Object Calisthenics

by Jeff Bay

Photo by Eduardo Madrid on Unsplash

Rules to get a better code design, if applied correctly ;)

RULE 1: Only one level of indentation

- → helps method focuses on doing only one thing
- → reduce size of the method
- → enabling easier reuse

```
public Player getWinnerBad() {
  Player previousPlayer = getPreviousPlayer();
  if (this.board[winPositions[0][0]].equals(previousPlayer.getMark().toString())) {
     if (this.board[winPositions[0][1]].equals(previousPlayer.getMark().toString())) {
        if (this.board[winPositions[0][2]].equals(previousPlayer.getMark().toString())) {
           return previousPlayer;
                                        public Player getWinnerGood() {
                                           Player previousPlayer = getPreviousPlayer();
  return null:
                                            if ((this.board[winPositions[0][0]].equals(previousPlayer.getMark().toString())
                                                  && this.board[winPositions[0][1]].equals(previousPlayer.getMark().toString())
                                                  && this.board[winPositions[0][2]].equals(previousPlayer.getMark().toString()))) {
                                               return previousPlayer;
                                            return null;
```

RULE 2: Don't use the ELSE keyword

- → focusing main execution lane
- \rightarrow avoiding complex conditional cases

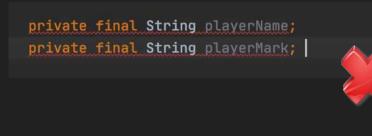
```
private void switchToNextPlayerBad() {
    if (this.currentPlayer.getMark() == Symbol.X) {
        this.currentPlayer = this.secondPlayer;
    } else {
        this.currentPlayer = this.firstPlayer;
    }
}
private void this.currentPlayer = this.currentPlayer;
}
```

private void switchToNextPlayerGood() {
 this.currentPlayer = this.currentPlayer.getMark() == Symbol.X
 ? this.secondPlayer
 : this.firstPlayer;
}



RULE 3: Wrap all primitives and strings in classes

- → expressing intents
- → proper namings e.g. domain specific with DDD





RULE 4: First class collection

- → expressing intents
- → proper namings
- → filter and joins belong to a collection

```
public class Board {
    private String[] board = new String[9];
    private final int[][] winPositions = {
        {0, 1, 2},
        {3, 4, 5},
        {6, 7, 8}
    };
}
```

```
public class Board {
    private String[] board = new String[9];
    private final WinPositions positions;
}
public class WinPositions {
    private final int[][] kombinations = {
        {0, 1, 2},
        {3, 4, 5},
        {6, 7, 8}
    };
```

RULE 5: One dot per line

- → tell object to do something instead of asking for internal representation
- → reduce the amount of knowledge

ticTacToe.getPlayer().setMark();



ticTacToe.play();



RULE 6: Don't abbreviate

 \rightarrow abbreviations can be confusing, so it's better to be clear

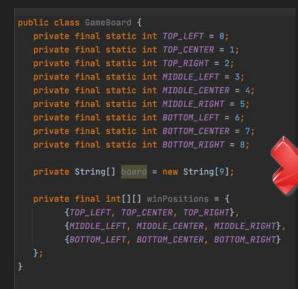
};

```
private final int[][] winPositions = {
        {TL, TC, TR},
        {ML, MC, MR},
        {BL, BC, BR}
};
```

private final int[][] winPositions = {
 {TOP_LEFT, TOP_CENTER, TOP_RIGHT},
 {MIDDLE_LEFT, MIDDLE_CENTER, MIDDLE_RIGHT},
 {BOTTOM_LEFT, BOTTOM_CENTER, BOTTOM_RIGHT}

RULE 7: Keep all entities small

- → small classes tend to focus on doing just one thing
- → easier to reuse and easier to understand
- \rightarrow clear intention



ipublic class WinPositions {
 private final static int TOP_LEFT = 0;
 private final static int TOP_CENTER = 1;
 private final static int TOP_RIGHT = 2;
 private final static int MIDDLE_LEFT = 3;
 private final static int MIDDLE_CENTER = 4;
 private final static int MIDDLE_RIGHT = 5;
 private final static int BOTTOM_LEFT = 6;
 private final static int BOTTOM_CENTER = 7;
 private final static int BOTTOM_RIGHT = 8;

private final int[][] positions = {
 {TOP_LEFT, TOP_CENTER, TOP_RIGHT},
 {MIDDLE_LEFT, MIDDLE_CENTER, MIDDLE_RIGHT},
 {BOTTOM_LEFT, BOTTOM_CENTER, BOTTOM_RIGHT}
};

}

public class Board {

private String[] board = new String[9];
private final WinPositions positions;

RULE 8: No class with more than 2 instance variables

- → the more instance variable the lower the cohesion within the class
- \rightarrow less coupling and modularization

```
public class Board {
    private final Player firstPlayer;
    private final Player secondPlayer;
    private Player currentPlayer;
    private final WinPositions positions;
```

private String[] board = new String[9];

}



}

public class Board {
 private String[] board = new String[9];
 private final WinPositions positions;



RULE 9: No getter/ setter/ properties

- → instead of asking an object for its data, tell object what it should do
- → distinguish between data structures and objects, they have different responsibilities

```
public class Player {
                                                                      public class Player {
  private final String name;
                                                                          private final String name;
  private final Symbol mark;
                                                                          private final Symbol mark;
  public Player(String name, Symbol mark){...}
                                                                          public Player(String name, Symbol mark) {
                                                                             this.mark = mark;
  public Symbol getMark() { return this.mark; }
                                                                             this.name = name;
                                                                          }
  public void setMark(Symbol mark) { this.mark = mark }
                                                                          public void putNextMark() {
  public Symbol getPlayer() { return this.mark; }
                                                                             // put mark
                                                                          }
  public void setPlayer(Symbol mark) { this.mark = mark }
```

RULE 10: All classes must have state

- → try creating classes that have clear responsibilities and require state
- → no static classes and methods

Contains some clean code principles

- → Single responsibility principle
- → DRY
- → KISS
- → ...