



Java Foundations

7-1 Criando uma Classe

ORACLE
Academy



Copyright © 2022, Oracle e/ou suas empresas afiliadas. Oracle, Java e MySQL são marcas comerciais registradas da Oracle Corporation e/ou de suas empresas afiliadas. Outros nomes podem ser marcas comerciais de seus respectivos proprietários.

Objetivos

- Esta lição abrange os seguintes objetivos:
 - Criar uma classe test/main Java
 - Crie uma classe Java em seu IDE
 - Usar condicionais em métodos
 - Traduzir especificações ou uma descrição em campos e comportamentos



Conceitos Orientados a Objetos

- Ficamos algum tempo testando as instruções condicionais e os loops
- Agora seria um bom momento para analisar os conceitos da programação orientada a objetos e seus benefícios
- O restante desta seção descreve detalhadamente a programação orientada a objetos

Exercício 1



- Execute Basic Puzzles 6 e 7
 - <https://objectstorage.uk-london-1.oraclecloud.com/n/lrvrlgaqj8dd/b/Games/o/JavaPuzzleBall/index.html>
 - Sua Meta: projetar uma solução que desvie a bola para o Duke
- Considere o seguinte:
 - O que acontece quando você coloca um ícone em um círculo azul?



ORACLE
Academy

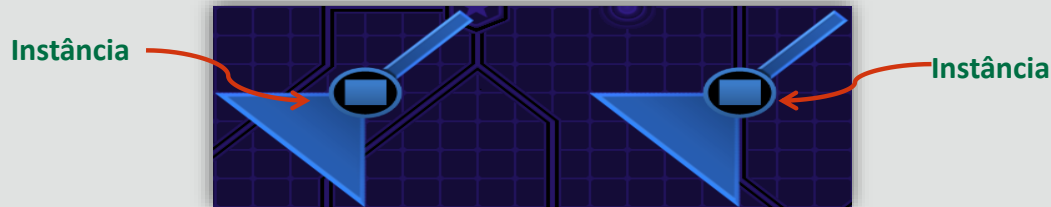
JFo 7-1
Criando uma Classe

Copyright © 2022, Oracle e/ou suas empresas afiliadas. Oracle, Java e MySQL são marcas comerciais registradas da Oracle Corporation e/ou de suas empresas afiliadas. Outros nomes podem ser marcas comerciais de seus respectivos proprietários.

5

Explicação Detalhada sobre o Java Puzzle Ball

- O que acontece quando você coloca ícones em um círculo azul?
 - Uma parede aparece em cada instância de um objeto blue bumper
 - As paredes dão aos bumpers comportamentos que desviam e interagem com a bola
 - Todas as instâncias do bumper azul compartilham esses mesmos comportamentos



ORACLE
Academy

JFo 7-1
Criando uma Classe

Copyright © 2022, Oracle e/ou suas empresas afiliadas. Oracle, Java e MySQL são marcas comerciais registradas da Oracle Corporation e/ou de suas empresas afiliadas. Outros nomes podem ser marcas comerciais de seus respectivos proprietários.

6

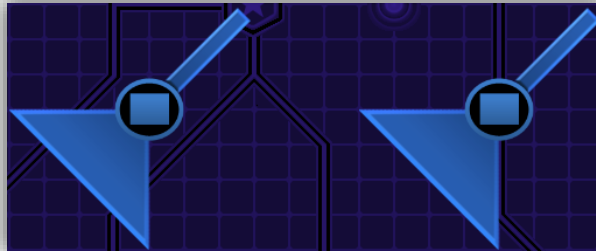
Um bumper azul é um objeto, e toda instância desses objetos compartilharão o mesmo comportamento ao interagir com a bola. Esses comportamentos podem incluir um desvio através do triângulo ou da parede simples.

Descrevendo um Objeto Blue Bumper



- Propriedades:

- Cor
- Forma
- Posição de x
- Posição de y
(Campos)



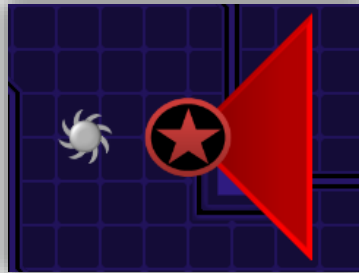
- Comportamentos:

- Fazer um som de ping
- Piscar
- Desviar a bola
- Ser destruído
(Métodos)



Lógica Condicional e Loops em Classes

- As condicionais e os loops também desempenham um papel importante nos métodos que você escreve para uma classe
- O método main era um local conveniente para testar e aprender a lógica condicional e os loops
- Mas lembre-se...
 - O método main pretende ser uma classe de driver
 - O programa inteiro não deve ser escrito no método main

E Se a Bola Colidir com um Bumper?



- Um método com a seguinte lógica será chamado:

```
public void onCollisionWithBall(Ball ball){  
    if(ball.isBlade == true){           //A bola é uma lâmina   
        getDestroyed();  
    }  
    else{                               //A bola não é uma lâmina   
        deflectBall();  
    } //fim if`  
} //fim do método main
```

O `RedBumper` tem um método para tratar colisões. Quando esse método é chamado, ele verifica se a bola é uma lâmina. `isBlade` é uma propriedade booleana que pertence à classe `Ball`. Se a bola for uma lâmina, o bumper será destruído. Caso contrário, a bola será desviada. Ser destruído e desviar a bola são comportamentos de um Bumper. Isso torna-se possível com os métodos `getDestroyed()` e `deflectBall()`.

Modelando uma Caderneta de Poupança

- Você poderia modelar uma caderneta de poupança desta forma:

```
public class SavingsAccount{  
    public static void main(String args[]){  
        int balance = 1000;  
        String name = "Damien";  
    }//fim do método main  
}//fim da classe SavingsAccount
```

- E duas contas como esta:

```
int balance1 = 1000;  
String name1 = "Damien";  
  
int balance2 = 2000;  
String name2 = "Bill";    //Copiar, Colar, Renomear
```

Modelando Muitas Contas

- Como você modelaria 1000 contas?

```
...  
//Você pensa...  
//Eu realmente preciso copiar e colar 1000 vezes?
```

- Como você adicionaria um parâmetro para cada conta?

```
...  
//Você pensa...  
//Tem que haver uma maneira melhor!
```

- Existe uma maneira melhor: use uma classe
 - E não o método main

Como Estruturar uma Classe

- O código precisa caber neste formato:

```
1 public class SavingsAccount {  
2  
3     Propriedades  
4  
5  
6     Comportamentos  
7  
8  
9 }
```

Como Estruturar uma Classe

- O código precisa caber neste formato:

```
1 public class SavingsAccount {  
2     public double balance;  
3     public double interestRate = 0.01;  
4     public String name;  
5  
6     public void displayCustomer(){  
7         System.out.println("Cliente: " + name);  
8     } //fim do método displayCustomer  
9 } //fim da classe SavingsAccount
```

- Com uma única linha de código (linha 3), todas as 1000 contas têm uma taxa de juros
 - E podemos alterar a taxa a qualquer momento para qualquer conta

O Método Main como uma Classe de Driver

- Coloque o método main em uma classe de teste
 - O método main costuma ser usado para instanciação

```
public class AccountTest {  
    public static void main(String[] args){  
  
        SavingsAccount sa0001 = new SavingsAccount();  
        sa0001.balance = 1000;  
        sa0001.name = "Damien";  
        sa0001.interestRate = 0.02;  
  
        SavingsAccount sa0002 = new SavingsAccount();  
        sa0002.balance = 2000;  
        sa0002.name = "Bill";  
    } //fim do método main  
} //fim da classe AccountTest
```

Exercício 2

- Crie um novo projeto Java
- Crie uma classe `AccountTest` com um método `main`
- Crie uma classe `CheckingAccount`
 - Inclua campos para `balance` e `name`
- Instancie um objeto `CheckingAccount` no método `main`
 - Atribua valores aos campos `balance` e `name` desse objeto

Escopo da Variável

- Os campos podem ser acessados de qualquer local em uma classe
 - Isso significa de dentro dos métodos

```
public class SavingsAccount {  
    public double balance;  
    public double interestRate;  
    public String name;  
  
    public void displayCustomer(){  
        System.out.println("Cliente: " + name);  
        System.out.println("Saldo: " + balance);  
        System.out.println("Taxa: " + interestRate);  
    } //fim do método displayCustomer  
} //fim da classe SavingsAccount
```


Escopo da Variável

- As variáveis criadas dentro de um método não podem ser acessadas fora desse método
 - Isso inclui parâmetros de métodos

```
public class SavingsAccount {  
    public double balance;  
    public double interestRate;  
    public String name;
```

```
    public void deposit(int x){  
        balance += x;  
    }//fim do método deposit
```

Escopo de x



```
    public void badMethod(){  
        System.out.println(x);  
    }//fim do método badMethod  
}//fim da classe SavingsAccount
```

Não escopo de x

Acessando Campos e Métodos de Outra Classe

1. Crie uma instância
2. Use o operador dot (.)

```
public class AccountTest {  
    public static void main(String[] args){  
        1) SavingsAccount sa0001 = new SavingsAccount();  
        2) { sa0001.name = "Damien";  
            sa0001.deposit(1000);  
        }  
    }  
}  
//fim da classe AccountTest
```


```
public class SavingsAccount {  
    public String name;  
    public double balance;  
  
    public void deposit(int x){  
        balance += x;  
    }  
}  
//fim do método deposit  
//fim da classe SavingsAccount
```

Passando Valores a Métodos

- 1000 é passado para o método deposit()
- O valor de x torna-se 1000

```
public class AccountTest {  
    public static void main(String[] args){  
        SavingsAccount sa0001 = new SavingsAccount();  
        sa0001.name = "Damien";  
        sa0001.deposit(1000);  
    } //fim da classe AccountTest
```

```
public class SavingsAccount {  
    public String name;  
    public double balance;  
  
    public void deposit(int x){  
        balance += x;  
    } //fim do método deposit  
} //fim da classe SavingsAccount
```



x = 1000

Exercício 3

- Continue a editar o projeto `AccountTest`
- Escreva um método `withdraw()` para verificar contas que...
 - Aceitem um argumento `double` do valor a ser retirado
 - Imprimam um aviso se o saldo for muito baixo para fazer a retirada
 - Imprimam um aviso se o argumento de retirada for negativo
 - Se não houver avisos, o valor da retirada será subtraído do saldo Imprima o novo saldo
- Teste este método com a instância do Exercício 2

E Se Eu Precisar de um Valor de um Método?

- As variáveis são limitadas pelo respectivo escopo
- Mas é possível obter o valor dessas variáveis com base em um método

```
public class SavingsAccount {  
    public double balance;  
    public double interestRate;  
    public String name;
```

```
    public void calcInterest(){  
        double interest = balance*interestRate/12;  
  
    }//fim do método calcInterest
```

Escopo de
juros

```
}//fim da classe SavingsAccount
```

Retornando Valores de Métodos

- Se você quiser obter um valor de um método...
 - Escreva uma instrução de retorno
 - Altere o tipo do método de void para o tipo a ser retornado

```
public class SavingsAccount {  
    public double balance;  
    public double interestRate;  
    public String name;  
  
    //Este método tem um tipo de retorno double  
    public double calcInterest(){  
        double interest = balance * interestRate / 12;  
        return interest;  
    } //fim do método calcInterest  
} //fim da classe SavingsAccount
```

Retornando Valores: Exemplo

- Quando `getInterest()` retorna um valor...

```
public class AccountTest {  
    public static void main(String[] args){  
        SavingsAccount sa0001 = new SavingsAccount();  
        sa0001.balance = 1000;  
        sa0001.balance += sa0001.calcInterest();  
    } //fim da classe AccountTest
```

- Isso é equivalente a escrever...

```
public class AccountTest {  
    public static void main(String[] args){  
        SavingsAccount sa0001 = new SavingsAccount();  
        sa0001.balance = 1000;  
        sa0001.balance += 0.83;  
    } //fim da classe AccountTest
```

- Mas é melhor e mais flexível porque o valor é calculado, em vez de ser codificado

Resumo Sobre Métodos

```
public double calculate(int x, double y){  
    double quotient = x/y;  
    return quotient;  
}//fim do método calculate
```


Limitando o Método Main

- O método main deve ser o mais curto possível
- O exemplo abaixo não é muito bom porque...
 - Aumentar o saldo de uma conta com base nos juros é um comportamento típico das contas
 - O código desse comportamento deve ser escrito como um método dentro da classe SavingsAccount
 - Também é perigoso ter um programa de conta em que o campo do saldo possa ser livremente manipulado

```
public class AccountTest {  
    public static void main(String[] args){  
        SavingsAccount sa0001 = new SavingsAccount();  
        sa0001.balance = 1000;  
        sa0001.balance += sa0001.calcInterest();  
    } //fim do método main  
}
```

O Restante desta Seção

- Aprenderemos a evitar esses cenários problemáticos ao desenvolver uma classe
- Mas, para esta lição, basta entender como:
 - Interpretar uma descrição ou uma especificação
 - Quebrá-la em propriedades e comportamentos
 - Traduzir essas propriedades e esses comportamentos em campos e métodos

Exercício 4

- Continue a editar o projeto `AccountTest`
- Crie uma nova classe de acordo com a descrição
- Certifique-se de instanciar essa classe e testar seus métodos
 - Crie um Título do Tesouro
 - Uma pessoa pode comprar um título por qualquer período entre 1 e 60 meses
 - Um título rende juros todo mês até seu prazo terminar (0 mês restante)
 - O prazo e a taxa de juros são definidos ao mesmo tempo
 - A taxa de juros do título baseia-se em seu prazo de acordo com o seguinte sistema de níveis:

0–11 months : 0,5%
12–23 months : 1,0%
24–35 months : 1,5%
36–47 months : 2,0%
48–60 months : 2,5%

Descrevendo um Título do Tesouro

- Propriedades:

- Nome
- Saldo
- Prazo
- Meses Restantes
- Taxa de Juros



- Comportamentos:

- Definir a Taxa de Juros com Base no Prazo
- Receber Juros
- Vencer (0 mês restante)

Você pode ter proposto esses campos e comportamentos.

Traduzindo para o Código Java: Parte 1

- Sua classe Bond pode ter representado campos como este:

```
public class Bond{  
    public String name;  
    public double balance, rate;  
    public int term, monthsRemaining;
```

O código continua no próximo slide...

Talvez você tenha escrito seu programa desta forma: A tradução dos campos em tipos de dados é um exercício fácil.

Traduzindo para o Código Java: Parte 2

- E incluir os seguintes métodos:

```
public void setTermAndRate(int t){  
    if(t>=0 && t<12)  
        rate = 0.005;  
    else if(t>=12 && t<24)  
        rate = 0.010;  
    else if(t>=24 && t<36)  
        rate = 0.015;  
    else if(t>=36 && t<48)  
        rate = 0.020;  
    else if(t>=48 && t<=60)  
        rate = 0.025;  
    else{  
        System.out.println("Prazo Inválido");  
        t = 0;  
    }  
    term = t;  
    monthsRemaining = t;  
} //fim do método setTermAndRate
```

O código continua no próximo slide...

A taxa depende do prazo. É possível usar uma construção if/else para verificar o valor do prazo e atribuir a taxa correta com base nesse prazo. Também é útil verificar se o prazo é inválido. Nesse caso, o prazo é definido como 0 para impedir um comportamento inadequado da conta.

Traduzindo para o Código Java: Parte 3

```
public void earnInterest(){
    if(monthsRemaining > 0){
        balance += balance * rate / 12;
        monthsRemaining--;
        System.out.println("Saldo: $" + balance);
        System.out.println("Taxa: " + rate);
        System.out.println("Meses Restantes: "
                           + monthsRemaining);
    }
    else{
        System.out.println("Título Vencido");
    } //fim if
} //fim do método earnInterest
} //fim da classe Bond
```

Rendimento é o último comportamento que você precisa traduzir para um método. Uma instrução if/else é útil aqui também. Se houver meses restantes no prazo do CD, o juro será adicionado e um mês a menos está restando para receber os juros. Se não houver meses restantes, o CD está vencido e não poderá receber juros.

Resumo

- Nesta lição, você deverá ter aprendido a:
 - Criar uma classe test/main Java
 - Crie uma classe Java em seu IDE
 - Usar condicionais em métodos
 - Traduzir especificações ou uma descrição em campos e comportamentos



The Oracle Academy logo is centered on a light gray background. It features the word "ORACLE" in a bold, orange, sans-serif font. Below it, the word "Academy" is written in a smaller, dark gray, sans-serif font. The entire logo is framed by two horizontal dark gray bars, one at the top and one at the bottom.

ORACLE

Academy