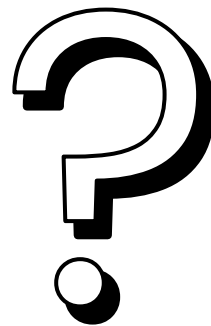





QUIZ GAME

Teste seus conhecimentos em programação
orientada a objetos!



TEAM



FELIPE MARTINS



JARDIANA GALVÃO

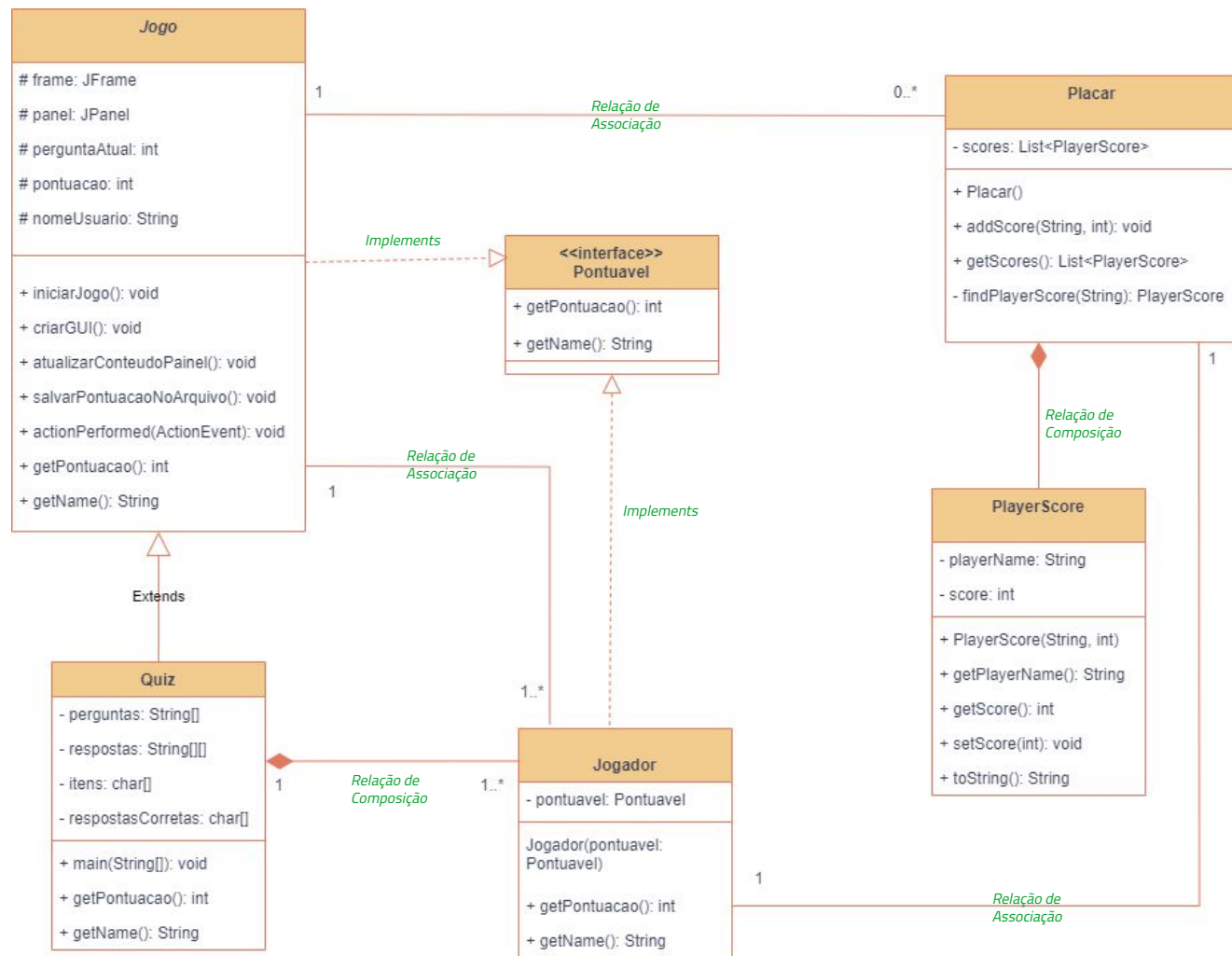


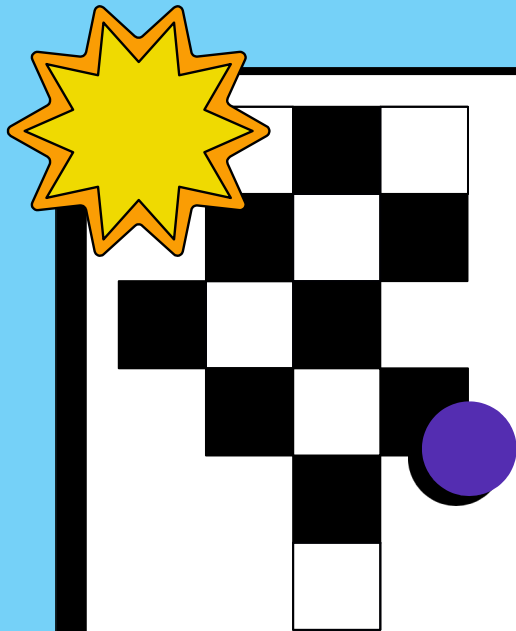
SAMUEL DE LIMA



SAUL SANTOS

DIAGRAMA DE CLASSES





REQUISITOS DO PROJETO

Abordar conceitos estudados na disciplina de POO



1. Classe Abstrata e Herança:

Classe Abstrata:

A classe abstrata é representada por `abstract class Jogo`. Ela é abstrata e contém métodos abstratos.



Esse é um tipo de classe especial que não pode ser instanciada, apenas herdada.

```
31 abstract class Jogo implements ActionListener, Pontuavel {
32     protected JFrame frame;
33     protected JPanel panel;
34     protected int perguntaAtual = 0;
35     protected int pontuacao = 0;
36     protected String nomeUsuario;
37
38     public void iniciarJogo() {
39         SwingUtilities.invokeLater(() -> {
40             criarGUI();
41         });
42     }
43
44     protected abstract void criarGUI();
45     protected abstract void atualizarConteudoPainel();
46     protected abstract void exibirPlacarLideres();
47     protected abstract void salvarPontuacaoNoArquivo();
48
49     @Override
50     public abstract void actionPerformed(ActionEvent e);
51 }
52
```

1. Classe Abstrata e Herança:

Herança:

A classe `Quiz` herda da classe abstrata `Jogo` com public class `Quiz extends Jogo`.



- Herança é o Paradigma de Orientação a Objetos que permite que uma classe (chamada de classe-filha) herde comportamentos de uma classe (chamada de classe-pai);
- A herança é implementada através da palavra reservada **`extends`**.

```
57 public class Quiz extends Jogo {
58
59     private JFrame frame;
60     private JPanel panel;
61     private int perguntaAtual = 0;
62     private int pontuacao = 0;
63     private String nomeUsuario;
64
65     String[] perguntas = {" "};
66
67     String[][] respostas = {" "};
68
69     char[] itens = {'A', 'B', 'C', 'D'};
70
71     char[] respostasCorretas = {'D', 'C', 'B', 'A', 'D', 'B', 'A', 'C', 'B', 'B'};
72
73     public static void main(String[] args) {
74         Quiz quiz = new Quiz();
75         quiz.iniciarJogo();
76         Jogador jogador = new Jogador(quiz);
77         System.out.println(jogador.getName());
78         System.out.println("O quiz vale de 0 a " + jogador.getPontuacao());
79     }
80
81     @Override
82     public int getPontuacao() {
83         return pontuacao;
84     }
85
86     @Override
87     public String getName() {
88         return nomeUsuario;
89     }
90 }
```

2. Interface e Implementação:

Interface:

A interface é representada por `interface Pontuavel`. Ela define métodos `getPontuacao` e `getName`.



- Uma interface é uma maneira de declarar o comportamento de uma classe. Nesta declaração não especificamos exatamente como acontece internamente cada comportamento.
- A interface possibilita a criação de classes que possuam apenas métodos com visibilidade pública.

```
8
9 interface Pontuavel {
10     int getPontuacao();
11     String getName();
12 }
13
14 class Jogador implements Pontuavel {
15     public Jogador(Pontuavel pontuavel) {
16     }
17
18     @Override
19     public int getPontuacao() {
20         return 10;
21     }
22
23     @Override
24     public String getName() {
25         return "Bem vindo Aluno de POO";
26     }
27 }
28
```

2. Interface e Implementação:

Implementação:

A classe `Jogador` implementa a interface `Pontuavel` com `class Jogador implements Pontuavel` e a classe `Jogo` implementa a interface `Pontuavel` com `abstract class Jogo implements ... Pontuavel`.



- As interfaces exigem que todos os métodos sejam implementados pelas classes que as implementam.

```
14 class Jogador implements Pontuavel {
15     public Jogador(Pontuavel pontuavel) {
16     }
17
18     @Override
19     public int getPontuacao() {
20         return 10;
21     }
22
23     @Override
24     public String getName() {
25         return "Bem vindo Aluno de POO";
26     }
27 }
28
29 abstract class Jogo implements ActionListener, Pontuavel {
30     protected JFrame frame;
31     protected JPanel panel;
32     protected int perguntaAtual = 0;
33     protected int pontuacao = 0;
34     protected String nomeUsuario;
35
36     public void iniciarJogo() {
37         SwingUtilities.invokeLater(() -> {
38             criarGUI();
39         });
40     }
```


3. Sobrescrita de Método:

Há sobrescrita de métodos nas classes `Jogador` (métodos `getPontuacao` e `getName`) e na classe `Quiz` (métodos `getPontuacao`, `getName`, `criarGUI`, `atualizarConteudoPainel`, `exibirPlacarLideres`, `salvarPontuacaoNoArquivo` e `actionPerformed`).



- Sobrescrever um método significa dar uma nova forma ao mesmo, uma nova versão. Ela ocorre quando uma classe filha fornece uma implementação específica para um método que já está definido na classe pai.

```
14 class Jogador implements Pontuavel {
15     public Jogador(Pontuavel pontuavel) {
16     }
17
18     @Override
19     public int getPontuacao() {
20         return 10;
21     }
22
23     @Override
24     public String getName() {
25         return "Bem vindo Aluno de POO";
26     }
27 }
```

```
55 public class Quiz extends Jogo {
56
57
58     @Override
59     public int getPontuacao() {
60         return pontuacao;
61     }
62
63     @Override
64     public String getName() {
65         return nomeUsuario;
66     }
67
68     @Override
69     protected void criarGUI() {
70         frame = new JFrame("");
71     }
72 }
```

4. Atributos Privados com Métodos Get e Set:

Todos os atributos nas classes `Jogo`, `Quiz`, `Jogador`, e `PlayerScore` são privados. Métodos `get` e `set` estão implementados para esses atributos.



Getters e setters são usados para proteger seus dados, especialmente na criação de classes. Para cada instância de variável, um método `getter` retorna seu valor, enquanto um método `setter` o define ou atualiza.

```
41 public static class PlayerScore {
42     private String playerName;
43     private int score;
44
45     public PlayerScore(String playerName, int score) {
46         this.playerName = playerName;
47         this.score = score;
48     }
49
50     public String getPlayerName() {
51         return playerName;
52     }
53
54     public int getScore() {
55         return score;
56     }
57
58     public void setScore(int score) {
59         this.score = score;
60     }
61
62     @Override
63     public String toString() {
64         return playerName + ": " + score;
65     }
66 }
```

5. Tratamento de Exceção:

Há tratamento de exceção na classe `Quiz` ao salvar a pontuação no arquivo (`salvarPontuacaoNoArquivo`).

O `IOException` é tratado usando `try` e `catch`.



O tratamento de exceções permite que você lide com erros e exceções sem interromper o fluxo normal do programa.

Duas das palavras-chave mais importantes no tratamento de exceções em Java são `try` e `catch`.

```
239
240
241 protected void salvarPontuacaoNoArquivo() {
242     try {
243         PrintWriter writer = new PrintWriter(new FileWriter("D:\\Temp\\ws-eclipse\\Quiz\\src\\pontuacao.txt", true));
244         writer.println("Nome: " + nomeUsuario + ", Pontuação: " + pontuacao);
245     } catch (IOException e) {
246         e.printStackTrace();
247     }
248 }
```

6. Interface Gráfica com o Usuário (GUI):

A interface gráfica está implementada na classe `Quiz`, especialmente no método `criarGUI`. Ela utiliza a biblioteca Swing para criar uma interface interativa



O componente GUI é um objeto com o qual o usuário interage através de, por exemplo: Mouse, Teclado, ou componentes como elementos desenhados na tela. Exemplos: botão, textbox, label, etc.

```
112     protected void criarGUI() {
113         frame = new JFrame("");
114         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
115         frame.setSize(750, 650);
116         frame.setResizable(false);
117
118         panel = new JPanel(new GridBagLayout());
119
120         JLabel titleLabel = new JLabel("POO Adventure - QuizGame");
121         titleLabel.setFont(new Font("Roboto", Font.BOLD, 30));
122
123         GridBagConstraints titleConstraints = new GridBagConstraints();
124         titleConstraints.gridx = 0;
125         titleConstraints.gridy = 0;
126         titleConstraints.insets = new Insets(50, 0, 30, 0);
127         panel.add(titleLabel, titleConstraints);
128
129         JButton playButton = new JButton("Play");
130
131         GridBagConstraints buttonConstraints = new GridBagConstraints();
132         buttonConstraints.gridx = 0;
133         buttonConstraints.gridy = 1;
134         buttonConstraints.insets = new Insets(10, 0, 10, 0);
135         panel.add(playButton, buttonConstraints);
136
137         buttonConstraints.gridy = 2;
138
139         frame.add(panel, BorderLayout.CENTER);
140
141         frame.setVisible(true);
142     }
```

COMO JOGAR:

- O jogo tem o objetivo de testar os conhecimentos do jogador sobre os conceitos de programação orientada a objetos.
- O jogo é composto por **10 questões** em que o jogador pode escolher entre os itens "a, b, c ou d" onde apenas uma delas é a correta.
- Após escolher um dos itens como resposta o jogo segue para a próxima questão sem mostrar se a resposta escolhida foi a correta e no final mostra o placar.



LET'S START THE GAME!

POO Adventure – Quiz Game