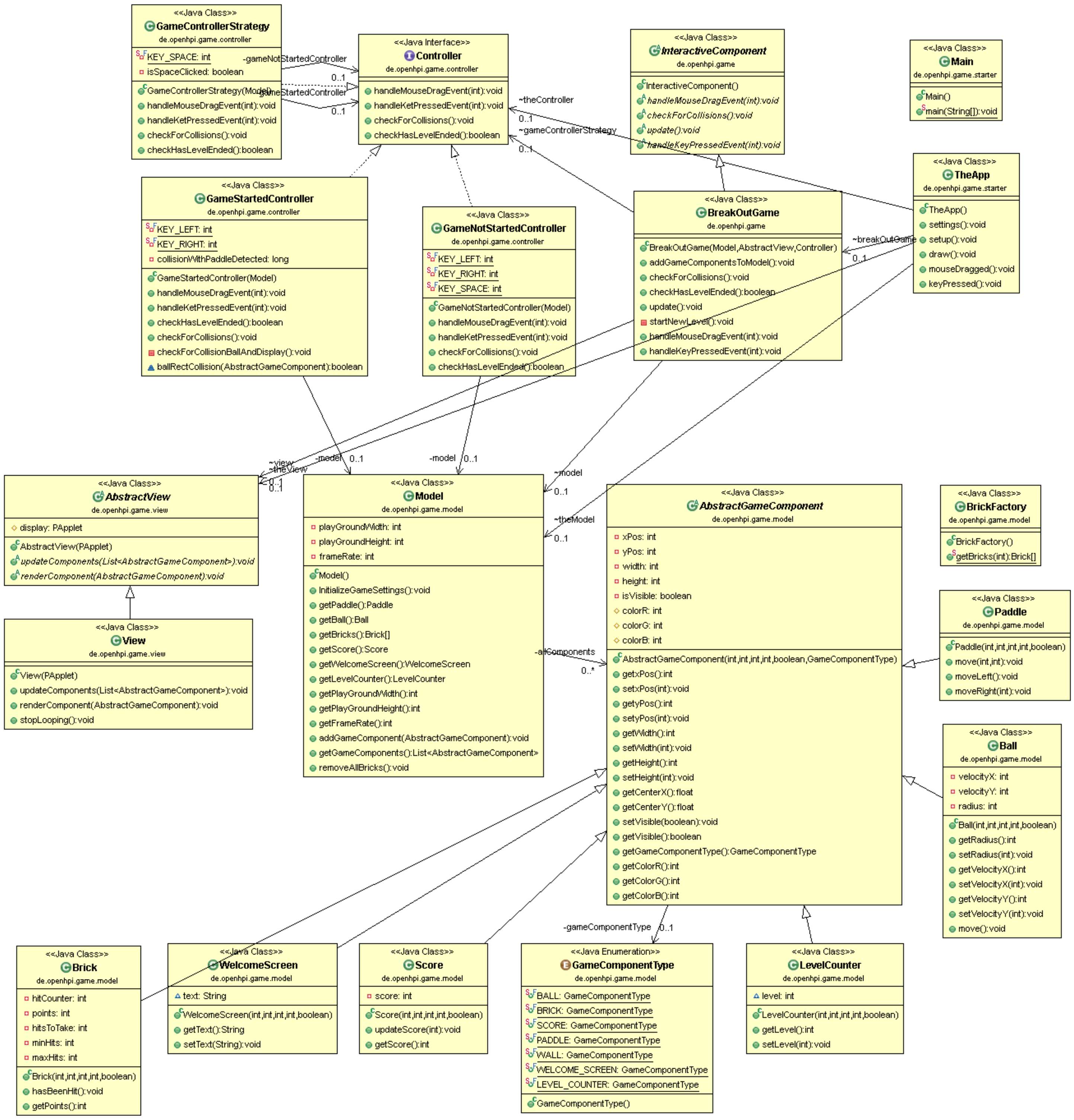
The Builder

- In the de.openhpi.game folder you will find two classes which follow the Builder Design Pattrn. Their responsibility is to make sure that all the necessary components of the game are initialized.



<<Java Class>>
InteractiveComponent
de.openhpi.game
InteractiveComponent()
handleMouseDragEvent(int):void
checkForCollisions():void
update():void
handleKeyPressedEvent(int):void
<<Java Class>>
BreakOutGame
de.openhpi.game
gameControllerStrategy: Controller
model: Model
view: AbstractView
BreakOutGame(Model,AbstractView,Controller)
addGameComponentsToModel():void
checkForCollisions():void
checkHasLevelEnded():boolean
update():void
startNewLevel():void
handleMouseDragEvent(int):void
handleKeyPressedEvent(int):void

The whole picture



Lab Report

A report on how the project evolved, which steps have been taken when, which problems have been encountered, etc.

We spend those almost two weeks of the training as a team trying to utilize our knowledge and experience and share as much as possible among the team members. There was a slow start in the beginning as we all tried to go through the course materials and only then work on the final deliverables. In the beginning we started to develop our own solutions and after the first discussions and review of team members products we decided to go for one of them as our master to work on. The rest of the time was to decide on the scope of final solution and programming. Bese on final scope, the first UML was also produced. Then we started in parallel to document the code and refactor it.

The details of the approach is described in the following subchapters.

How the project evolved

The project was done in few major steps which involved the team work as well as individual work:

1. Self-studies of the team members
2. Discussion on the approach, choice of the game version and initial scope of the game
3. Choice of the most advanced solution to work on
4. Technical discussions on the tools (choice of addons for UML or testing environment)
5. Finalization of the project base on chosen scope
6. Unit testing and fine tuning
7. Refactoring of the objects
8. Documentation (and final refactoring)
9. Submission

Failures and successes

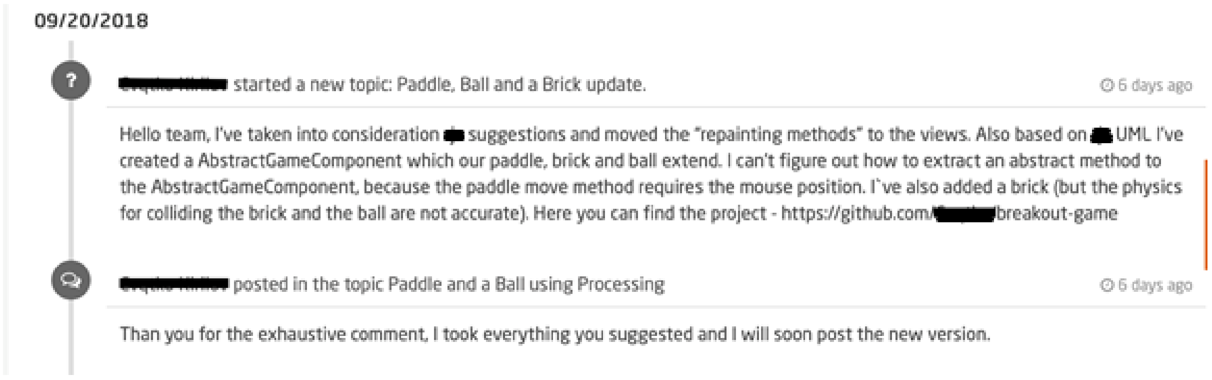
We have experienced several failures **and much more successes** during the training course. Some of them were on personal and some on team level. But we achieved what we agreed upon and finally learned a lot base on our mistakes and hard work.

We would commit that most of our issues were due to the constrains which we faced during the training. They were connected with:

- our experience level (different from person to person),
- technical environments which we were not used to
- time which we were able spend on the project
- work and life environment which caused some of us not being able to join our online meetings
- different analytic capabilities and approach which cause each one of us to approach the project from the different angel

So the major issues which we face were to:

- working as a team and agree on common approach
- use the tools and technology for the first time
- meet and cooperate on timely basis while we all had some other agenda
- technical issues to connect when we were agreed on time and the medium to meet virtually
- agreement on the way program is written and what classes/methods to use (see the screenshot of part of our discussion)
- limit the scope to the level which is achievable and manageable
- find the time for documentation and paper work as we all wanted to extend and extend the functionality
- find time for unit and integration testing


09/20/2018

[redacted] started a new topic: Paddle, Ball and a Brick update. 6 days ago

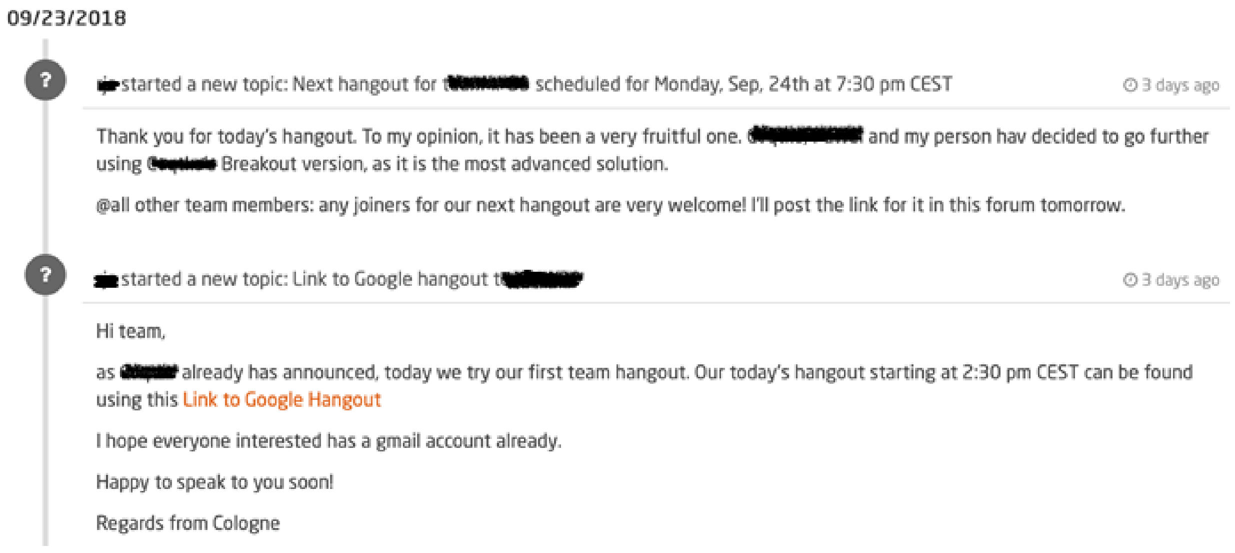
Hello team, I've taken into consideration [redacted] suggestions and moved the "repainting methods" to the views. Also based on [redacted] UML I've created a AbstractGameComponent which our paddle, brick and ball extend. I can't figure out how to extract an abstract method to the AbstractGameComponent, because the paddle move method requires the mouse position. I`ve also added a brick (but the physics for colliding the brick and the ball are not accurate). Here you can find the project - https://github.com/[redacted]breakout-game

[redacted] posted in the topic Paddle and a Ball using Processing 6 days ago

Than you for the exhaustive comment, I took everything you suggested and I will soon post the new version.

the successes which we want to highlight are the following:

- great learning for all team members so we learned a lot and created our own solutions
- testing with the examples provided in the training so we had to analyze, understand and extend the code provided and create a new one base on new knowledge
- new tools usage such us: Eclipse, UML generator, gitHub
- great experience in team work so we were able to agree, cooperate and deliver the expected output using our strengths and knowledge
- common success of achieving the expected deliverables
- learning from each other by discussing the case and sharing the model as well as looking at working program and testing it (please see one of our team meeting announcement screenshot)
- meet people from different countries (taking some sun from Bulgaria, cold rain of Warsaw and October Fest feeling from Germany ☺


09/23/2018

[redacted] started a new topic: Next hangout for [redacted] scheduled for Monday, Sep, 24th at 7:30 pm CEST 3 days ago

Thank you for today's hangout. To my opinion, it has been a very fruitful one. [redacted] and my person hav decided to go further using [redacted] Breakout version, as it is the most advanced solution.

@all other team members: any joiners for our next hangout are very welcome! I'll post the link for it in this forum tomorrow.

[redacted] started a new topic: Link to Google hangout [redacted] 3 days ago

Hi team,

as [redacted] already has announced, today we try our first team hangout. Our today's hangout starting at 2:30 pm CEST can be found using this Link to Google Hangout

I hope everyone interested has a gmail account already.

Happy to speak to you soon!

Regards from Cologne

Discussions and interactions with other participants

There were two major steps in our cooperation as a team:

- Personal work on the tasks decided via forum discussions
- Team work during on-line discussions

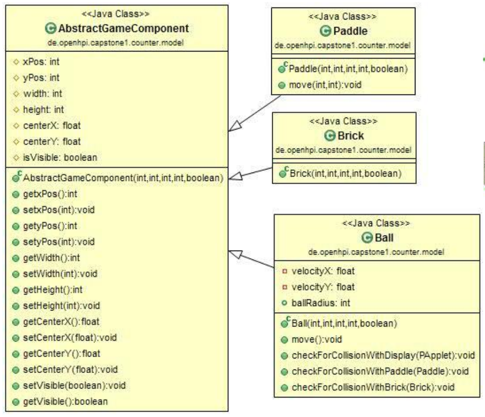
We used almost all available medias to communicate and exchange our thoughts:

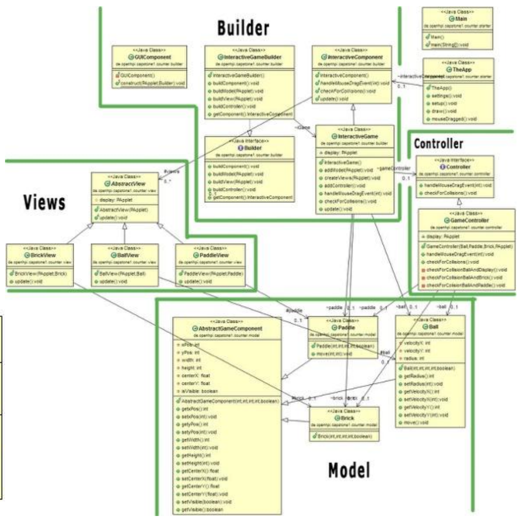
- Forum in open.hpi
- File sharing
- GitHub
- Emails
- Hangouts meeting with screen sharing

Day 1 (14.IX): We started as independent people to go through the course and also exercising with the provided examples. Just few minutes after the training started, we had one of our colleagues saying Hallo to us. We introduced ourselves quickly and muted for some time.

Week 1 (14.IX-21.IX): For next week we exchanged our understanding of the tasks and what we need to do. We also started to share our first code and giving access to our gitHubs to understand who achieved what. During that week we also realized that we are on completely different level of knowledge and experience, so we decided to focus on one of the most advanced solution while the rest of the solutions were left for their owners.

At the end of week 1 we also created our first UML diagram showing the core components of the game.





The work accelerated, and the diagram grow up to some more detailed one day by day. And it finalized in the form presented at the beginning of the document.

Beginning of Week 2 (22.IX-24.IX)

We decided for the first meeting with the participants using Hangouts. There were 3 meetings on which we:

- Discussed the final scope
- Reviewed WiP and decided where to stop it and where to extend
- Divided the roles
- Discussed weather conditions on our locations

The meetings took place on Sunday, 14.30 CET, Monday 19.30 CET, Tuesday, 21.00 CET

Between the meetings we shared our work via forums by posting there our remarks, documents and links to GitHub model.

End of Week 2 (25.IX-27.IX)

The final week (and rather the part of the week (just 3 days) were devoted to finalise the description as requested by the training team as well as to … finalize the code and … refactoring of the classes and their methods. It was ongoing work which might be done almost forever. The major reason we did it was to:

- add more comments to make it self-explanatory without documentation
- clear the methods which are not fully used
- use interfaces to make the relation clear and reusable
- change behavior of the objects

but the final day was mainly focused on

- filling the report and documenting the software and user manual
- updating UML
- doing final check (last call), Wed, 21.00 CET
- submitting the deliverables

We were heading to submit our work on 26th of September as we were afraid that the time zones differences might be misleading and put us in the risk to be late on the last day ☺.