

Roteiro 03

Utilizaremos ainda o mesmo projeto, mas agora com um novo pacote (roteiro3.parte1). Agora já conhecemos algumas técnicas da OO e alguns princípios e boas práticas vistas nos roteiros 01 e 02. Iremos evoluir um projeto até chegar a um dos padrões, o Strategy.

Princípios (boas práticas) de Padrões de Projeto.

- 1 – Programe para uma interface e não para uma implementação
- 2 – Favoreça o uso de Composição sobre Herança

Cenário:

Suponha que estejamos criando um projeto para um jogo de esportes. Inicialmente após a entrevista com nosso cliente, identificamos a necessidade de criar dois tipos de jogadores. Um jogador de tênis e outro de futebol. Todo jogador possui um nome que o identifica, e o jogo permite que os jogadores treinem, compitam e definam suas próprias táticas para uso do jogo.

Início do projeto – Pacote : roteiro3.parte1

1 – Dentro do projeto criar um pacote chamado **roteiro3.parte1**

2 – Inicialmente temos a primeira classe chamada **Player**. Esta classe deve ser criada no pacote **roteiro3.parte1**.

```
package roteiro3.parte1;

public abstract class Player {
    private String nome;

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

    public void treinar(){
        System.out.println(this.nome + " Executando o treino !");
    }

    public void estiloCompetidor(){
        System.out.println(this.nome + " Muito competitivo !");
    }

    public abstract void definirTatica();
}
```

Observe que a classe tem um atributo **nome** que pode ser informado no construtor da classe. Além disso tem 2 métodos concretos e um abstrato :

- treinar() – permite executar o treino do jogador.
- estiloCompetidor() – permite informar o quanto o jogador é competitivo.
- definirTatica() – método abstrato para definir a tática do jogador. Como cada tipo de jogador terá diferentes táticas, definimos este método como abstrato e por consequência a classe também precisa ser abstrata.

3 – Devemos agora criar as classes **SoccerPlayer** e **TennisPlayer** como subclasses da classe **Player** através de herança.

```
package roteiro3.parte1;

public class SoccerPlayer extends Player{

    public SoccerPlayer (String nome) {
        super(nome);
    }

    @Override
    public void definirTatica() {
        System.out.println(super.nome + " Trabalha em equipe !");
    }
}
```

Atenção : Como precisamos usar o atributo nome dentro da subclasse, precisaremos alterar a clausula de acesso deste atributo de **private** para **protected** dentro da superclasse.

4 – O mesmo deve ser feito ao criar a classe **TennisPlayer**, e a tática deste jogador é definida com a mensagem abaixo.

```
System.out.println(super.nome + " Rebate a bola por cima do oponente");
```

5 – Agora crie uma classe para teste chamada **TesteJogo**.

```
package roteiro3.parte1;

public class TesteJogo {

    public static void main(String[] args) {

        System.out.println("Detalhes do Jogador de Tennis");
        TennisPlayer guga = new TennisPlayer("Gustavo Kuerten");
        guga.treinar();
        guga.estiloCompetidor();
        guga.definirTatica();
        System.out.println("*****");

        System.out.println("Detalhes do Jogador de Futebol");
        SoccerPlayer ronaldo = new SoccerPlayer("Ronaldinho Gaucho");
        ronaldo.treinar();
        ronaldo.estiloCompetidor();
        ronaldo.definirTatica();
        System.out.println("*****");  }
}
```

6 – Utilize uma ferramenta de software qualquer para geração do diagrama de classes para esta etapa do projeto (Sugestão : Astah Community). Obs.: Adicione aqui o diagrama para que seja disponibilizado no teams

Pacote : roteiro3.parte2

Imagine agora que o nosso projeto precise evoluir e precisamos de 3 novos tipos de jogadores. Um de xadrez, um de golf e outro de cartas, e assim teremos as classes : **ChessPlayer**, **GolfPlayer**, **CardPlayer** . Acontece também que precisaremos incluir uma nova habilidade para os jogadores. Trata-se da habilidade de correr, e já podemos perceber que nem todos os tipos de jogadores precisarão desta habilidade.

1 – No mesmo projeto crie o pacote roteiro3.parte2

2 – Copie todas as classes criadas na parte1 para o novo pacote.

3 – Inicialmente vamos criar o método correr() na classe Player como forma de implementar a nova habilidade de um jogador.

```
package roteiro3.parte2;

public abstract class Player {
    private String nome;

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

    public void treinar(){
        System.out.println(this.nome + " Executando o treino !");
    }
    public void estiloCompetidor(){
        System.out.println(this.nome + " Muito competitivo !");
    }

    public void correr(){
        System.out.println(this.nome + " Corre muito durante o jogo !");
    }
    public abstract void definirTatica();
}
}
```

4 – Crie as classes **ChessPlayer**, **GolfPlayer**, **CardPlayer** que também herdam da classe Player, semelhante as classes **SoccerPlayer** e **TennisPlayer**. Segue abaixo a sugestão de tática para cada jogador

```
System.out.println(super.nome + " - Domina o centro do tabuleiro");
```

Jogador de xadrez

```
System.out.println(super.nome + " - Pontua com o nº de tacadas previstas");
```

Jogador de golf

```
System.out.println(super.nome + " - É um jogador muito calmo");
```

Jogador de cartas

5 – Depois de implementadas as classes do item 4, não demora para percebermos que temos um problema de modelagem. Os jogadores de Xadrez, Carta e Golf também irão herdar o método **correr**, e isso não faz muito sentido dada a especificidades destes tipos de jogadores.

Uma possível solução seria sobrescrever o método **correr** nestas classes, usando o conceito de sobrecarga de método da OO. Certamente existem outras soluções, piores ou melhores, mas ainda assim esta é uma solução viável para testes.

Abaixo temos a sobrecarga do método na classe **ChessPlayer** e o mesmo pode ser feito nas outras 2 classes.

```
package roteiro3.parte2;

public class ChessPlayer extends Player{

    public ChessPlayer(String nome) {
        super(nome);
    }

    @Override
    public void definirTatica() {
        System.out.println(super.nome + " - Domina o centro do tabuleiro");
    }

    @Override
    public void correr() {
        System.out.println(super.nome + " - Não Corre ! Ele pensa !");
    }
}
```

Outra possibilidade é sobrescrever o método simplesmente sem implementação como segue abaixo. Não é a solução mais elegante, mas funciona.

```
@Override
public void correr() {
    // Sem implementação
}
```

A seguir faremos o teste das novas implementações

6 – Faça os devidos testes usando a classe **TesteJogo** acrescentando os novos jogadores e a habilidade de correr. Verifique se aconteceu o resultado esperado.

```
package roteiro3.parte2;

public class TesteJogo {

    public static void main(String[] args) {

        System.out.println("Detalhes do Jogador de Tennis");
        TennisPlayer guga = new TennisPlayer("Gustavo Kuerten");
        guga.treinar();
        guga.estiloCompetidor();
        guga.definirTatica();
        guga.correr();
        System.out.println("*****");

        System.out.println("Detalhes do Jogador de Futebol");
        SoccerPlayer ronaldo = new SoccerPlayer("Ronaldinho Gaucho");
        ronaldo.treinar();
        ronaldo.estiloCompetidor();
        ronaldo.definirTatica();
        ronaldo.correr();
        System.out.println("*****");

        System.out.println("Detalhes do Jogador de Cartas");
        CardPlayer joseCartas = new CardPlayer("Jose das Cartas");
        joseCartas.treinar();
        joseCartas.estiloCompetidor();
        joseCartas.definirTatica();
        joseCartas.correr();
        System.out.println("*****");

        System.out.println("Detalhes do Jogador de Xadrez");
        ChessPlayer kasparov = new ChessPlayer("Kasparov");
        kasparov.treinar();
        kasparov.estiloCompetidor();
        kasparov.definirTatica();
        kasparov.correr();
        System.out.println("*****");

        System.out.println("Detalhes do Jogador de Golf");
        GolfPlayer tigerwoods = new GolfPlayer("Tiger Woods");
        tigerwoods.treinar();
        tigerwoods.estiloCompetidor();
        tigerwoods.definirTatica();
        tigerwoods.correr();
        System.out.println("*****");
    }
}
```

7 – Utilize uma ferramenta de software qualquer para geração do diagrama de classes para esta etapa do projeto (Sugestão : Astah Community). Obs.: Adicione aqui o diagrama para que seja disponibilizado no teams

8 – Esta solução atendeu plenamente aos novos requisitos do projeto ? Quais são as suas críticas ?

Pacote : roteiro3.parte3

Faremos agora uma tentativa de melhoria neste modelo, no que diz respeito a duplicidade de código e talvez uma forma mais elegante de implementar métodos que não desejáveis em subclasses quando trabalhamos com herança. Neste caso, estamos falando do método **correr()**.

Faremos novamente o uso de uma interface para declarar o método **correr()**. Chamaremos esta interface de **Runnable** como forma de indicar que algumas classes podem implementar os métodos existentes nela e outras classes não.

1 - No mesmo projeto crie o pacote **roteiro3.parte3**

2 – Copie todas as classes criadas na parte2 para o novo pacote

3 – Crie a interface **Runnable** com a declaração do método **correr()** conforme segue abaixo.

```
package roteiro3.parte3;

public interface Runnable {

    public void correr();
}
```

4 – Uma vez criada a interface com o método **correr()** devemos retirar este método da classe **Player**, pois não faz mais sentido.

```
package roteiro3.parte3;

public abstract class Player {
    protected String nome;

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

    public void treinar(){
        System.out.println(this.nome + " Executando o treino !");
    }

    public void estiloCompetidor(){
        System.out.println(this.nome + " Muito competitivo !");
    }

    public void correr(){
        System.out.println(this.nome + " Corre muito durante o jogo !");
    }

    public abstract void definirTatica();
}
```

5 – Agora faremos com que as classes de jogadores específicos implementem a interface **Runnable** quando for adequado. Observe que agora os nossos jogadores herdam da classe **Player** e podem implementar ou não a classe **Runnable**. Para efeito de teste, segue abaixo 2 classes implementadas como exemplo. Vamos então implementar as classes : **SoccerPlayer**, **TennisPlayer**, **ChessPlayer**

```
package roteiro3.parte3;

public class SoccerPlayer extends Player implements Runnable{

    public SoccerPlayer(String nome) {
        super(nome);
    }

    @Override
    public void definirTatica() {
        System.out.println(super.nome + " Trabalha em equipe !");
    }

    @Override
    public void correr() {
        System.out.println(this.nome + " Corre muito durante o
jogo !");
    }
}
```

```
package roteiro3.parte3;

public class ChessPlayer extends Player implements Runnable{

    public ChessPlayer(String nome) {
        super(nome);
    }

    @Override
    public void definirTatica() {
        System.out.println(super.nome + " - Domina o centro do
tabuleiro");
    }

    @Override
    public void correr() {
        System.out.println(super.nome + " - Não Corre ! Ele pensa
!");
    }
}
```

6 – Se neste momento todas as refatorações foram concluídas, já podemos fazer o teste na classe **TesteJogo**. Obs.: Lembre-se que agora os jogadores de golf e de cartas não possuem o método **correr()**, e dessa forma a chamada deste método na classe de teste deve ser retirada, caso contrário teremos um erro de compilação.

7 – Utilize uma ferramenta de software qualquer para geração do diagrama de classes para esta etapa do projeto (Sugestão : Astah Community). Obs.: Adicione aqui o diagrama para que seja disponibilizado no teams

Seguimos para o pacote2.parte4 !

Pacote : roteiro3.parte4

A proposta adotada no roteiro3.parte3 não trouxe grandes melhorias. Ainda existe uma certa duplicação de código e o recurso de herança ainda continua comprometido. De qualquer forma, o uso de uma interface para implementar a habilidade de correr nos jogadores é uma boa ideia e pode ser melhorada.

Entendemos que a ação de correr além de ser uma habilidade do jogador, é também um comportamento que o jogador pode assumir ou não. Em outras palavras, podemos atribuir o comportamento de um rápido corredor a um objeto jogador em um dado momento. Em outro momento ele pode ser lento ou simplesmente não correr. Assim como um jogador de cartas não corre em momento algum.

Criaremos a seguir **uma interface** e **duas classes** que implementam esta interface.

- Interface **RunBehavior** – representa um comportamento para o jogador que precisa correr.
- Classe **RunFast** – representa uma classe que implementa **RunBehavior** e será usada para um jogador que precisa correr rápido.
- Classe **RunNoWay** - representa uma classe que implementa **RunBehavior** e será usada para um jogador que não precisa correr.

Perceba que poderíamos criar outras classes que implementam **RunBehavior**, como por exemplo um comportamento para um jogador lento que seria algo do tipo RunSlow. Mas por enquanto façamos apenas as duas classes indicadas acima.

ATENÇÃO : Desta vez faremos um pouco diferente. Vamos aproveitar o código feito no pacote **roteiro3.parte2** para gerar este novo pacote **roteiro3.parte4**. A solução da parte3 não ficou bom o suficiente para evoluirmos a partir dela.

- 1 - No mesmo projeto crie o pacote roteiro3.parte4
- 2 – Copie todas as classes criadas na **parte2** para o novo pacote
- 3 - Crie a interface **RunBehavior** com a declaração do método **correr()** conforme segue abaixo.

```
package roteiro3.parte4;

public interface RunBehavior {

    public void correr();

}
```


4 – Crie as classes **RunFast** e **RunNoWay** de forma que implementem a interface **RunBehavior**, conforme segue abaixo.

```
package roteiro3.parte4;

public class RunFast implements RunBehavior{

    @Override
    public void correr() {
        System.out.println("Corre muito rápido !");
    }
}
```

```
package roteiro3.parte4;

public class RunNoWay implements RunBehavior{

    @Override
    public void correr() {
        // Sem implementação
    }
}
```

Com esta estratégia de modelagem criamos uma hierarquia em cima da ação/comportamento de um jogador que corre. Temos então 2 hierarquias de classes. Uma em cima de **Player** e outra em cima de **RunBehavior**.

Assim, podemos isolar o comportamento de quem corre da nossa classe **Player**.

5 - Precisamos agora refatorar a classe **Player**. Lembre-se que agora o método `correr()` é de responsabilidade da hierarquia montada em cima de **RunBehavior**. Precisamos então criar uma associação entre **Player** e **RunBehavior**. Crie um atributo na classe **Player** do tipo **RunBehavior** como o nome **habilidadeCorrer**.

```
package roteiro3.parte4;

public abstract class Player {
    protected String nome;
    protected RunBehavior habilidadeCorrer;

    public Player(String nome, RunBehavior habilidadeCorrer) {
        this.nome = nome;
        this.habilidadeCorrer = habilidadeCorrer;
    }

    public void treinar(){
        System.out.println(this.nome + " Executando o treino !");
    }

    public void estiloCompetidor(){
        System.out.println(this.nome + " Muito competitivo !");
    }

    public void correr(){
        this.habilidadeCorrer.correr();
    }

    public abstract void definirTatica();
}
```

Para facilitar a manipulação do atributo **habilidadeCorrer**, observe que colocamos como parâmetro no construtor da classe.

Observe também que o método **correr()**, também precisa de uma refatoração. Então, por meio de delegação acessamos o método `correr` que na verdade está em **RunBehavior**.

6 – Como consequência da refatoração feita no item 5, devemos também acrescentar a passagem de parâmetro do atributo **habilidadeCorrer** nas subclasses de Player (**SoccerPlayer**, **TennisPlayer**, **GolfPlayer**, **ChessPlayer**, **CardPlayer**) .

Atenção : Nenhuma destas subclasses deve ter o método correr, pois isolamos esta funcionalidade (Itens 4 e 5). Assim sendo, qualquer destas classes que ainda tenha o método correr, deve ser retirado.

Segue abaixo a implementação de 2 destas classes como exemplo.

```
package roteiro3.parte4;
```

```
public class SoccerPlayer extends Player{
```

```
    public SoccerPlayer(String nome, RunBehavior habilidadeCorrer) {  
        super(nome, habilidadeCorrer);  
    }
```

```
    @Override
```

```
    public void definirTatica() {  
        System.out.println(super.nome + " Trabalha em equipe !");  
    }  
}
```

```
package roteiro3.parte4;
```

```
public class CardPlayer extends Player{
```

```
    public CardPlayer(String nome, RunBehavior habilidadeCorrer) {  
        super(nome, habilidadeCorrer);  
    }
```

```
    @Override
```

```
    public void definirTatica() {  
        System.out.println(super.nome + " - É um jogador muito calmo");  
    }  
}
```

Façamos os testes a seguir !!

7 – Faça os devidos testes usando a classe **TesteJogo** , e na criação de cada objeto jogador passamos como parâmetro a habilidade de correr adequada.

```
package roteiro3.parte4;

public class TesteJogo {

    public static void main(String[] args) {

        RunFast correRapido = new RunFast();
        RunNoWay naoCorre = new RunNoWay();

        System.out.println("Detalhes do Jogador de Tennis");
        TennisPlayer guga = new TennisPlayer("Gustavo Kuerten", correRapido);
        guga.treinar();
        guga.estiloCompetidor();
        guga.definirTatica();
        guga.correr();
        System.out.println("*****");

        System.out.println("Detalhes do Jogador de Futebol");
        SoccerPlayer ronaldo = new SoccerPlayer("Ronaldinho Gaucho", correRapido);
        ronaldo.treinar();
        ronaldo.estiloCompetidor();
        ronaldo.definirTatica();
        ronaldo.correr();
        System.out.println("*****");

        System.out.println("Detalhes do Jogador de Cartas");
        CardPlayer joseCartas = new CardPlayer("Jose das Cartas", naoCorre);
        joseCartas.treinar();
        joseCartas.estiloCompetidor();
        joseCartas.definirTatica();
        joseCartas.correr();
        System.out.println("*****");

        System.out.println("Detalhes do Jogador de Xadrez");
        ChessPlayer kasparov = new ChessPlayer("Kasparov", naoCorre);
        kasparov.treinar();
        kasparov.estiloCompetidor();
        kasparov.definirTatica();
        kasparov.correr();
        System.out.println("*****");

        System.out.println("Detalhes do Jogador de Golf");
        GolfPlayer tigerwoods = new GolfPlayer("Tiger Woods", naoCorre);
        tigerwoods.treinar();
        tigerwoods.estiloCompetidor();
        tigerwoods.definirTatica();
        tigerwoods.correr();
        System.out.println("*****");
    }
}
```

10 – Utilize uma ferramenta de software qualquer para geração do diagrama de classes para esta etapa do projeto (Sugestão : Astah Community). Obs.: Adicione aqui o diagrama para que seja disponibilizado no teams

11 – Neste modelo final em que chegamos, seria possível modificar a habilidade de correr de algum jogador em tempo de execução. Ex.: Um jogador que inicialmente não corre, podemos fazer com que ele passe a correr ? Se sim, como isso poderia ser feito ?