Objects Interaction

Christian A. Rodríguez
Edited by Juan Mendivelso

Object Oriented Programming

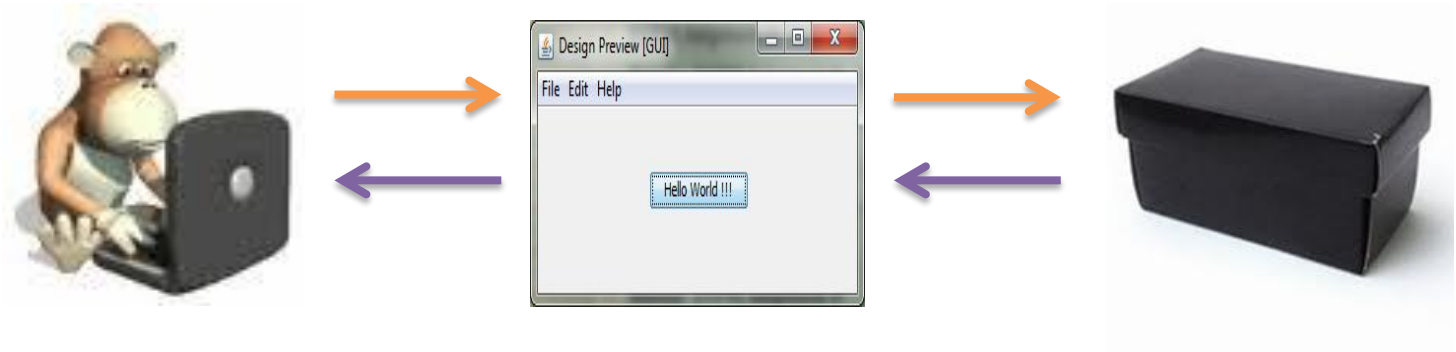
How software works?

User interface triggers

Objects collaboration

Software layers

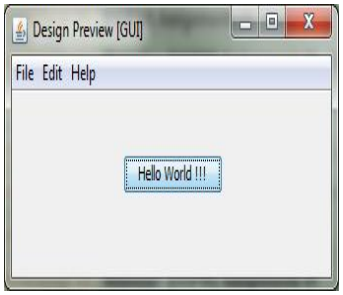
User perspective and the “black box”



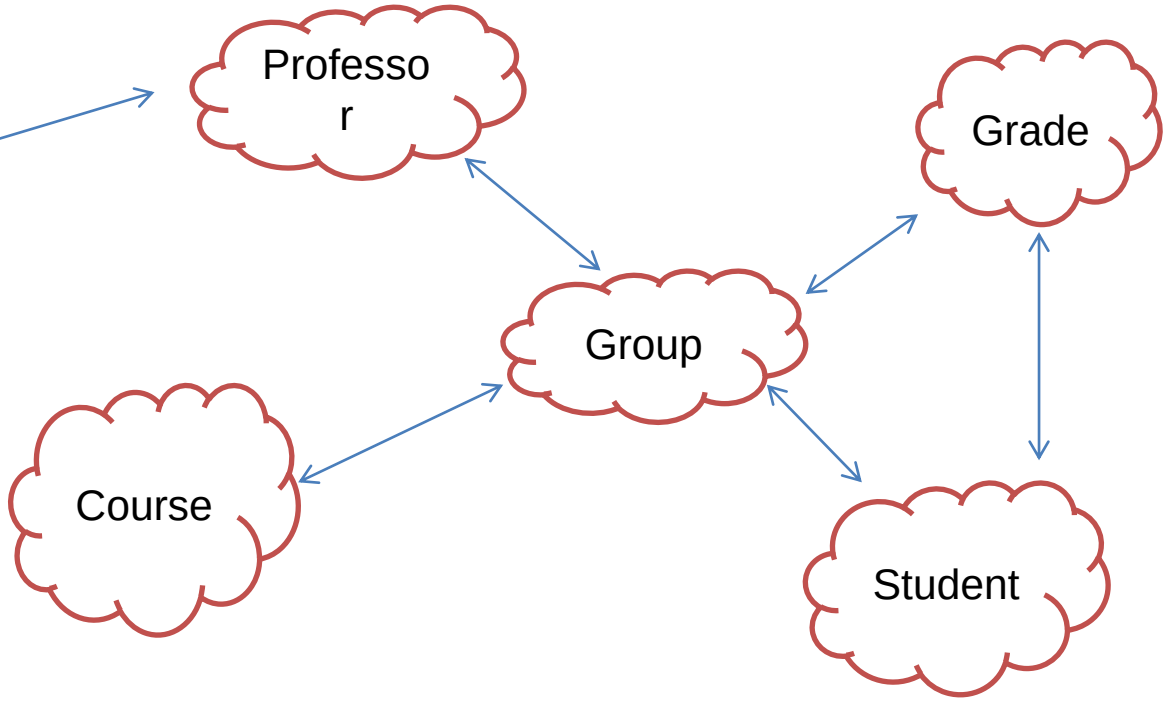
Users do not know how software internally works

UI used to be a bridge between user and the “black box”

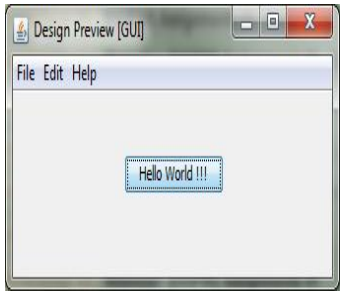
Black box contains networks of related objects



UI Click



Objects collaborate and delegate tasks



UI button
Click action

A professor wants to know his students' grades

Professor

Get this professor's group grades

Group

Get grades for each student

Grade

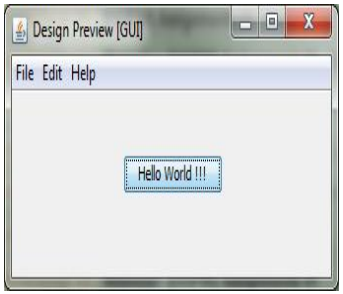
Get this student's grades

Get this group's students list

Student

Course

Delegation



UI button Click

A professor wants to know his students' grades

Professor

Get this professor groups

Group

Get grades for each student

Grade

Get this student's grades

Get this group's students list

Student

Professor object **delegates** part of his task to an Group object

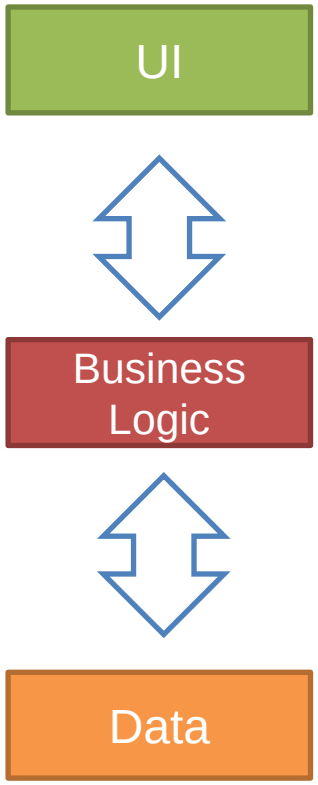
Delegation

Delegation is **invisible** for the object user

Delegation among software objects is exactly the same as delegation between people in the real world

How to organize all those objects?

- Software is generally structured in layers:
  - UI Layer
  - Business Logic Layer
  - Data Layer
- Each layer is responsible for a set o related tasks



Layer responsibilities

- Contains encapsulated classes definitions that represent the core data of the model.
- Contains data integrity validations.
- Do not contain business logic code.

Data layer

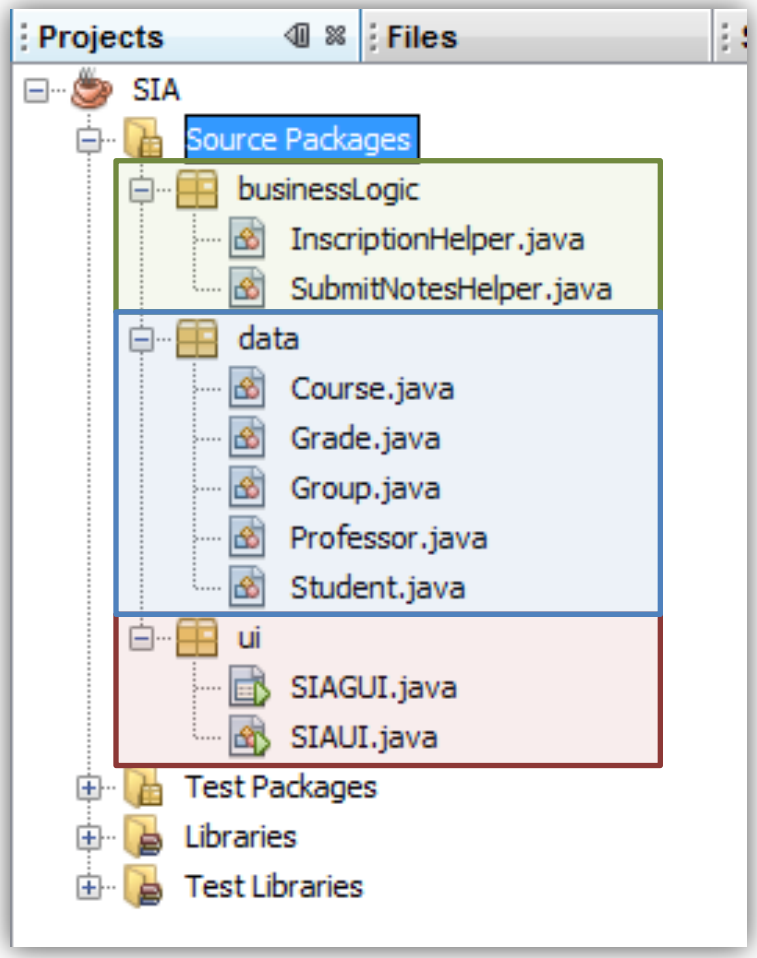
- Contains the "functional code".
- It comprised by the classes that are in charge of the main operations supported by the program.

Business Logic layer

- It is constituted by the classes in charge of interacting with the user.
- Contains the UI code, forms, applets, web pages, etc.
- Do not contain business logic code.

UI Layer

How to organize all those objects?



Classes can be organized in packages

Each layer can be represented by a package

Classes in different packages must be imported in other classes

Review: Tic Tac Toe

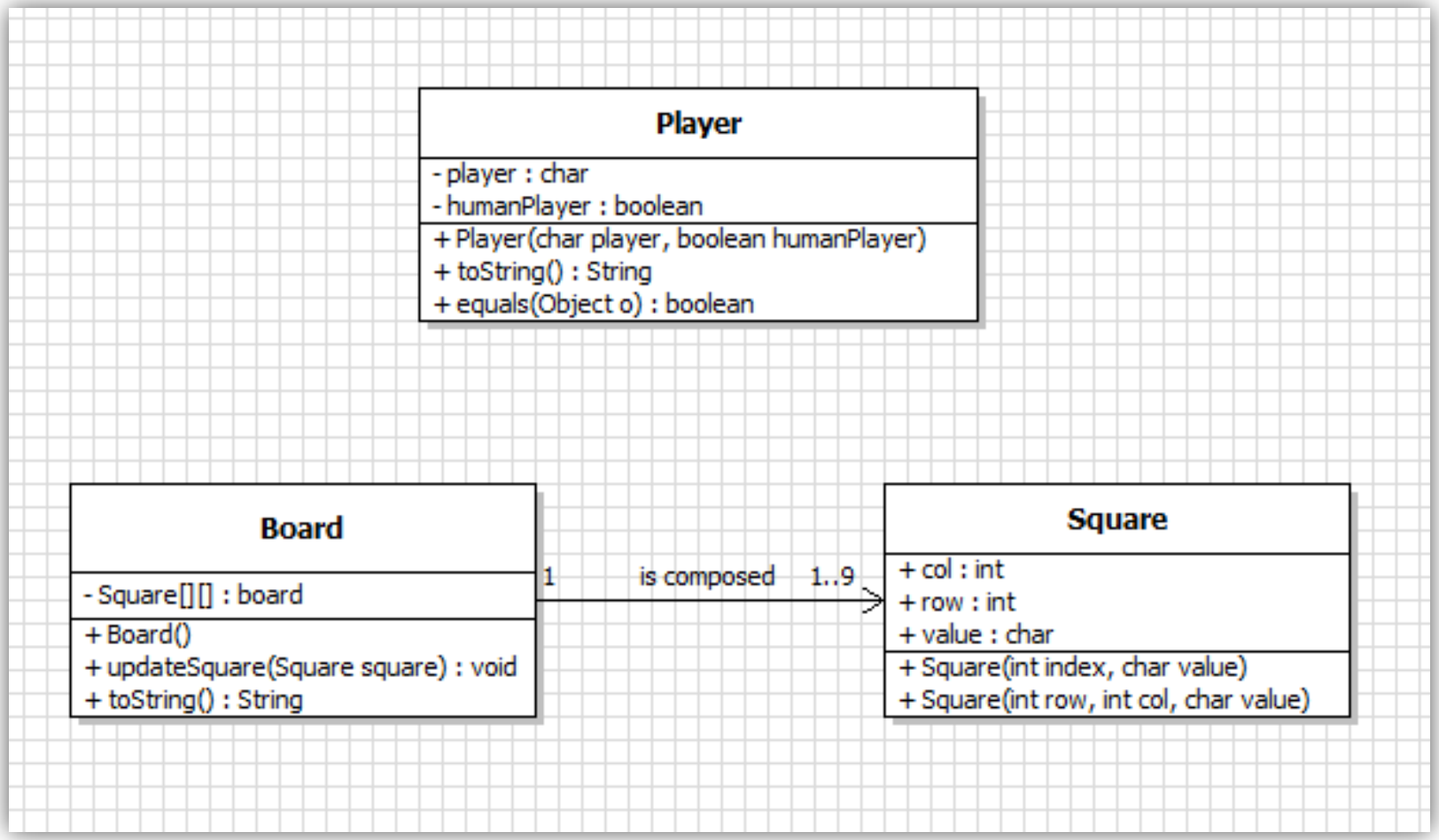
Layers and packages

Constructors and access modificators

Relation between objects

Data Layer

Data layer


Player
- player : char
- humanPlayer : boolean
+ Player(char player, boolean humanPlayer)
+ toString() : String
+ equals(Object o) : boolean
Board
- Square[][] : board
+ Board()
+ updateSquare(Square square) : void
+ toString() : String
1
is composed
1..9
Square
+ col : int
+ row : int
+ value : char
+ Square(int index, char value)
+ Square(int row, int col, char value)

```java
public class Square {

    private int row;
    private int col;
    private char value;

    public Square(int row, int col, char value) {
        this.row = row;
        this.col = col;
        this.value = value;
    }

    public Square(int index, char value) {
        this.row = (index - 1) / 3;
        this.col = (index - 1) % 3;
        this.value = value;
    }

    public int getCol() {...}

    public void setCol(int col) {...}

    public int getRow() {...}

    public void setRow(int row) {...}

    public char getValue() {...}

    public void setValue(char value) {...}

    @Override
    public String toString() {...}
}
```

Overloading constructors

Overriding toString method

```java
@Override
public String toString() {
    return String.valueOf(this.getValue());
}
```

String.valueOf( ) is used to cast variables to String

valueOf(Object o)	String
valueOf(boolean bln)	String
valueOf(char c)	String
valueOf(char[] chars)	String
valueOf(double d)	String
valueOf(float f)	String
valueOf(int i)	String
valueOf(long l)	String
valueOf(char[] chars, int i, int i1)	String

Board class

Overriding default constructor

```
public class Board {

    private Square[][] board;

    public Board() {

        char value = '0';
        board = new Square[3][3];

        for (int row = 0; row < board.length; row++) {
            for (int col = 0; col < board.length; col++) {
                Square square = new Square(row, col, (char) (++value));
                board[row][col] = square;
            }
        }
    }

    public Square[][] getBoard() {...}

    public void updateSquare(Square square) {...}

    @Override
    public String toString() {...}
}
```

Printing user defined objects method

```java
public static void printBoard(Board board) {
    System.out.println(board);
}
```

```
tictactoe.data.Board@530daa
```

toString method

toString method allow us **override the default way how the object is printed**

Important, do not forget it

```
@Override
public String toString() {
    String printBoard = "\n";

    for (int row = 0; row < board.length; row++) {
        printBoard = printBoard.concat("\t|");
        for (int col = 0; col < board.length; col++) {
            printBoard = printBoard.concat(
                    String.valueOf(board[row][col]).concat("|"));
        }
        printBoard = printBoard.concat("\n");
    }
    return printBoard;
}
```

The return type is String

Overriding toString method

```java
public static void printBoard(Board board) {
    System.out.println(board);
}
```

```
|1|2|3|
|4|5|6|
|7|8|9|
```

Player class

```java
public class Player {

    private char player;
    private boolean humanPlayer;

    public Player(char player, boolean humanPlayer) {...}

    public char getPlayer() {...}

    public void setPlayer(char player) {...}

    public boolean isHumanPlayer() {...}

    @Override
    public String toString() {...}

    @Override
    public boolean equals(Object obj) {...}
}
```

Overriding toString method

```java
@Override
public String toString() {

    String printPlayer = "I am the player "
            + String.valueOf(this.getPlayer()) + " and I am ";

    if (this.isHumanPlayer()) {
        printPlayer = printPlayer.concat("HUMAN");
    } else {
        printPlayer = printPlayer.concat("A ROBOT");
    }
    return printPlayer;
}
```

How do you know if two user-defined objects are equal?

Overriding equals method

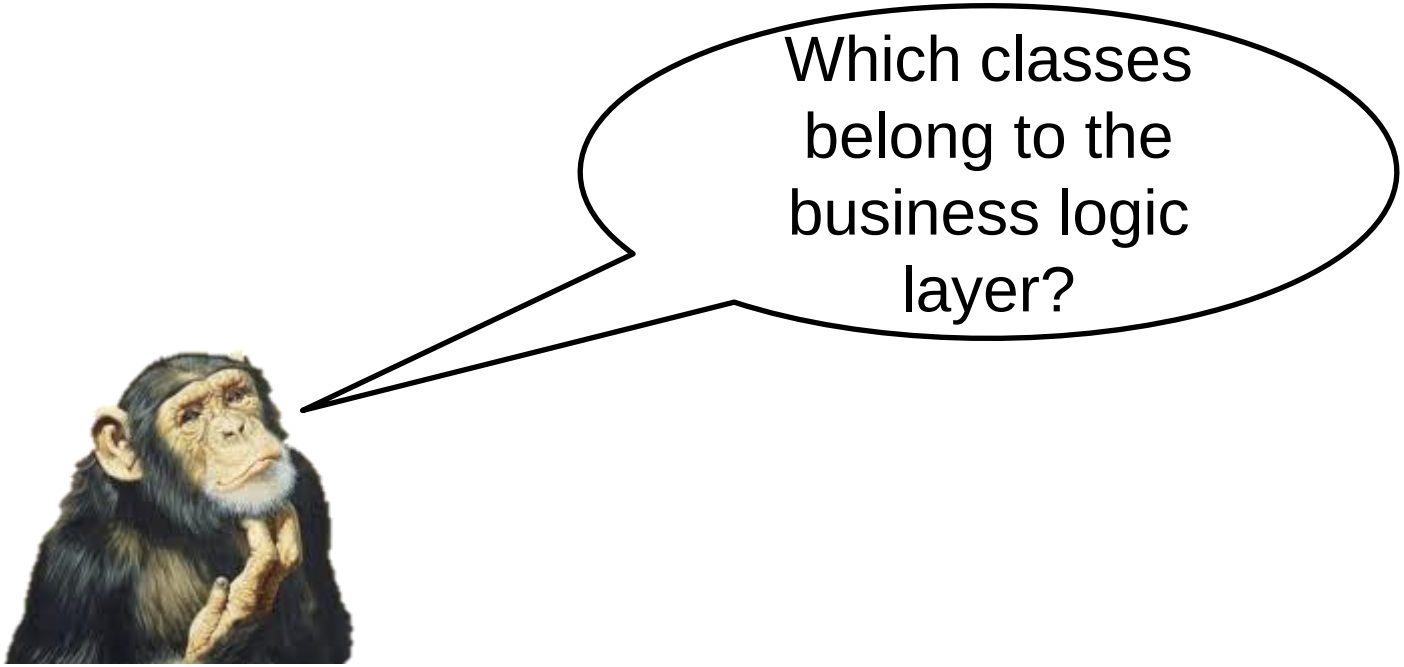
equals method allow us to **override the default way how two user defined objects are equals or not**

Important, do not forget it

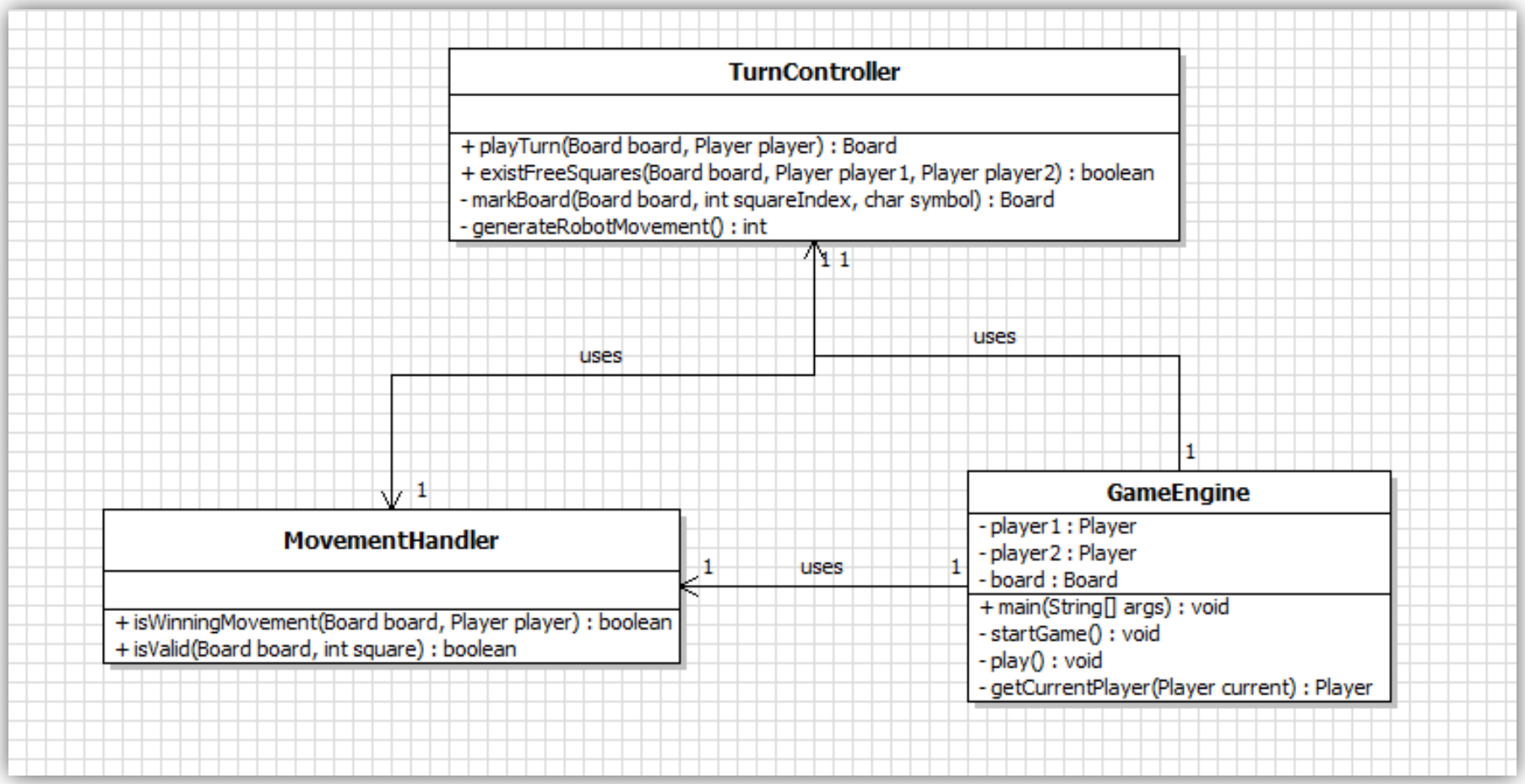
If the two objects have the same player (Symbol) are equals

```
@Override
public boolean equals(Object obj) {
    if (obj == null) {
        return false;
    }
    if (getClass() != obj.getClass()) {
        return false;
    }
    final Player other = (Player) obj;
    if (this.player != other.player) {
        return false;
    }
    return true;
}
```

Business logic Layer



Business logic layer



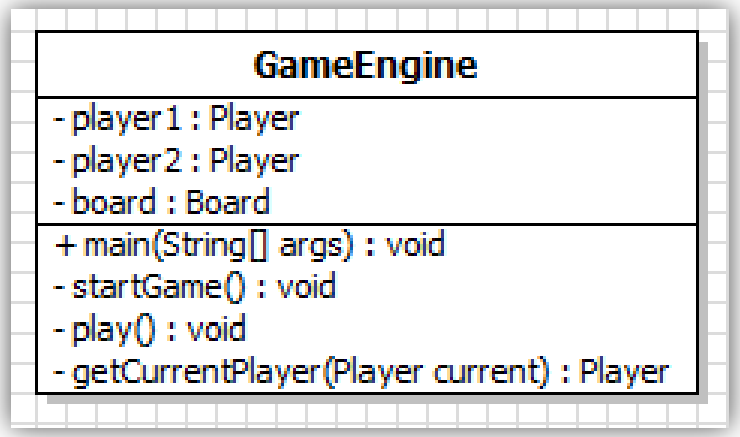
GameEngine Class

This is the game starting point

```
import tictactoe.data.Board;
import tictactoe.data.Player;
import tictactoe.ui.UI;

public class GameEngine {

    private static Player player1;
    private static Player player2;
    private static Board board;

    public static void main(String[] args) {...}

    private static void startGame() {...}

    private static void play() {...}

    private static Player getCurrentPlayer(Player current) {...}
}
```

Methods can be private

GameEngine Class

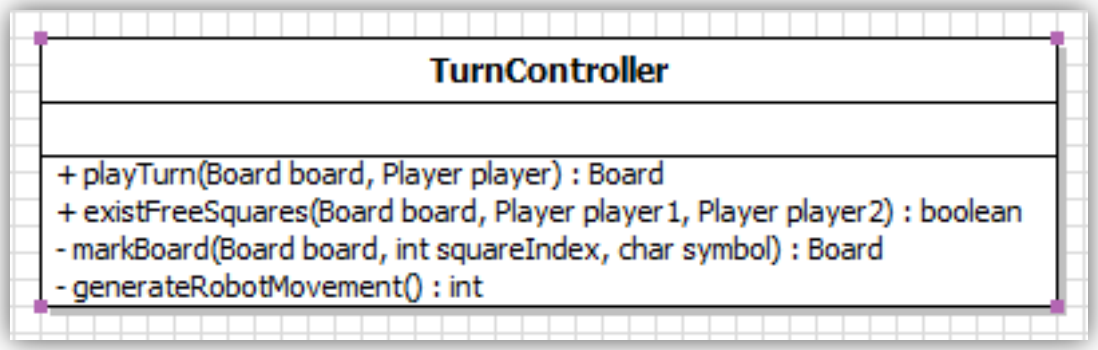


Class	Method signature	Function
GameEngine	startGame ( )	Initialize objects and variables Call the method play
	play ( )	Iterate until win or finish condition is reached
	getCurrentPlayer (Player)	Change the current player at the end of each turn

TurnController Class

```java
import java.util.Random;
import tictactoe.data.Board;
import tictactoe.data.Player;
import tictactoe.data.Square;
import tictactoe.ui.UI;

public class TurnController {

    public static Board playTurn(Board board, Player player) {...}

    private static Board markBoard(Board board, int squareIndex, char symbol) {...}

    public static boolean existFreeSquares(Board board, Player player1, Player player2) {...}

    private static int generateRobotMovement() {...}
}
```

TurnController Class

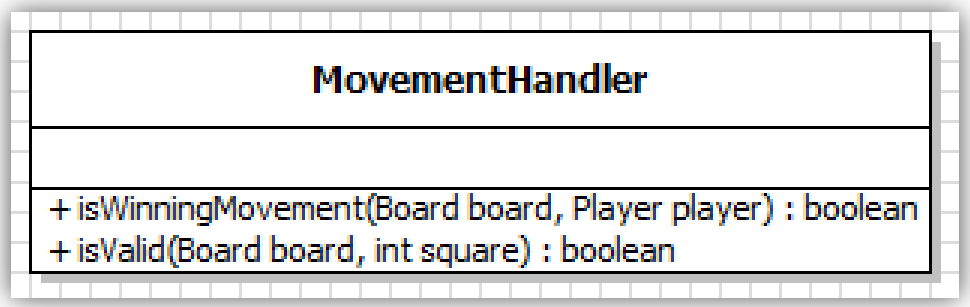


Class	Method signature	Function
TurnController	playTurn (Board , Player)	Handle movement turn Call movement validator Call board modifier
	markBoard (Board , int , char)	Modify board after a valid play
	existFreeSquares (Board , Player , Player)	Check if there are available squares to play
	generateRobotMovement ( )	Generate random robot movement

MovementHandler Class

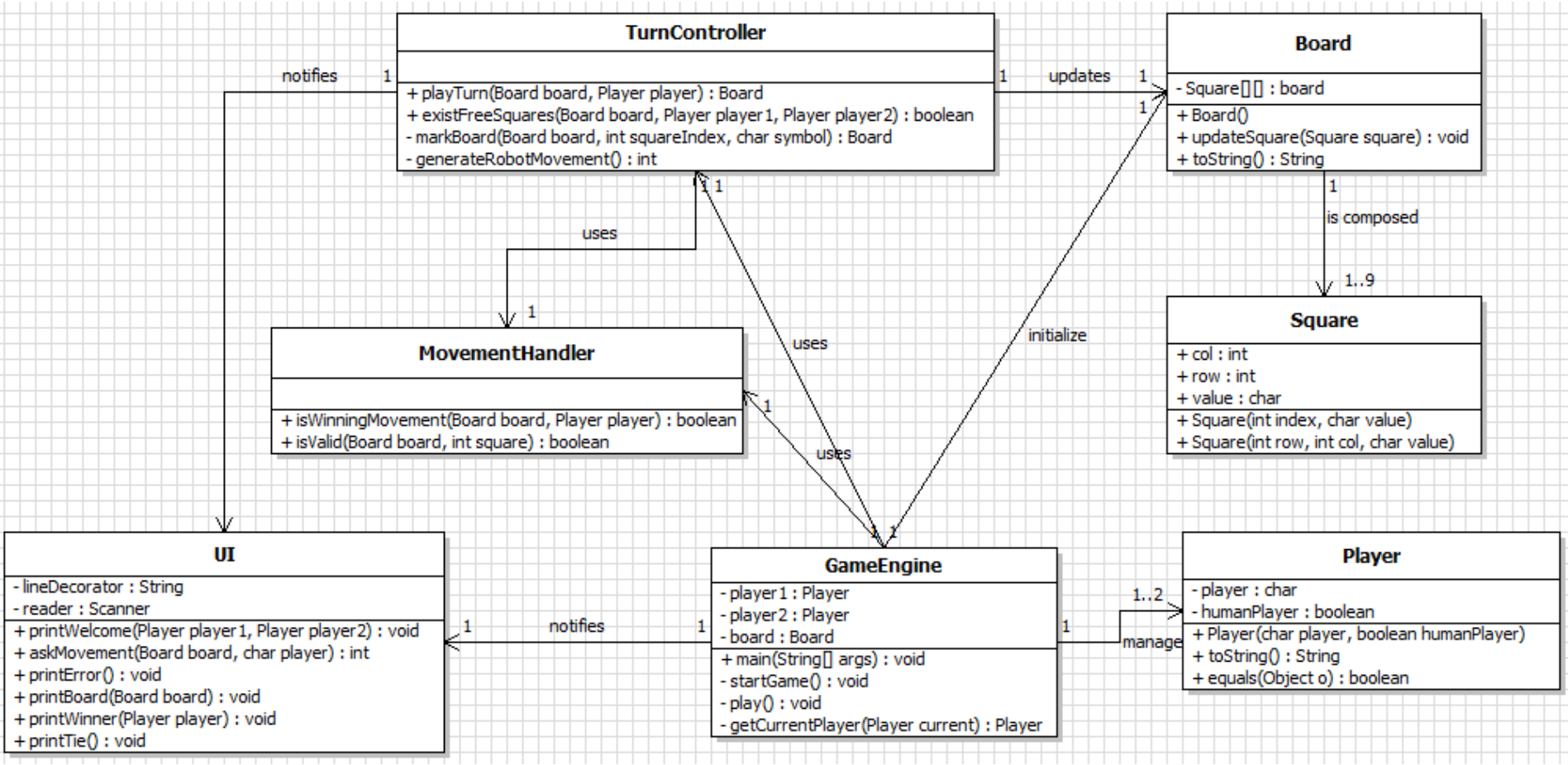
```
import tictactoe.data.Board;
import tictactoe.data.Player;
import tictactoe.data.Square;

public class MovementHandler {

    public static boolean isValid(Board board, int square) {...}

    public static boolean isWinningMovement(Board board, Player player) {...}
}
```

MovementHandler Class

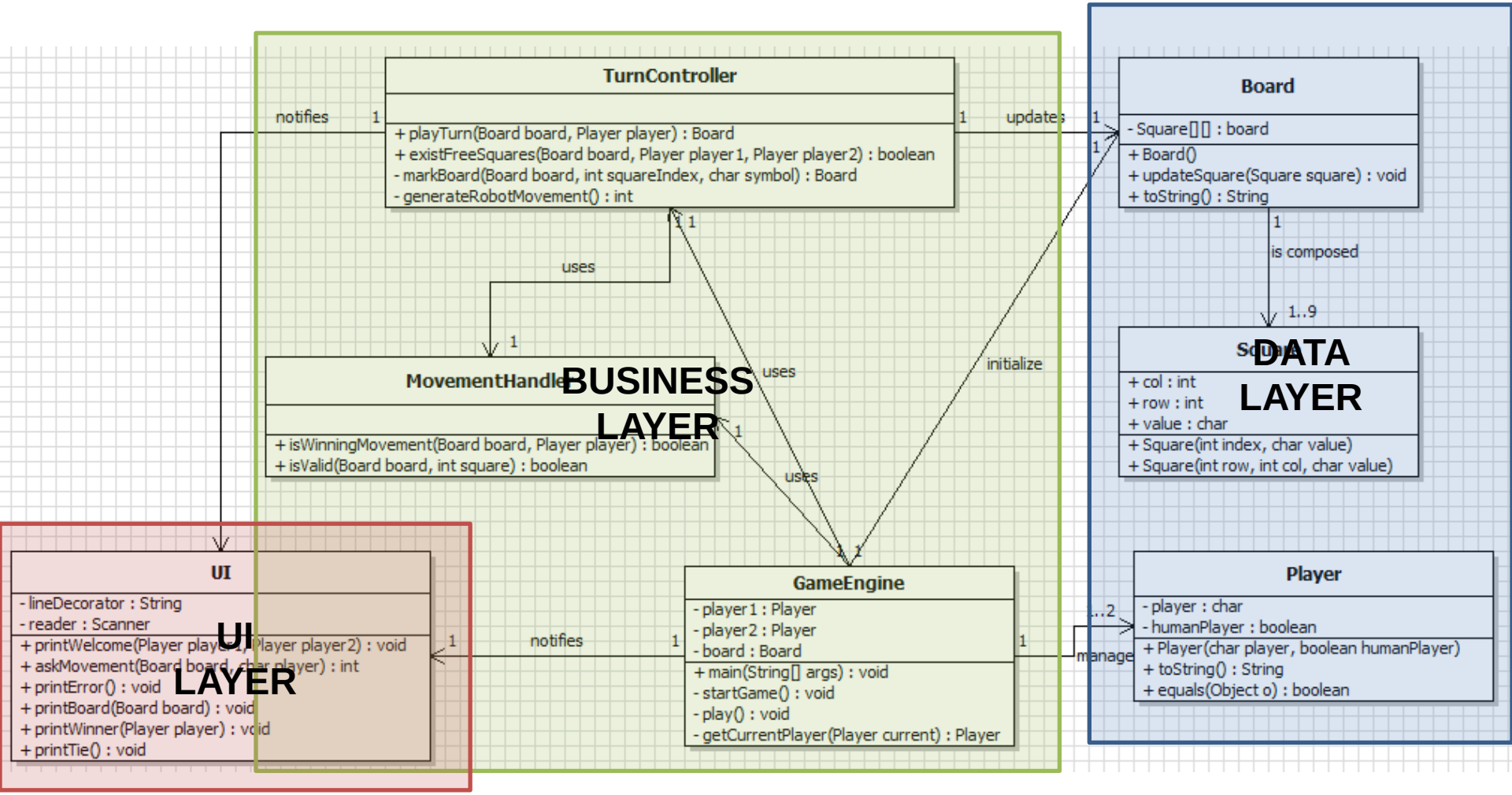


Class	Method signature	Function
MovementHandler	isValid (Board , int)	Check if a square selection is available to be marked
	isWinningMovement (Board , Player)	Check if the last movement causes the player's victory

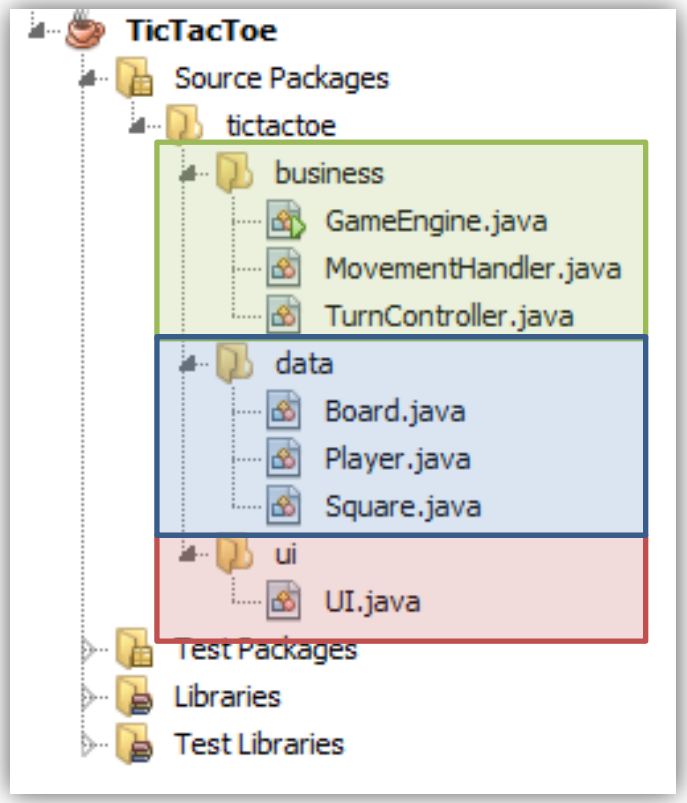
UML Class diagram



UML Class diagram



Each layers is represented as a package



- Three layers
  - Data
  - Business logic
  - UI

Tic Tac Toe

Check the code here

References

- [Barker] J. Barker, *Beginning Java Objects: From Concepts To Code*, Second Edition, Apress, 2005.