# Interval Racer System Design Overview

Contents

[Interval Racer System Design Overview 1](#_Toc450846488)

[Software Stack 2](#_Toc450846489)

[Reasoning 2](#_Toc450846490)

[Global System Description 3](#_Toc450846491)

[The RaceEngine 3](#_Toc450846492)

[The Race Model 4](#_Toc450846493)

[Player and RaceCar 4](#_Toc450846494)

[Race Configuration 4](#_Toc450846495)

[The Race Track 5](#_Toc450846496)

[Race Statistics 5](#_Toc450846497)

[Page navigation 5](#_Toc450846498)

[Updating The Race View 6](#_Toc450846499)

# Software Stack

For the assignment the following software stack is used:

* Java
* JSF
* Javascript
* HTML
* Omnifaces
* GSon
* Glassfish
* Eclipse
  + EclEmma
  + Findbugs
  + Checkstyle
* JUnit
* Mockito
* Maven

## Reasoning

Java was required. Javascript and HTML are all a given when doing web based solution. Eclipse is my preferred IDE. Junit is used for unittesting and is supported by Mockito for mocking.

Maven is used as a build tool.

JSF; I have very little experience with web frameworks, JSF I have minor training.

Omnifaces; I needed a framework with support for web sockets, JSF 2.3 will support it, however 2.2 at the moment does not. Since OmniFaces has the JSR implementation we can fase it out after JSF 2.3 hits the market.

Gson; Convenience for generating JSON files from Objects.

Deployment is on Glassfish. For convenience reasons I wanted the full support of an Enterprise server. Support for web sockets is out of the box then.

# Global System Description

Before the start of the race players can join and the race can be configured on the first two pages of the application. On the start of the race the RaceEngine is started and the players are redirected to the raceview. The RaceEngine starts processing each clock cycle the players commands and the effects on the race by updating the Race model. After each clock cycle the players are notified to update their race.

Players join the game through the Join page, their information is stored in a SessionScoped bean throughout the game. The Race itself is stored inside an ApplicationScoped bean. This means that there can be only one race game at a time. After each Race the players sessions are discarded and players are redirected to the Join page.

# The RaceEngine

The following sequence diagram shows the steps of the RaceEngine. The engine processes clockcycles which are triggered using a Timer. The engine registers its self as a TimerTask and schedules it at a predefined interval.

Each clockcycle the RaceEngine processes 5 steps:

* Evaluate player commands
* Translate RaceCars
* Evaluate checkpoints and finish
* Check race duration
* Notify clients

The first three steps update the Race model. This is left out of the sequence diagram for readability.

# The Race Model

The RaceEngine uses the race model to process and store players and game data. The class diagram below shows a simplified version of the race model, only class names are given for readability. For further description see the JavaDoc in the class files.



## Player and RaceCar

The player class provides the name of the player, a Boolean if the player is racing and the RaceCar of the player. It also stores the last command the player has given, which is used in the RaceEngine.

The RaceCar describes the color, state, startingNr, speed, direction and position of the RaceCar. The Race Engine manipulates the RaceCar through its functional interface which comprises of the following commands:

* turnLeft
* turnRight
* increaseSpeed
* decreaseSpeed

The RaceCar has minor logic for determining maximum and minimum speeds.

## Race Configuration

The Race config holds the configuration for the Race. It holds the following configuration items:

* numbers of laps that have to be driven
* time interval for the clock cycles of the Race Engine
* maximum race duration

## The Race Track

The race track is build using track elements, the track elements together describe the track. The track elements and race track are smart objects that contain logic. The race track has two methods that provide information about the track. First it provides the available starting positions on the track. Secondly it gives the track element for a position on the track.

There can be different types of track elements, at the moment the following have been provided:

* HorizontalTrackElement
* VerticalTrackElement
* SoftCurveTrackElement

Using these three elements all kinds of tracks can be created. Track elements provide an interface which enables the race engine to determine the type of road a RaceCar is racing on and if the element is a checkpoint of finish.

The SoftCurveTrackElement has a rotation angle so that it can be rotated to form 4 different corners in a 90 degree angles

## Race Statistics

The race statistics object is used to store statistics about the race. It is split up into Race, player and lap statistics. Using a functional interface the statistic information is stored into the model. This is done during the clock cycles of the race engine. The statistics object is parsed on the statistics page to show the statistics and provides a to JSON method for easy storage on the file system

# Page navigation

The default flow through the application is shown below. The default flow goes from the Join page to the WaitRoom. From there to the RaceView and when the race is finished go to the statistics page. After the statistics have shown go back to the Join page. In order to do this all players sessions are invalidated. Alternative flow is from the waitroom to the configuration page where the race configuration can be altered.

  
To control the user flow and capture on authorized page visits each page checks if the user viewing is on the correct page. This is done through a JSF prerendered view call to the session bean PlayerManger. Here is determined if the player can view this page at the moment. If not the player is redirected to an appropriate page. This is done according to the following matrix

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Status** | Player not Joined | Player Joined | Racing | RaceEngine Stop |
| **Pages** |  |  |  |  |
| Join | - | WaitRoom | RaceView | - |
| WaitRoom | Join | - | RaceView | - |
| Config | Join | - | RaceView | - |
| RaceView | Join | - | - | Statistics |

## Updating The Race View

This is the head page of the application. In the race view a HTML5 Canvas is used to draw the race track and the players cars on it, to do this Javascript is used. The data for the position of the RaceCars comes from the Race model and is initially parsed within the JSF page. The update cycle is done using a JSON message through a web socket. The server pushes each clock cycle all players and cars in JSON format to the clients connected on the web socket channel.

DISCLAIMER: At the time of writing, the update of the canvas is not working accordingly. The race track is displayed but the cars are not. In order to show the cars a page reload is initiated following in a flickering race view.