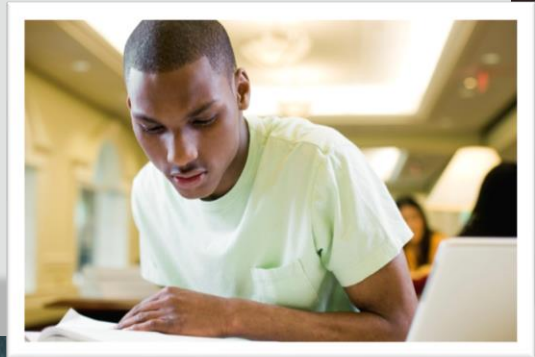




Java Foundations

7-4 Sobrepondo Métodos

ORACLE
Academy



Copyright © 2020, Oracle e/ou suas empresas afiliadas. Todos os direitos reservados.

Objetivos

- Esta lição abrange os seguintes objetivos:
 - Entender os efeitos de vários construtores em uma classe
 - Definir a sobreposição de um método
 - Explicar a assinatura do método
 - Entender quando a sobreposição é e não é possível



Tópicos

- **Sobrepondo Construtores**
- Sobrepondo Métodos
- Quando a Sobreposição Não É Possível



ORACLE
Academy

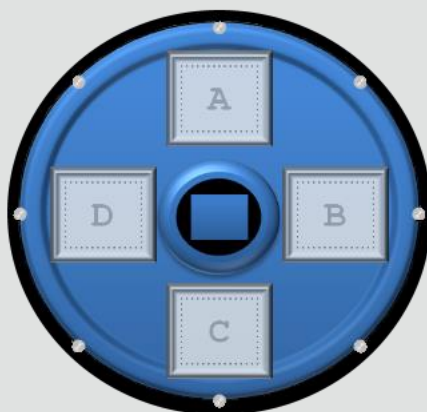
JFo 7-4
Sobrepondo Métodos

Copyright © 2020, Oracle e/ou suas empresas afiliadas. Todos os direitos reservados.



Exercício 1

- Execute o Basic Puzzle 8
- Considere o seguinte:
 - O que você pode dizer sobre as luzes ao redor de cada círculo?



Por que Adicionamos Luzes aos Círculos?

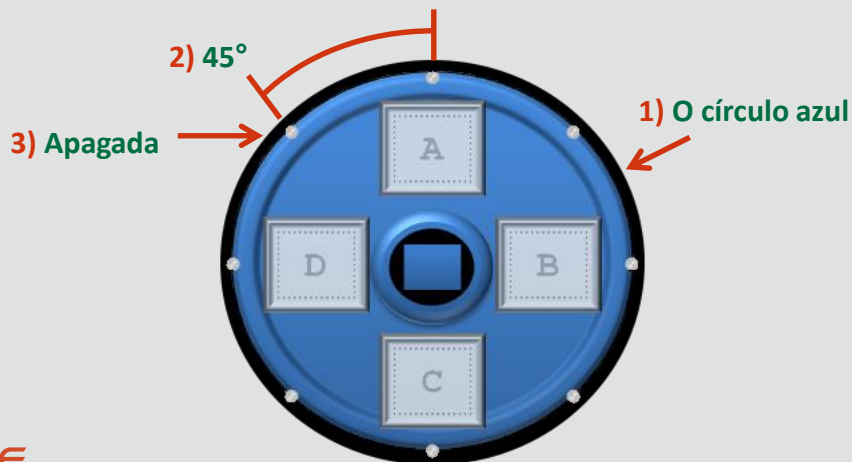
- Os builds anteriores não continham essas luzes
 - Elas nunca fizeram parte do projeto original
 - Então por que foram adicionadas?
- As luzes foram adicionadas para minimizar a confusão dos jogadores
 - Alguns jogadores não percebiam que o círculo giraria até o ângulo de 45° mais próximo
 - Alguns jogadores precisavam girar o círculo várias vezes até atingirem o próximo incremento de 45°
 - Isso gerou confusão e frustração porque os jogadores pensavam: **"O círculo não gira para onde eu quero que gire"**

O Plano para Solucionar Esses Problemas

- Adicione oito luzes a cada círculo
 - As luzes funcionam como uma marcação
 - Elas mostram cada incremento de 45° em que o círculo poderia girar
- Cada luz pode brilhar, o que indica:
 - A rotação em que o círculo foi arrastado
 - A rotação em que o círculo girará se for liberado

Propriedades das Luzes

- Uma luz requer as seguintes propriedades:
 - O círculo a que ela pertence
 - Sua rotação ao redor desse círculo
 - Se ela deve estar acesa



Programando a Classe Light

- Veja a seguir uma versão simplificada dessa classe:

```
public class UIWheelLight {  
    //Campos  
    public UIWheel wheel;  
    public double rotation;  
    public boolean isLit;  
  
    //Construtor  
    public UIWheelLight(UIWheel w, double r, boolean l){  
        wheel = w;  
        rotation = r;  
        isLit = l;  
    } //fim Construtor  
} //fim da classe UIWheelLight
```

Chamando o Construtor UIWheelLight

- Uma chamada de construtor teria uma aparência como esta:

```
UIWheelLight light1 = new UIWheelLight(blueWheel, 45, false);
```

- Mas aí pensamos: “Estou com muita preguiça de digitar tudo isso!”
 - Existe um motivo legítimo para isso
 - Não é porque somos programadores ruins
 - Nem porque somos inexperientes



Por que É Ótimo Ser Preguiçoso

- Um pequeno cálculo matemático informou-nos...
 - Existem oito luzes em um círculo
 - E uma luz adicional aparecerá acesa
 - 8/9 (ou 89%) das luzes serão instanciadas apagadas
 - 89% é uma maioria significativa
- Portanto, o último argumento é redundante e complicará o código 89% das vezes
- Um código complicado é ruim e deve ser evitado

```
UIWheelLight light1 = new UIWheelLight(blueWheel, 45, false);
```

Redundante

Na verdade, as luzes não acendem e apagam. Em vez disso, quando uma luz precisa estar "acesa", instanciamos uma nona luz e posicionamos-a sobre a luz "apagada" correspondente.

Sobrepondo Construtores

- Você pode criar mais de um construtor em uma classe
 - Esse procedimento é conhecido como sobreposição de um construtor
 - Uma classe pode ter um número ilimitado de construtores
- Cada construtor sobreposto recebe o mesmo nome
- A diferença entre eles está nos seguintes aspectos:
 - Número de parâmetros
 - Tipos de parâmetros
 - Ordem de parâmetros

Construtores Sobrepostos: Exemplo

- A implementação dessa estratégia na classe UIWheelLight é semelhante a algo como o seguinte:

```
public class UIWheelLight { 2 parâmetros
    ...
    //Construtores
    public UIWheelLight(UIWheel w, double r){
        wheel = w;
        rotation = r;
        isLit = false;
    }//fim Construtor
    public UIWheelLight(UIWheel w, double r, boolean l){ 3 parâmetros
        wheel = w;
        rotation = r;
        isLit = l;
    }//fim Construtor
} //fim da classe UIWheelLight
```

Chamando Construtores Sobrepostos

- Um objeto pode ser instanciado chamando qualquer um de seus construtores de classe
- Você fornece os argumentos, e o Java localiza o construtor mais apropriado

```
UIWheelLight light1 = new UIWheelLight(blueWheel, 45);
```

```
UIWheelLight light1 = new UIWheelLight(blueWheel, 45, false);
```



Exercício 2

- Continue a editar o projeto PrisonTest
 - Uma versão deste programa é fornecida para você
- Sobreponha o construtor existente
 - Crie seu próprio construtor com argumento zero
 - Chamar esse construtor deve inicializar campos com os valores a seguir
 - Instancie um objeto com seu construtor

	Variável:	p02
	Nome:	null
	Altura:	0.0
	Sentença:	0

Reconhecendo a Redundância em Construtores

- Um código muito semelhante é repetido nesses construtores
- É possível minimizar essa redundância

```
public class UIWheelLight {  
    public UIWheelLight(UIWheel w, double r){  
        wheel = w;  
        rotation = r;  
        isLit = false;  
    } //fim construtor  
  
    public UIWheelLight(UIWheel w, double r, boolean l){  
        wheel = w;  
        rotation = r;  
        isLit = l;  
    } //fim construtor  
} //fim da classe UIWheelLight
```

Primeira ocorrência

Repetida

Construtores Podem Chamar Outros Construtores

- Ao usar a palavra-chave `this`, um construtor pode chamar outro

```
public class UIWheelLight {  
    //Construtores  
    public UIWheelLight(UIWheel w, double r){  
        this(w, r, false);  
    }//fim construtor  
  
    public UIWheelLight(UIWheel w, double r, boolean l){  
        wheel = w;  
        rotation = r;  
        isLit = l;  
    }//fim construtor  
}//fim da classe UIWheelLight
```

Isso é útil porque, se a lógica em um construtor precisar mudar, o código só precisará ser alterado em um local.

Tópicos

- Sobrepondo Construtores
- **Sobrepondo Métodos**
- Quando a Sobreposição Não É Possível



ORACLE
Academy

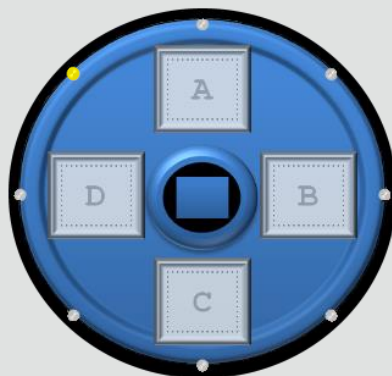
JFo 7-4
Sobrepondo Métodos

Copyright © 2020, Oracle e/ou suas empresas afiliadas. Todos os direitos reservados.

18

Comportamento da Luz

- Dependendo de onde você clique, a luz amarela comporta-se ligeiramente diferente
 - Se você clicar no círculo, a luz será posicionada com base na localização do cursor do mouse
 - Se você clicar no orifício A, B, C ou D, a luz será posicionada com base no centro desse orifício



Como Programamos Esta Diferença Sutil no Comportamento?

- Sobreposemos o método responsável por posicionar a luz amarela
- O código é semelhante a este:

```
public class UIWheelLight {  
    ...  
    public void setPosition(double x, double y){  
        //Fazer um cálculo  
    }//fim do método setPosition  
  
    public void setPosition(double x, double y, UISlot s){  
        //Fazer um cálculo um pouco diferente  
    }//fim do método setPosition  
}//fim da classe UIWheelLight
```

x e y são as posições x e y em que o mouse foi clicado.

Sobrepondo Métodos

- Qualquer método pode ser sobreposto, inclusive...
 - Construtores
 - Métodos que modelam comportamentos de objetos
 - Métodos que fazem cálculos
- Todas as versões de um método sobrecarregado recebem o mesmo nome
- A diferença entre eles está nos seguintes aspectos:
 - Número de parâmetros
 - Tipos de parâmetros
 - Ordem de parâmetros

Número de Parâmetros

- Cada método sobreposto a seguir tem um número diferente de parâmetros

```
public class Calculator {  
  
    public double sum(double num1){  
        return num1;  
    }//fim do método sum  
  
    public double sum(double num1, double num2){  
        return num1 + num2;  
    }//fim do método sum  
  
    public double sum(double num1, double num2, double num3){  
        return num1 + num2 + num3;  
    }//fim do método sum  
  
}//fim da classe Calculator
```

Tipo de Parâmetros

- Cada método sobreposto a seguir tem parâmetros de tipos diferentes

```
public class Calculator {  
  
    public double sum(double num1, double num2){  
        return num1 + num2;  
    }//fim do método sum  
  
    public double sum(int num1, int num2){  
        return num1 + num2;  
    }//fim do método sum  
  
}//fim da classe Calculator
```

Ordem de Parâmetros

- Cada método sobrecarregado tem parâmetros em uma ordem diferente

```
public class Calculator {  
  
    public double sum(int num1, double num2){  
        return num1 + num2;  
    }//fim do método sum  
  
    public double sum(double num1, int num2){  
        return num1 + num2;  
    }//fim do método sum  
  
}//fim da classe Calculator
```


Chamando Métodos Sobrecarregados

- Você fornece os argumentos, e o Java localiza o método mais apropriado

```
public class CalculatorTest{  
  
    public static void main(String[] args){  
        Calculator calc = new Calculator();  
  
        calc.sum(1, 2);  
        calc.sum(1, 2, 3);  
        calc.sum(1.5, 4.5);  
    }//fim do método main  
  
}//fim da classe CalculatorTest
```



Exercício 3

- Continue a editar o projeto `PrisonTest`
- Escreva um método que imprima todo campo `Prisoner`
 - Esse deve ser um método com argumento zero
- Sobreponha esse método para aceitar um argumento booliano
 - Se o booliano for verdadeiro, esse método deverá chamar o método `think()`
- Chame as duas versões desse método em um objeto

Reconhecendo a Redundância em Métodos

- Um código muito semelhante é repetido nesses métodos
- É possível minimizar essa redundância

```
public class Calculator{  
    ...  
    public double calcY(double m, double x){  
        double y = 0;  
        y = mx;  
        return y; ;  
    }//fim do método calcY  
    public double calcY(double m, double x, double b){  
        double y = 0;  
        y = mx + b;  
        return y;  
    }//fim do método calcY  
}//fim da classe Calculator
```

First occurrence

Repetida

Observação para os Instrutores: o código nestes dois slides deve estar na mesma posição para que suas pequenas diferenças tornem-se mais aparentes.

Os Métodos Podem Chamar Outros Métodos na Mesma Classe

- Neste exemplo, um método retorna um valor para o outro

```
public class Calculator{  
    ...  
    public double calcY(double m, double x){  
        return calcY(m,x,0);  
    }//fim do método calcY  
  
    public double calcY(double m, double x, double b){  
        double y = 0;  
        y = mx + b;  
        return y;  
    }//fim do método calcY  
}//fim da classe Calculator
```

Isso é útil porque, se os cálculos estiverem errados ou precisarem ser ajustados, o código precisará ser alterado uma única vez.

Observação para os Instrutores: o código nestes dois slides deve estar na mesma posição para que suas pequenas diferenças tornem-se mais aparentes.



Exercício 4

- Continue a editar o projeto `PrisonerTest`
- Identifique e minimize qualquer código repetido no construtor e nos métodos `display()`
- Execute o programa para ter certeza de que ele está funcionando corretamente

Tópicos

- Sobrepondo Construtores
- Sobrepondo Métodos
- **Quando a Sobreposição Não É Possível**



ORACLE
Academy

JFo 7-4
Sobrepondo Métodos

Copyright © 2020, Oracle e/ou suas empresas afiliadas. Todos os direitos reservados.

30

A Assinatura do Método

- Uma assinatura de método é criada com base...
 - No nome do método
 - No número de parâmetros
 - No tipo de parâmetros
 - Na ordem dos parâmetros
- A assinatura de um método será única desde que um desses itens acima seja diferente

Essa é a assinatura do método

```
public void setPosition(double x, double y){  
    //Fazer um cálculo  
} //fim do método setPosition
```

Observação para os Instrutores: o código nestes dois slides deve estar na mesma posição para que suas pequenas diferenças tornem-se mais aparentes.

Não É Assinatura do Método

- A assinatura do método não inclui...
 - O nome dos parâmetros
 - O tipo de retorno do método
- A alteração de um desses itens acima não é suficiente para sobrepor um método

Estes itens não fazem parte da assinatura do método

```
public void setPosition(double x, double y){  
    //Fazer um cálculo  
} //fim do método setPosition
```

Observação para os Instrutores: o código nestes dois slides deve estar na mesma posição para que suas pequenas diferenças tornem-se mais aparentes.

Correspondendo Chamadas de Métodos para Assinaturas

- Neste exemplo, a contagem faz com que seja mais fácil ver qual versão de `sum()` deve ser chamada
- A chamada do método tem três argumentos
- Qual assinatura do método tem três parâmetros?

```
sum(1, 2, 3);
```

```
public class Calculator {  
  
    public double sum(double num1, double num2){  
        return num1 + num2;  
    }//fim do método sum  
    public double sum(double num1, double num2, double num3){  
        return num1 + num2 + num3;  
    }//fim do método sum  
}//fim da classe Calculator
```

Observação para os Instrutores: o código nestes quatro slides deve estar na mesma posição para que suas pequenas diferenças tornem-se mais aparentes.

Falta de Correspondência entre Nomes de Parâmetros

- Você consegue dizer qual versão de `sum()` deverá ser chamada se os nomes dos parâmetros forem diferentes?
 - Não
 - E o Java também não consegue

```
sum(1, 2);
```

```
public class Calculator {  
  
    public double sum(double num1, double num2){  
        return num1 + num2;  
    }//fim do método sum  
    public double sum(double x, double y){  
        return x + y;  
    }//fim do método sum  
}//fim da classe Calculator
```

Observação para os Instrutores: o código nestes quatro slides deve estar na mesma posição para que suas pequenas diferenças tornem-se mais aparentes.

Falta de Correspondência entre Tipos de Retorno

- Você consegue dizer qual versão de `sum()` deverá ser chamada se os tipos de retorno forem diferentes?
 - Não
 - E o Java também não consegue

```
sum(1, 2);
```

```
public class Calculator {  
  
    public double sum(double num1, double num2){  
        return num1 + num2;  
    }//fim do método sum  
    public int sum(double num1, double num2){  
        return num1 + num2;  
    }//fim do método sum  
}//fim da classe Calculator
```

ORACLE
Academy

JFo 7-4
Sobrepondo Métodos

Copyright © 2020, Oracle e/ou suas empresas afiliadas. Todos os direitos reservados.

35

Observação para os Instrutores: o código nestes quatro slides deve estar na mesma posição para que suas pequenas diferenças tornem-se mais aparentes.

Sobreposição Primeiro

- Os métodos não serão devidamente sobrecarregados até suas assinaturas serem diferentes
- Quando isso acontecer, então você deverá modificar o tipo de retorno e os nomes dos parâmetros

```
sum(1, 2);
```

```
public class Calculator {  
  
    public double sum(double num1, double num2){  
        return num1 + num2;  
    }//fim do método sum  
    public int sum(double num1, double num2, double num3){  
        return num1 + num2 + num3;  
    }//fim do método sum  
}//fim da classe Calculator
```

Observação para os Instrutores: o código nestes quatro slides deve estar na mesma posição para que suas pequenas diferenças tornem-se mais aparentes.

Resumo dos Métodos de Sobreposição

- Têm o mesmo nome
- Têm assinaturas diferentes:
 - O número de parâmetros
 - Os tipos de parâmetros
 - A ordem de parâmetros
- Podem ter uma funcionalidade diferente ou semelhante

Resumo

- Nesta lição, você deverá ter aprendido a:
 - Entender os efeitos de vários construtores em uma classe
 - Definir a sobreposição de um método
 - Explicar a assinatura do método
 - Entender quando a sobreposição é e não é possível



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