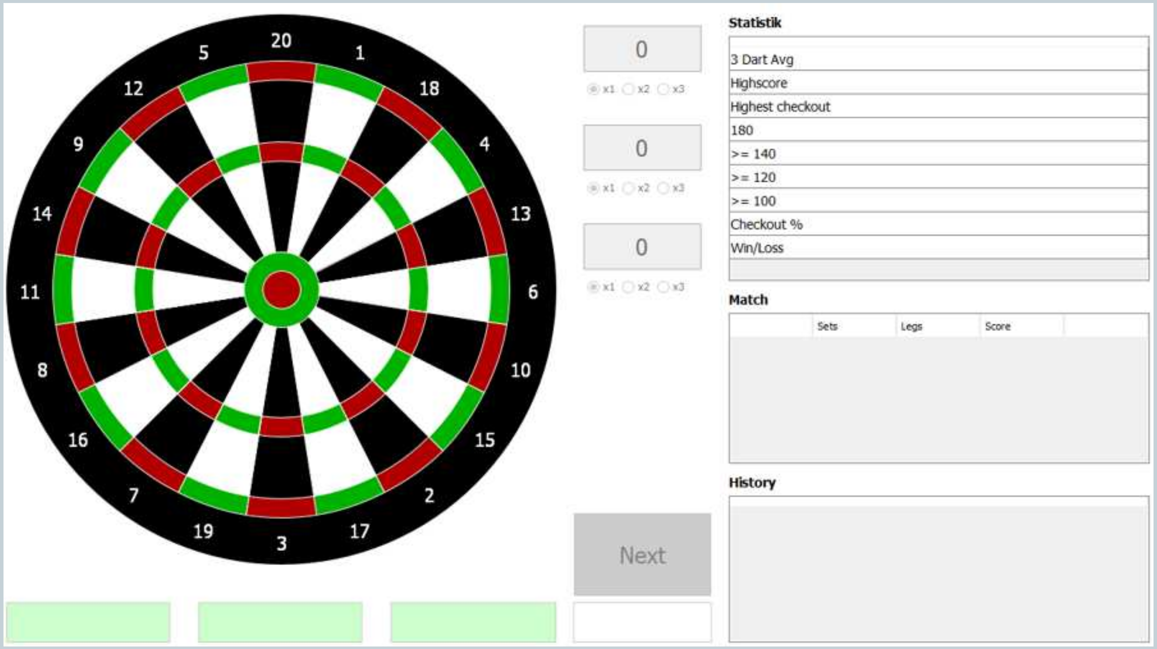
DartManager

TPP & OBJECT CALISTENICS



0

[✓ x1] x2 x3

0

[✓ x1] x2 x3

0

[✓ x1] x2 x3

Statistik

3 Dart Avg	
Highscore	
Highest checkout	
180	
>= 140	
>= 120	
>= 100	
Checkout %	
Win/Loss	

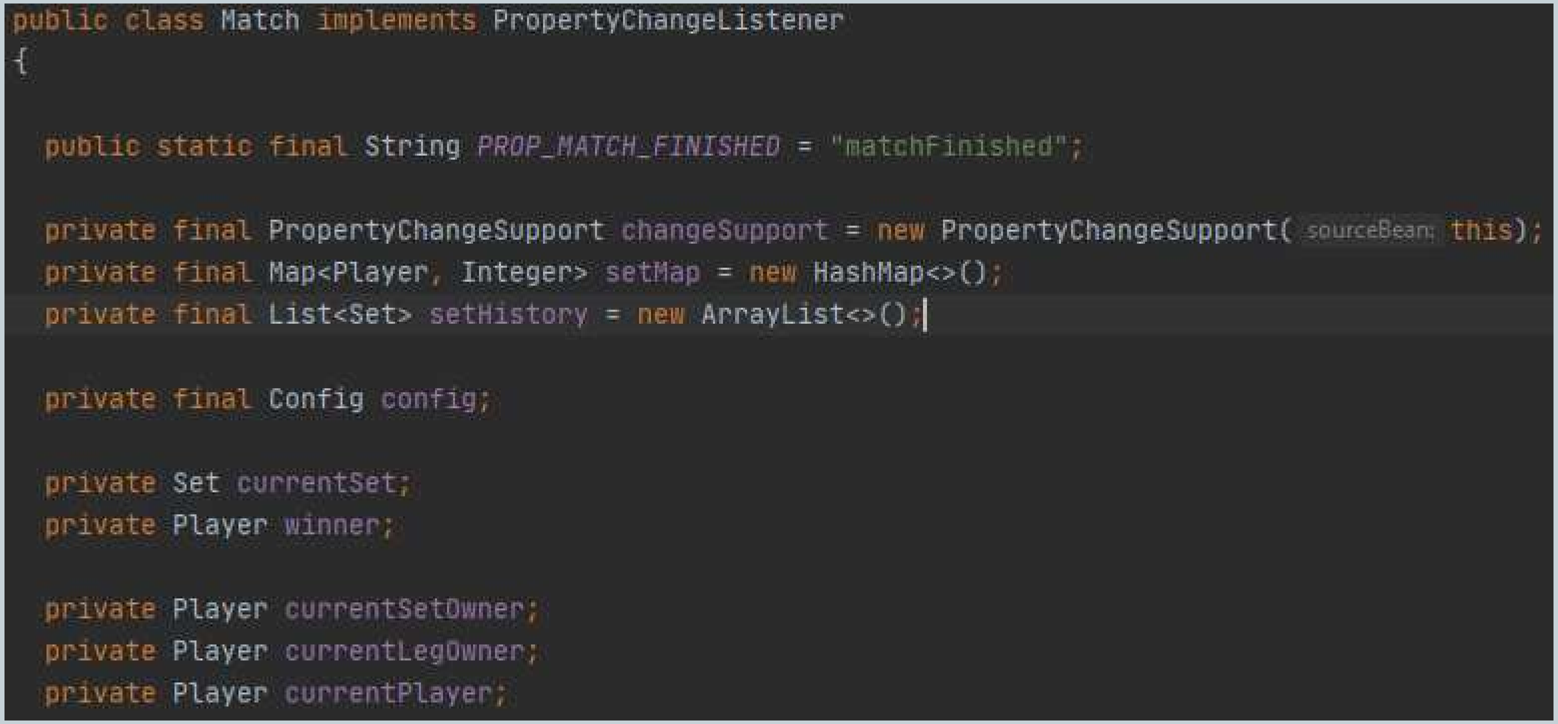
Match

	Sets	Legs	Score

History

Legacy code I

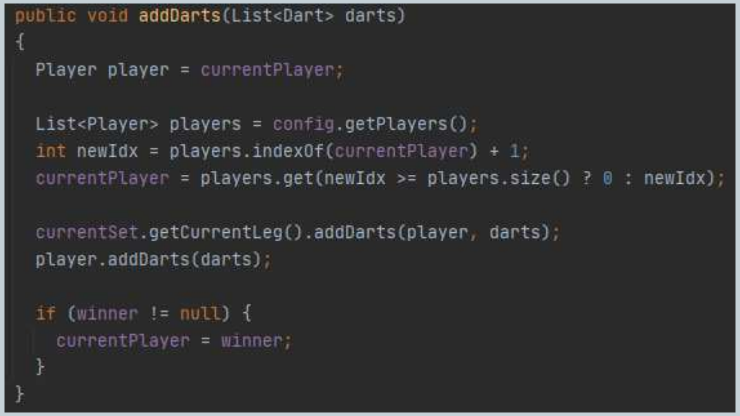
- UI driven design
- No clean abstraction of dart logic



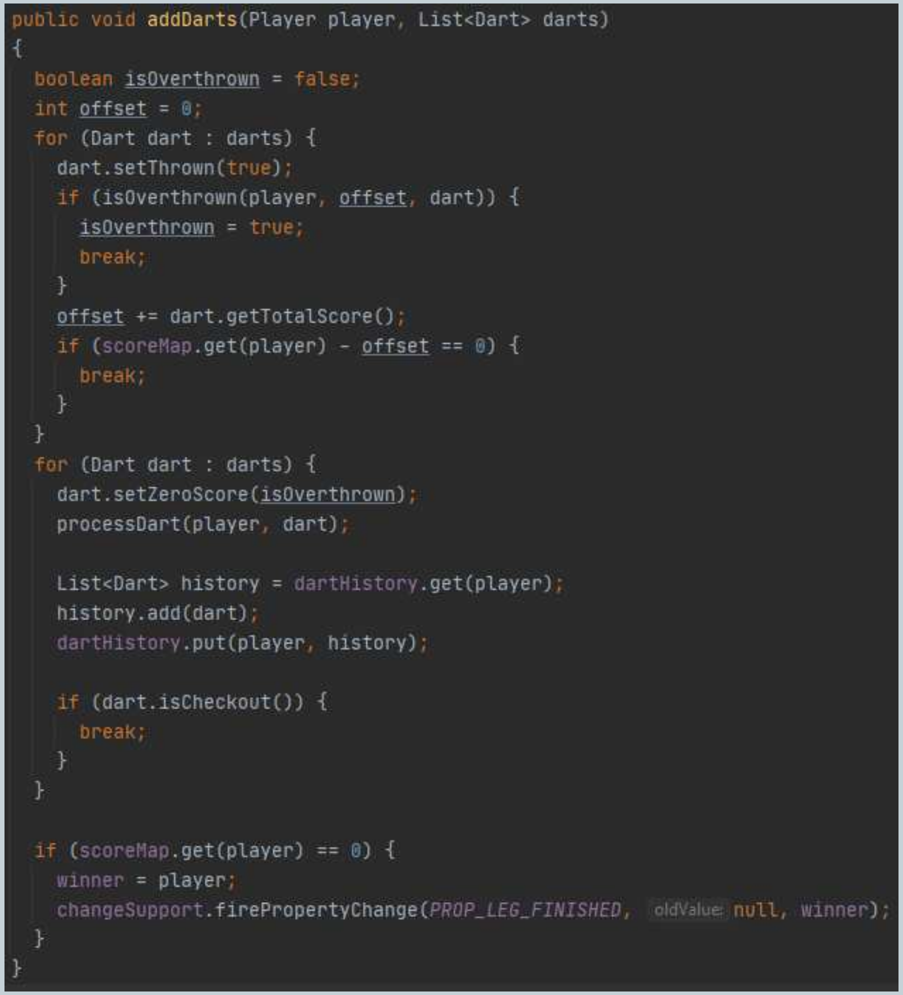
```java
public class Match implements PropertyChangeListener
{

  public static final String PROP_MATCH_FINISHED = "matchFinished";

  private final PropertyChangeSupport changeSupport = new PropertyChangeSupport( sourceBean: this);
  private final Map<Player, Integer> setMap = new HashMap<>();
  private final List<Set> setHistory = new ArrayList<>();

  private final Config config;

  private Set currentSet;
  private Player winner;

  private Player currentSetOwner;
  private Player currentLegOwner;
  private Player currentPlayer;
```

Legacy code II

- Logic all over the place
- Unreadable code



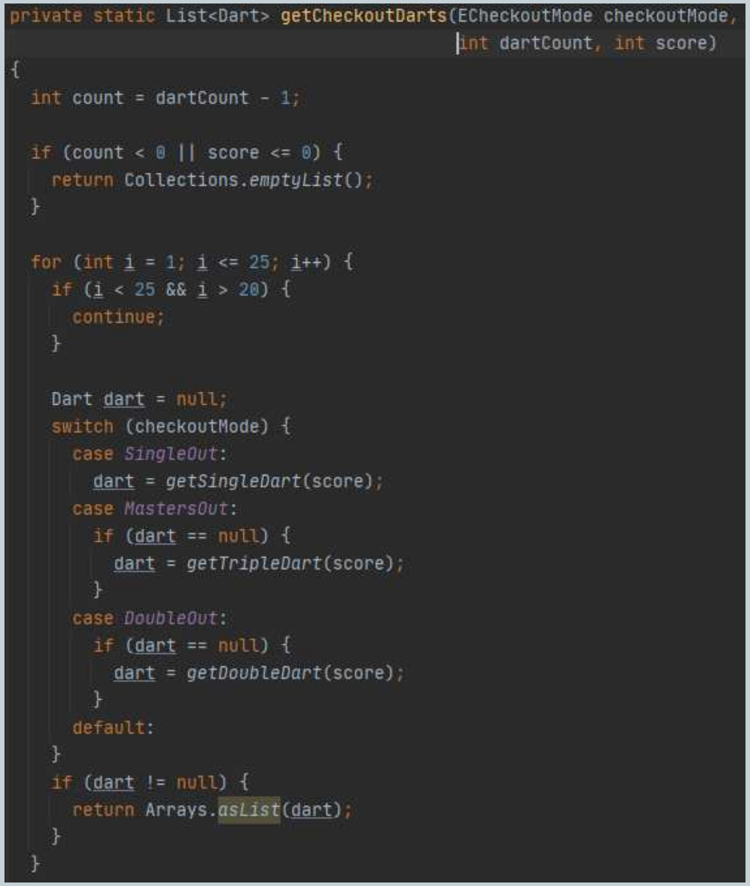
```java
public void addDarts(List<Dart> darts)
{
  Player player = currentPlayer;

  List<Player> players = config.getPlayers();
  int newIdx = players.indexOf(currentPlayer) + 1;
  currentPlayer = players.get(newIdx >= players.size() ? 0 : newIdx);

  currentSet.getCurrentLeg().addDarts(player, darts);
  player.addDarts(darts);

  if (winner != null) {
    currentPlayer = winner;
  }
}
```



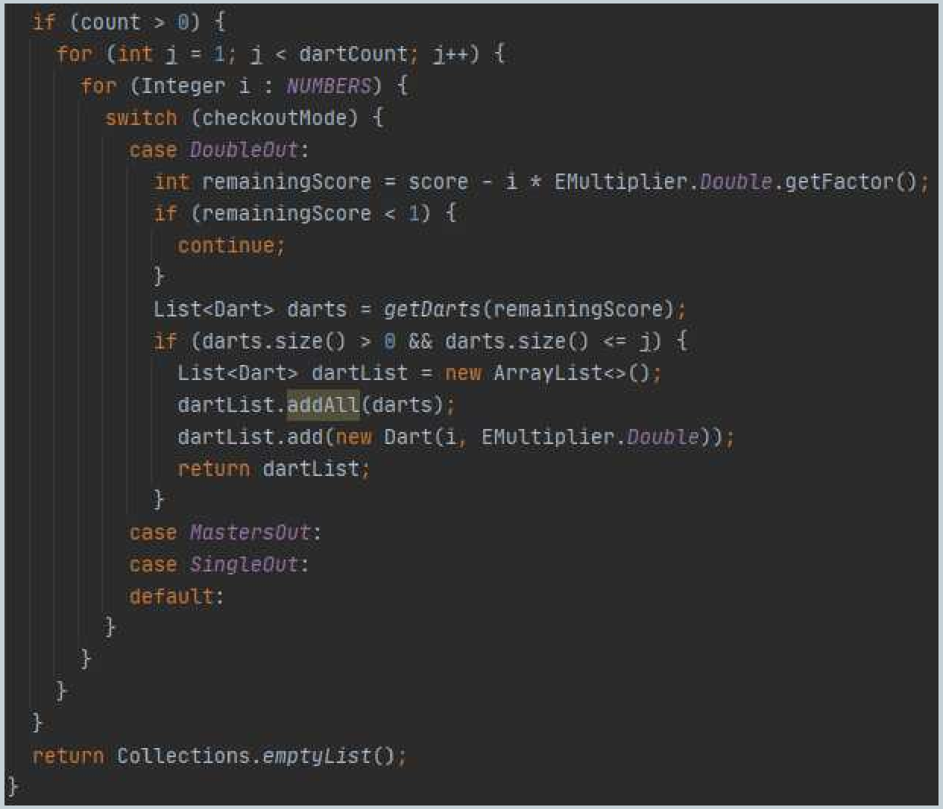
```java
public void addDarts(Player player, List<Dart> darts)
{
  boolean isOverthrown = false;
  int offset = 0;
  for (Dart dart : darts) {
    dart.setThrown(true);
    if (isOverthrown(player, offset, dart)) {
      isOverthrown = true;
      break;
    }
    offset += dart.getTotalScore();
    if (scoreMap.get(player) - offset == 0) {
      break;
    }
  }
  for (Dart dart : darts) {
    dart.setZeroScore(isOverthrown);
    processDart(player, dart);

    List<Dart> history = dartHistory.get(player);
    history.add(dart);
    dartHistory.put(player, history);

    if (dart.isCheckout()) {
      break;
    }
  }

  if (scoreMap.get(player) == 0) {
    winner = player;
    changeSupport.firePropertyChange(PROP_LEG_FINISHED,  oldValue: null, winner);
  }
}
```

Legacy code III

- Do I understand my own code?



```java
private static List<Dart> getCheckoutDarts(ECheckoutMode checkoutMode,
                                           int dartCount, int score)
{
  int count = dartCount - 1;

  if (count < 0 || score <= 0) {
    return Collections.emptyList();
  }

  for (int i = 1; i <= 25; i++) {
    if (i < 25 && i > 20) {
      continue;
    }

    Dart dart = null;
    switch (checkoutMode) {
      case SingleOut:
        dart = getSingleDart(score);
      case MastersOut:
        if (dart == null) {
          dart = getTripleDart(score);
        }
      case DoubleOut:
        if (dart == null) {
          dart = getDoubleDart(score);
        }
      default:
    }
    if (dart != null) {
      return Arrays.asList(dart);
    }
  }
```



```java
  if (count > 0) {
    for (int j = 1; j < dartCount; j++) {
      for (Integer i : NUMBERS) {
        switch (checkoutMode) {
          case DoubleOut:
            int remainingScore = score - i * EMultiplier.Double.getFactor();
            if (remainingScore < 1) {
              continue;
            }
            List<Dart> darts = getDarts(remainingScore);
            if (darts.size() > 0 && darts.size() <= j) {
              List<Dart> dartList = new ArrayList<>();
              dartList.addAll(darts);
              dartList.add(new Dart(i, EMultiplier.Double));
              return dartList;
            }
          case MastersOut:
          case SingleOut:
          default:
        }
      }
    }
  }
  return Collections.emptyList();
}
```

Refactoring?

- Problems
  - Almost no tests available
  - UI and Logic have a very high coupling level
- Conclusion
  - It takes longer to write tests and refactor than just rewrite the application
  - Created Dart Kata

Dart Kata I

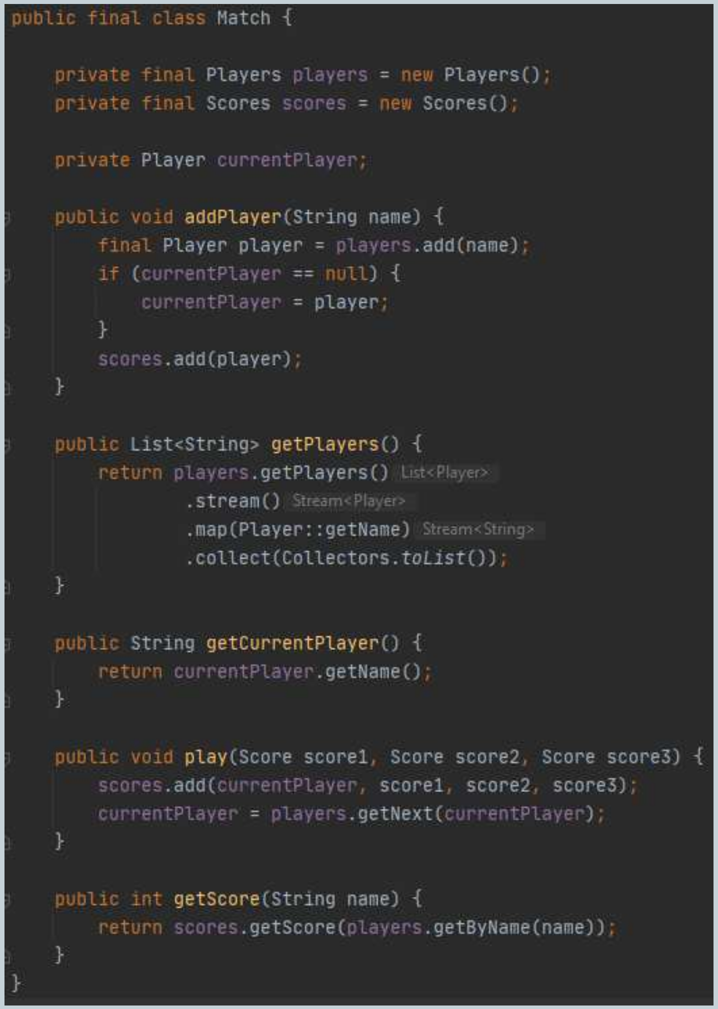
- The player size is set to 2.
- A player should have a name.
- Player 1 always starts.
- The initial score for each player is 301.
- Each player can throw 3 darts per turn.
- The supported numbers are 0-20 & 25.

Dart Kata II

- The numbers 1-20 support double and triple (e.g. hitting 3 times triple 20 results in 180 = maximum score per turn)
- The number 25 supports double (single-bull & bull).
- The first player to bring the score to 0 wins the match (this is called checkout).
- A checkout is only possible with a double (e.g. remaining score is 32 and a double 16 is hit).
- If a player overthrows (e.g. hitting 20 with 10 remaining) the player has no score for this turn.

Result I

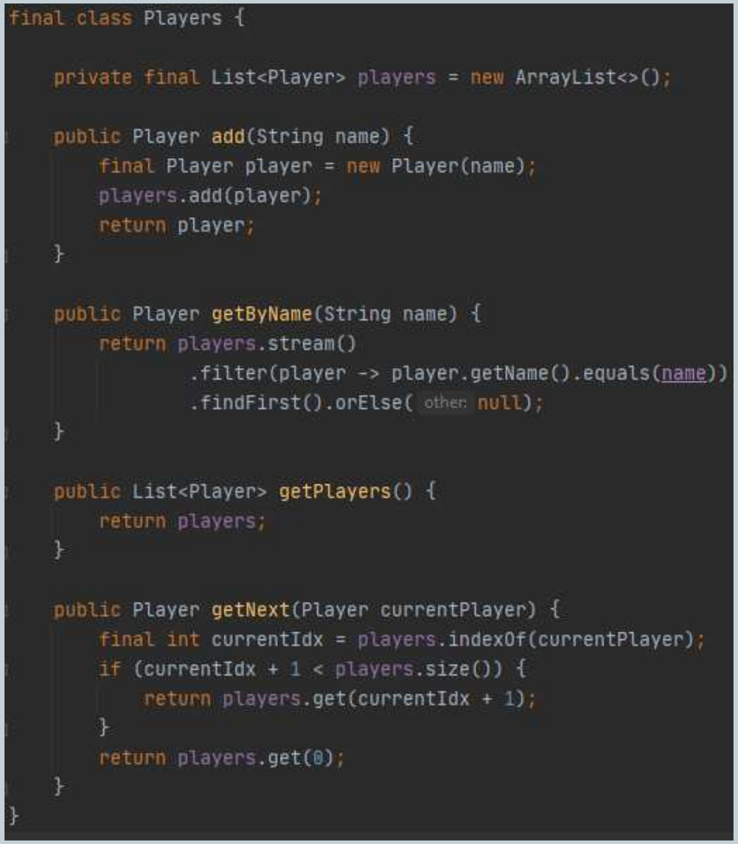
- Only one entry point
- Logic is where it belongs
- Code is readable
- Code is tested
- Code follows rules
- Code is easily extendable
- Code can be used without UI



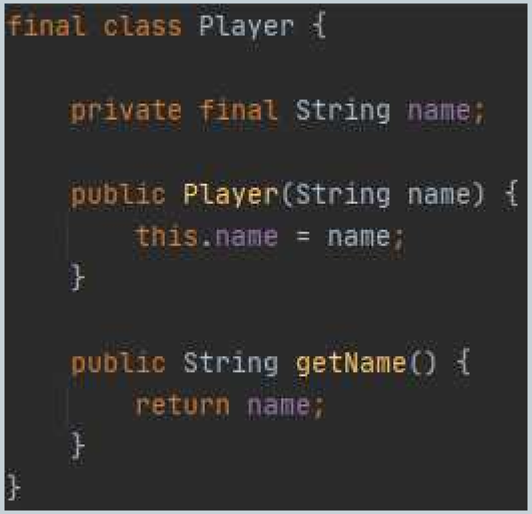
```java
public final class Match {

    private final Players players = new Players();
    private final Scores scores = new Scores();

    private Player currentPlayer;

    public void addPlayer(String name) {
        final Player player = players.add(name);
        if (currentPlayer == null) {
            currentPlayer = player;
        }
        scores.add(player);
    }

    public List<String> getPlayers() {
        return players.getPlayers() List<Player>
                .stream() Stream<Player>
                .map(Player::getName) Stream<String>
                .collect(Collectors.toList());
    }

    public String getCurrentPlayer() {
        return currentPlayer.getName();
    }

    public void play(Score score1, Score score2, Score score3) {
        scores.add(currentPlayer, score1, score2, score3);
        currentPlayer = players.getNext(currentPlayer);
    }

    public int getScore(String name) {
        return scores.getScore(players.getByName(name));
    }
}
```

Result II

- Have a class which managers the players



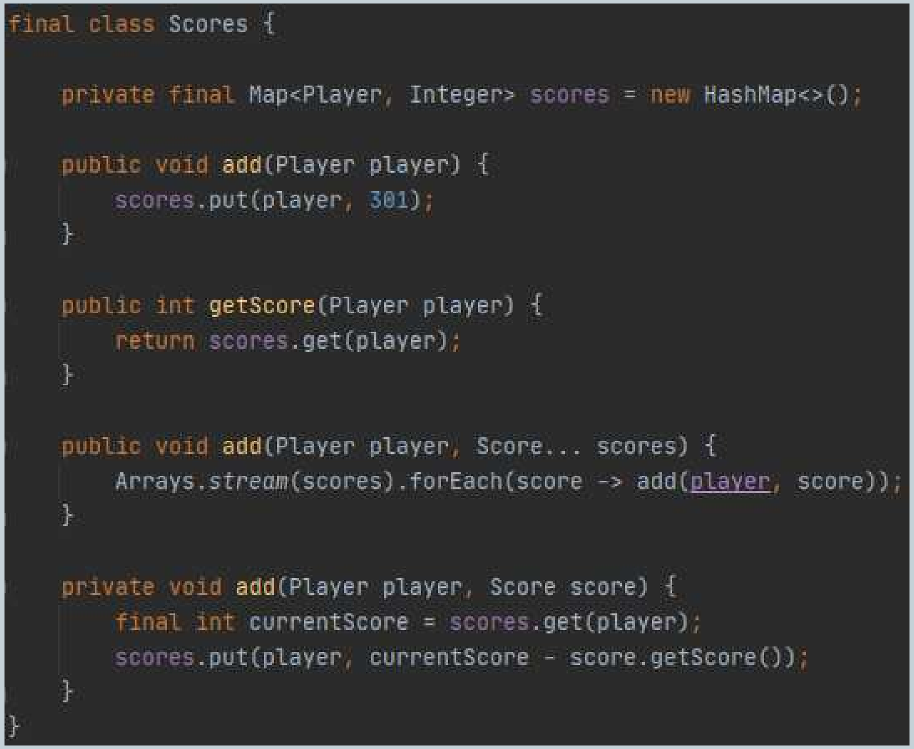
```java
final class Players {

    private final List<Player> players = new ArrayList<>();

    public Player add(String name) {
        final Player player = new Player(name);
        players.add(player);
        return player;
    }

    public Player getByName(String name) {
        return players.stream()
                .filter(player -> player.getName().equals(name))
                .findFirst().orElse( other: null);
    }

    public List<Player> getPlayers() {
        return players;
    }

    public Player getNext(Player currentPlayer) {
        final int currentIdx = players.indexOf(currentPlayer);
        if (currentIdx + 1 < players.size()) {
            return players.get(currentIdx + 1);
        }
        return players.get(0);
    }
}
```



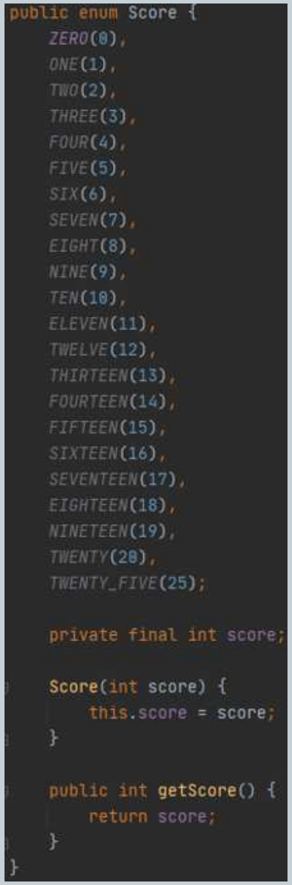
```java
final class Player {

    private final String name;

    public Player(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }
}
```

Result III

- Have a class which manages the score



```java
final class Scores {

    private final Map<Player, Integer> scores = new HashMap<>();

    public void add(Player player) {
        scores.put(player, 301);
    }

    public int getScore(Player player) {
        return scores.get(player);
    }

    public void add(Player player, Score... scores) {
        Arrays.stream(scores).forEach(score -> add(player, score));
    }

    private void add(Player player, Score score) {
        final int currentScore = scores.get(player);
        scores.put(player, currentScore - score.getScore());
    }
}
```



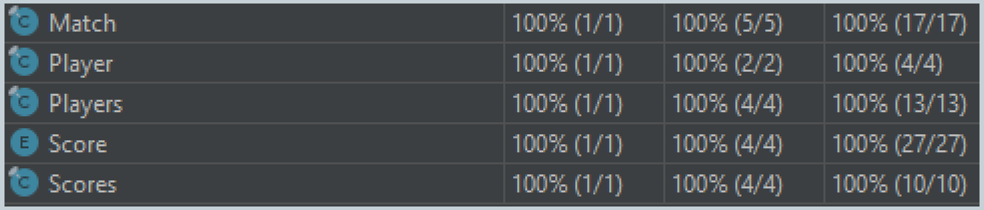
```java
public enum Score {
    ZERO(0),
    ONE(1),
    TWO(2),
    THREE(3),
    FOUR(4),
    FIVE(5),
    SIX(6),
    SEVEN(7),
    EIGHT(8),
    NINE(9),
    TEN(10),
    ELEVEN(11),
    TWELVE(12),
    THIRTEEN(13),
    FOURTEEN(14),
    FIFTEEN(15),
    SIXTEEN(16),
    SEVENTEEN(17),
    EIGHTEEN(18),
    NINETEEN(19),
    TWENTY(20),
    TWENTY_FIVE(25);

    private final int score;

    Score(int score) {
        this.score = score;
    }

    public int getScore() {
        return score;
    }
}
```

Problems

- Don’t think too far ahead, take one step at a time
- Don’t refactor too early

Conclusion

- Step by step approach by defining rules
- TDD
  - Solve one problem at the time
- TPP
  - Only add complexity when needed
- Object calistenics
  - Refactor the right way
- Coverage at 100%



Match	100% (1/1)	100% (5/5)	100% (17/17)
Player	100% (1/1)	100% (2/2)	100% (4/4)
Players	100% (1/1)	100% (4/4)	100% (13/13)
Score	100% (1/1)	100% (4/4)	100% (27/27)
Scores	100% (1/1)	100% (4/4)	100% (10/10)

Outlook

Current result looks very promising so…

- Complete Kata
- Create new Kata to add new features
- Adapt project with every lesson learned
- Create UI in JavaFX

Thank you!

- References:
  - «Agile Technical Practices Distilled» by Pedro Moreira Santos, Marco Consolaro & Alessandro Di Gioia
- Sources of old project:
  - https://github.com/Teazl/DartManager
- Sources of new project:
  - https://github.com/Teazl/DartManager2