

Group assignment 4: Refined OO model

Próun hugbúnaðar Spring 2015

Students: (Group F2a) Einar Helgi Þrastarson Hannes Pétur Eggertsson Sigurður Birkir Sigurðsson Teachers: Matthias Book Kristín Fjóla Tómasdóttir

1 Introduction

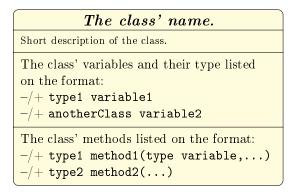
In this document there's the class diagram for group F2a. Group members are: Einar Helgi Prastarson (personal ID number: 110287-2919), Hannes Pétur Eggertsson (240889-2939) and Sigurður Birkir Sigurðsson (120589-2539). Our project is to build an user interface for a fantasy football game. In our class diagram we felt it made sense to split the classes into two categories, back-end classes and front-end classes. Then, in a third diagram there's another diagram that shows the connections between the back-end and We will all present this document on Wednesday, March?th 2015.

1.1 Notation

In our class diagrams we use the following notation:

- means a private variable or method (not directly accessable by other classed).
- + means a public variable or method (directly accessable by other classes).

Each class in the diagram has four sections shown below:



If the class wasn't created by us it is filled with red. Back-end classes are green and front-end classes are yellow. Classes are interconnected using arrows:

Class A
$$\xrightarrow{uses}$$
 Class B

In most cases we can tell how many classes 'Class A' and 'Class B' will be associated with, this is shown by placing an arrow at the beginning and end of an arrow, e.g.

Class A
$$\xrightarrow{1 \quad uses \quad 0-10}$$
 Class B

if each instance of 'Class A' will use 'Class B' in a range of 0 to 10 instances. If no numbers are shown it generally means the association is 1 uses 1.

2 Class diagram

We decided to split our class diagram into two figures: **Back-end classes** and **Front-end classes**. The back-end classes take care of storing and keeping track of all information as the game is running. The front-end classes take care of displaying the information to the users playing the game as well as handling their input.

2.1 Back-end classes

User

This class keeps track of all information about each user playing the game.

- int id
- int money
- int score
- int roundscore
- String name
- Roster roster
- + User(String name, int id)
- + int getMoney()
- + boolean isAffordable(int price)
- + void changeMoney(int dMoney)
- + Roster getRoster()
- + int getScore()
- + int getRoundScore()
- + void setScore(int newscore)
- + String getName()
- + void setName(String newname)

Main Game

The main back-end class. Keeps track of the state of the game. It exists always while the game is running.

- static final MainGame game
- StatsHistory stats
- List<User> users
- int round
- int currentUser
- static MainGame()
- + MainGame getInstance()
- + void setNumUsers(int num)
- + void nextUser()
- + int getRound()
- + List<User> getUsers()
- + StatsHistory getStatsHistory()
- + User getCurrentUser()

ObjectScores

A class with information about each player.

- Object object
- List<Integer> scores
- List<Integer> totalscores
- + ObjectScores(Object object)
- + void addScore(int score)
- + List<Integer> getScores()
- + List<Integer> getTotalScores()
- + Object getObject()

Roster

Keeps track of which football players are in which user team/roster.

- List<Player> goalkeepers
- List<Player> goalkeepersOnField
- List<Player> defenders
- List<Player> defendersOnField
- List<Player> midfielders
- List<Player> midfieldersOnField
- List<Player> forwards
- List<Player> forwardsOnField
- int numberOfPlayersOnField
- + Roster()

1-N

- + int getNumberOfPlayersOnField()
- + boolean removePlayerFromField(Player player)
- + void removePlayerFromRoster(Player player)
- void removePlayer(Player p, boolean fromRoster)
- + boolean addPlayerToField(Player player)
- + boolean addPlayerToRoster(Player player)
- + List < List < Player > getPlayersInRoster()
- + List< List<Player> > getPlayersOnField()
- + boolean isInRoster(Player player)
- + boolean isOnField(Player player)

Player «interface»

This class will be made by group F1a. Each instance will contain information about a football player. It will (at least)git pull have the following instance variables and functions.

- enum Position
- + String getName()
- + Integer getPrice()
- + Position getPosition

StatsHistory

A class that has statistical information.

- List<ObjectScores> allplayerscores
- List<ObjectScores> alluserscores
- List<ObjectScores> allrosterscores
- + StatsHistory()
- + void createPlayerScoreObject(Object player)
- + void createUserScoreObject(Object user)
- + void createRosterScoreObject(Object roster)
- + List<Integer> getPlayerScores(Player player)
- + List<Integer> getUserScores(User user)
- + void addScoreToPlayer(Player player, int score)
- + void addScoreToUser(User user, int score)

N is er number of total users in the current game and P is the total amount of football players in the game.

Front-end classes 2.2MainEndgamePanelStartPanelThe main front-end class. It is initialized StartPanel will be spawned at the very start Pops up after the 10th round. at the start of the game and runs until the of the game. It will ask users to type in their Shows the winner and statistgame is terminated. name before the game begins. ics about the game. - static final Main instance - JPanel center + EndgamePanel() - JFrame frame JTextField field - JPanel right - List<String> names - JPanel change - int numEmpty - MainGame game JButton startGame - static Main() JButton addPlayer ScorePanel+ static Main getInstance() Shows all user's scores and + StartPanel() + void setEndgamePanel() shows GraphDataPanel. + addPlayerHandler() + void startGame() + void changeCenter() + ScorePanel() + void restartFrame() + void setPanelAsScore() MarketPanel+ void setPanelAsMarket() Shows the user the market of players which he/she can buy + void setPanelAsFieldViewer() + void setPanelAsLeague() - final JTextField field + void setPanelAsRoster() - String player_choice + Dimension returnPanelSize() - String team_choice + void main(String[] args) - String pos_choice Graph Data Panel- JTable jtable - JScrollPane scroll Creates a linear graph with the user's score - JPanel wrapper showing their score after each round. - List<Player> results - final Color[] col + MarketPanel(JScrollPane scroll, int val) + GraphData() + JComboBox<String> addComboBox(+ void paintComponent(Graphics g) List<String> choices, String flag) - void refreshJTable() NameChangePanel- JTable getJTable(String player_searchd, String team_searchd, String pos_searchd) Allows the user to change his/her name, - Object[][] getTableData() also shows some key information: current player, money, and round. $Field \ Viewer Panel$ - final JTextField name This class shows the current user his roster on a football field. + NameChange() + void changeName(String newName) -final JPanel[] players + void addChangeListener() -final Roster roster + FieldViewerPanel() RosterPanel+ JLabel createLabels(String name) Shows the players his current roster and the status + void paintComponent(Graphics g) of his/her players, e.g. injuries and yellow/red cards. League Panel- Roster roster - JLabel num_players The panel that shows the users the current league - final Integer IS_ON_FIELD_COLUMN standings, and which games are upcoming.

Multiplicities are all 1:1.

+ RosterPanel()

+ LeaguePanel()

2.2.1 Component classes

We are also using the following two component classes

ButtonColumn

Changes a single column of a JTable to buttons.

- JTable table
- Action action
- int mnemonic
- Border originalBorder
- Border focusBorder
- JButton renderButton
- JButton editButton
- Object editorValue
- boolean isButtonColumnEditor
- + ButtonColumn(JTable table, Action action, int column)
- + Border getFocusBorder()
- + void setFocusBorder(Border focusBorder)
- + int getMnemonic
- + void setMnemonic(int mnemonic)
- + Component getTableCellEditorComponent(JTable t, Object val, boolean selected, int row, int col)
- + Object getCellEditorValue()
- + Component getTableCellRendererComponent(JTable t, Object val, boolean selected, boolean focus, int row, int col)
- + void actionPerformed(ActionEvent e)
- $+\ {\tt void\ mousePressed(MouseEvent\ e)}$
- + void mouseReleased(MouseEvent e)

CustomButton

Our own custom button that implements JButton. Looks much nicer than the default Swing button.

- Color hoverBackgroundColor
- Color pressedBackgroundColor
- Color fontColor
- Color hoverFontColor
- + CustomButton()
- + CustomButton(String text)
- + Color getHoverBackgroundColor()
- + Color getFontColor()
- + void setFontColor(Color color)
- + void setContentAreaFilled(boolean b)
- + void setHoverBGColor(Color color)
- + Color getHoverFontColor()
- + void setHoverFontColor(Color color)
- + Color getPressedBGColor()
- + void setPressedBGColor(Color color)
- + void paintComponent(Graphics g)

2.3 Connections between front-end and back-end classes

There are a number of connections between back-end and front-end classes that the previous diagrams could not show. These connections will be shown here.

2.3.1 Main

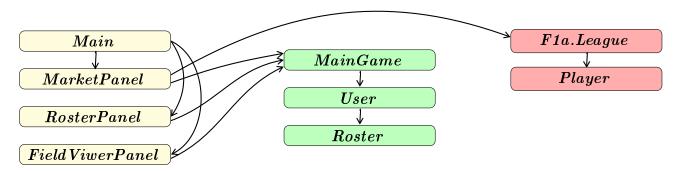
When the game starts, MainGame creates a instance of MainGame.



2.3.2 MarketPanel, RosterPanel, and FieldViwerPanel

MarketPanel needs to get all teams from a class group F1a will create. That class will return to the MarketPanel all the Teams (and therefore all Player objects). MarketPanel will also need to get the current user roster to determine which player the user owns and which he does not own. It can do that through the MainGame class just as the diagram below shows. This connection will be shown more clearly in one of our sequence diagrams.

Similar to the MarketPanel, the RosterPanel and FieldViwerPanel also need to get the current roster.



2.3.3 NameChangePanel

The NameChangePanel is associated with the User class in case of the user changes his/her name. It also gets some information about the user's current money and the current game's current round.



3 Sequence diagrams

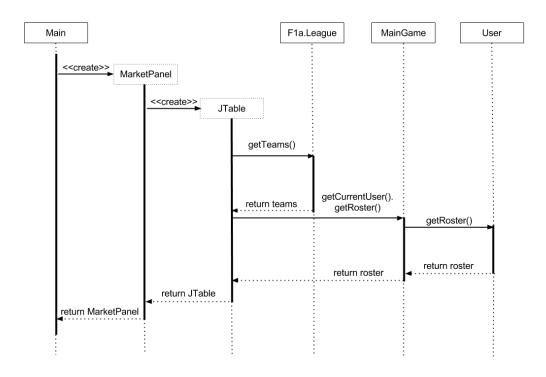


Figure 1: Sequence diagram 1.

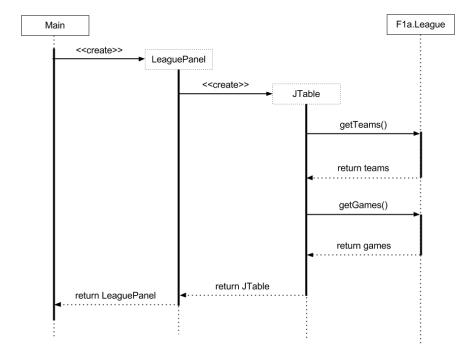


Figure 2: Sequence diagram 2.