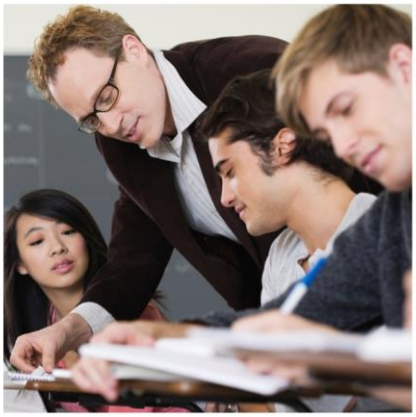
ORACLE

Academy

Java Foundations

7-3
Construtores



ORACLE
Academy

Objetivos

- Esta lição abrange os seguintes objetivos:
  - Entender valores padrão
  - Travar o programa com uma referência null
  - Entender o construtor padrão
  - Escrever um construtor que aceite argumentos
  - Inicializar campos com um construtor
  - Usar this como uma referência a objeto

Lembre-se da Classe Prisoner

- Ela pode ter sido parecida com este código
- Ela contém campos e métodos

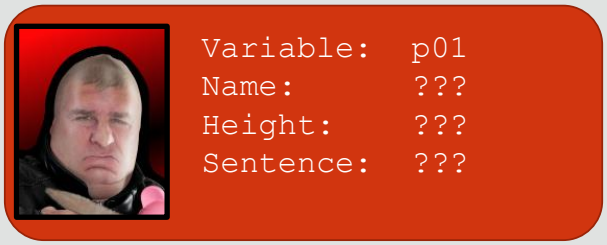
```
public class Prisoner {
   //Campos
   public String name;
   public double height;
   public int sentence;

   //Métodos
   public void think(){
      System.out.println("Terei minha vingança.");
   }//fim do método think
}//fim da classe Prisoner
```

Campos São Variáveis

- As variáveis contêm valores
- Os valores podem ser acessados
- Talvez o código precise acessar variáveis para...
  - Fazer cálculos
  - Verificar valores atuais
  - Alterar um valor
- O que pode acontecer se um campo for acessado antes de ser atribuído um valor a ele?

Exercício 1

- Continue editando com o projeto `PrisonTest`
  - Uma versão deste programa é fornecida nos arquivos `PrisonTest_Student_7_3.java` e `Prisoner_Student_7_3.java`
- Verifique o que acontece quando campos são acessados antes de serem atribuídos valores a eles
  - Instancie um Prisioneiro
  - Tente imprimir o valor de cada campo



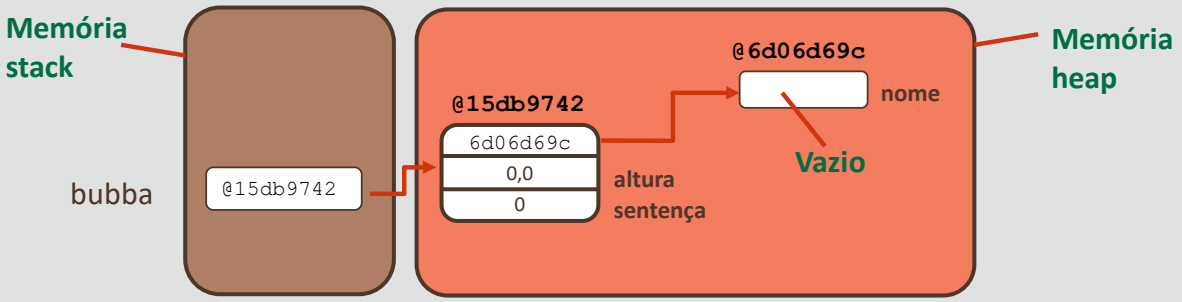
Acessando Campos Não Inicializados

- Se os campos não forem inicializados, eles assumirão um valor padrão
- O Java fornece os seguintes valores padrão:

Tipo de Dados	Valor Padrão
`boolean`	`false`
`int`	0
`double`	0.0
`String`	`null`
Qualquer tipo de objeto	`null`

Referências a Objetos Nulos

- Os objetos podem ter um valor null
- Um objeto nulo aponta para um local vazio na memória
- Se um Objeto tiver outro Objeto como um campo (como uma String), seu valor padrão será null



É Perigoso Acessar Objetos Nulos

- E se um objeto nulo contiver um campo ou um método que precise ser acessado?
  - Isso fará com que o programa trave!
  - O erro específico é NullPointerException

```
public static void main(String[] args){
   String test = null;
   System.out.println(test.length());
}//fim do método main
```

A Importância de Inicializar Campos

- Sempre é bom minimizar as chances de travamento do programa
- E, às vezes, os valores padrão do Java não são desejáveis
- Os demais tópicos desta lição analisarão alternativas úteis para inicializar campos

Definindo Campos de Prisoner

- Atualmente, precisamos de uma linha de código para definir cada campo
- São necessárias quatro linhas para cada objeto Prisoner

```
public class PrisonTest {
   public static void main(String[] args){
      Prisoner p01 = new Prisoner();
      Prisoner p02 = new Prisoner();

      p01.name = "Bubba";
      p01.height = 2.08;
      p01.sentence = 4;
      p02.name = "Twitch";
      p02.height = 1.73;
      p02.sentence = 3;
   }//fim do método main
}//fim da classe PrisonTest
```

Os Métodos Tornam o Código Mais Eficiente

- Se você perceber que está precisando repetir linhas de código idênticas...
  - A programação pode tornar-se uma tarefa monótona
  - Talvez seja possível fazer o mesmo trabalho em menos linhas
  - Tente escrever esse código como parte de um método

```
p01.name = "Bubba";
p01.height = 2.08;
p01.sentence = 4;

p02.name = "Twitch";
p02.height = 1.73;
p02.sentence = 3;
```

Primeira ocorrência

Repetição

Exercício 2

- Continue editando com o projeto `PrisonTest`
- Os campos podem ser definidos de maneira mais eficiente?
  - Adicione um método setFields() à classe Prisoner
  - Esse método deve usar três argumentos, que são usados para definir os valores de cada campo
  - Substitua o código no método main por chamadas a esse método

```
Variável:  p01
Nome:      Bubba
Altura:    6'10"
           (2,08m)
Sentença:  4 anos
```

```
Variável:  p02
Nome:      Twitch
Altura:    5’8”
           (1,73m)
Sentença:  3 anos
```

Escrevendo um Método para Definir Campos

- Sua solução pode ser parecida com esta:

```
public class Prisoner {
   public String name;
   public double height;
   public int sentence;

   public void setFields(String n, double h, int s){
      name = n;
      height = h;
      sentence = s;
   }//fim do método setFields
}//fim da classe Prisoner
```

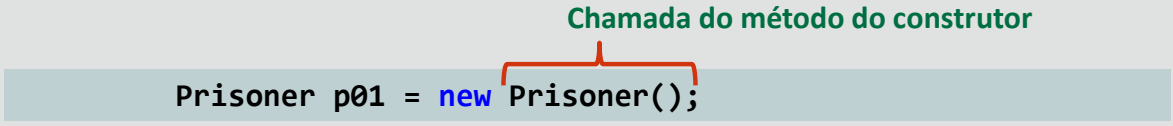
Definindo Campos de Prisoner

- São necessárias quatro linhas para cada objeto Prisoner
- Mas é possível fazer o mesmo trabalho em uma única linha

```
public class PrisonTest {
   public static void main(String[] args){
      Prisoner p01 = new Prisoner();
      Prisoner p02 = new Prisoner();

      p01.setFields("Bubba", 2.08, 4);
      p02.setFields("Twitch", 1.73, 3);

   }//fim do método main
}//fim da classe PrisonTest
```

Chamando um Construtor

- Um construtor é um método especial
- Sua meta é "construir" um objeto definindo os valores iniciais dos campos
- O construtor de um objeto é chamado uma vez
  - Isso ocorre durante a instanciação
  - E nunca mais é chamado novamente
- Estamos chamando construtores todo esse tempo



O Construtor Padrão

- O Java fornece automaticamente um construtor para cada classe
- Ele nunca é escrito explicitamente em uma classe
- Ele denomina-se construtor padrão
- É considerado um construtor com argumento zero


```
Prisoner p01 = new Prisoner();
```

Escrevendo um Método Construtor

- Você pode substituir o construtor padrão por um construtor que você mesmo escreveu
- Os construtores são escritos como qualquer outro objeto, exceto:
  - Eles não têm um tipo de retorno (nem mesmo void)
  - O nome é atribuído a eles de acordo com o nome da classe

```
//Construtor
public Prisoner(){
   System.out.println("Este é um construtor");
}//fim construtor
```

Exercício 3, Parte 1

- Continue editando com o projeto `PrisonTest`
- Copie o construtor na classe `Prisoner`
  - Execute o programa
  - Observe como o código neste método é executado quando objetos Prisoner são instanciados

```
//Construtor
public Prisoner(){
   System.out.println("Este é um construtor");
}//fim construtor
```

Exercício 3, Parte 2

- Como você poderia modificar esse construtor de modo que ele definisse cada campo na classe?
  - Use seu conhecimento sobre métodos para encontrar uma solução
  - Lembre-se de que os construtores são métodos
  - Remova o método setFields()
  - Sua solução deve tornar este método redundante
- Seu IDE reclamará do método main:
  - Como é possível corrigir esses problemas?
  - Execute o programa depois que tiver uma solução

Você Pode Ter Notado...

- Os construtores podem ser escritos de maneira que aceitem argumentos os quais definam valores iniciais do campo
- Quando você cria seu próprio construtor, o construtor padrão não fica mais disponível
- O código torna-se mais útil e requer menos linhas
  - Os próximos slides ilustram essa maior eficiência

```
//Construtor
public Prisoner(String n, double h, int s){
      name = n;
      height = h;
      sentence = s;
}//fim construtor
```

Definindo Campos sem um Construtor

- São necessárias quatro linhas para cada objeto Prisoner

```
public class PrisonTest {
   public static void main(String[] args){
      Prisoner p01 = new Prisoner();
      Prisoner p02 = new Prisoner();

      p01.name = "Bubba";
      p01.height = 2.08;
      p01.sentence = 4;

      p02.name = "Twitch";
      p02.height = 1.73;
      p02.sentence = 3;
   }//fim do método main
}//fim da classe PrisonTest
```


Observação para os Instrutores: as caixas de código nesses três slides devem estar na mesma posição e no mesmo tamanho da fonte para ilustrar a progressão da eficiência.

Definindo Campos com um Método

- São necessárias duas linhas para cada objeto Prisoner

```
public class PrisonTest {

    public static void main(String[] args){
        Prisoner p01 = new Prisoner();
        Prisoner p02 = new Prisoner();

        p01.setFields("Bubba", 2.08, 4);
        p02.setFields("Twitch", 1.73, 3);

 }//fim do método main
}//fim da classe PrisonTest
```


Observação para os Instrutores: as caixas de código nesses três slides devem estar na mesma posição e no mesmo tamanho da fonte para ilustrar a progressão da eficiência.

Definindo Campos com um Construtor

- É necessária uma linha para cada objeto Prisoner

```
public class PrisonTest {

   public static void main(String[] args){
      Prisoner p01 = new Prisoner("Bubba", 2.08, 4);
      Prisoner p02 = new Prisoner("Twitch", 1.73, 3);

   }//fim do método main
}//fim da classe PrisonTest
```


Para Instrutores: as caixas de código nesses três slides devem estar na mesma posição e no mesmo tamanho da fonte para ilustrar a progressão da eficiência.

Atribuindo Nomes a Parâmetros

- Os nomes de variáveis com um único caractere são usados com frequência...
  - Se a variável tiver um escopo muito limitado
  - Se não houver muitas variáveis para serem controladas
  - Para fins de teste
- Mas, no início deste curso, incentivamos a atribuição de nomes descritivos às variáveis
  - Isso ajuda a evitar confusão
  - Siga à risca essa convenção para campos
  - Alguns desenvolvedores gostam de aplicar essa convenção aos parâmetros dos métodos

Atribuindo Nomes a Parâmetros da Mesma Maneira que a Campos

- Essa também é uma prática comum, principalmente com construtores
  - Fica mais claro saber a que os parâmetros estão se referindo
  - Mas isso cria complicações de escopo
- No exemplo a seguir, o campo ou parâmetro nome é impresso?

```
public class Prisoner {
   public String name;

   public setName(String name){
      System.out.println(name);
   }//fim do método setName
}//fim da classe Prisoner
```


Observação para os Instrutores: o código nesses três slides deve estar na mesma posição para que suas alterações fiquem mais aparentes quando os slides forem alterados.

Qual Versão de name É Impressa?

- O parâmetro é impresso
  - As variáveis dentro do escopo mais local têm prioridade
  - Ou seja, as variáveis dentro do escopo mais recente
- O campo continua podendo ser acessado?
  - Sim! Os campos existem dentro do escopo dos métodos de suas classes
  - Mas é necessária uma sintaxe adicional para acessá-los

```
public class Prisoner {
   public String name;

   public setName(String name){
      System.out.println(name);
   }//fim do método setName
}//fim da classe Prisoner
```


Observação para os Instrutores: o código nesses três slides deve estar na mesma posição para que suas alterações fiquem mais aparentes quando os slides forem alterados.

A palavra-chave this

- this é uma referência ao objeto atual
  - Você pode tratá-lo de como qualquer outra referência de objeto
  - O que significa que você pode usar o operador dot (.)
- this.name acessa o campo de Prisoner
- this.setName() acessa o método de Prisoner

```
public class Prisoner {
   public String name;

   public setName(String name){
      System.out.println(name);
   }//fim do método setName
}//fim da classe Prisoner
```


Observação para os Instrutores: o código nesses três slides deve estar na mesma posição para que suas alterações fiquem mais aparentes quando os slides forem alterados.

Exercício 4

- Modifique o construtor `Prisoner`
  - Altere os parâmetros desse método para que o nome de cada parâmetro corresponda ao nome de um campo
  - Defina o valor de cada campo usando a palavra-chave `this`

Resumo dos Construtores

- São métodos especiais em uma classe
- Têm o mesmo nome que a classe
- Não têm um tipo de retorno (nem mesmo void)
- São chamados uma única vez durante a instanciação do objeto
- Podem aceitar argumentos
- São usados para definir valores iniciais dos campos
- Se você não criar seu próprio construtor, o Java fornecerá um construtor padrão com argumento zero

Resumo

- Nesta lição, você deverá ter aprendido a:
  - Entender valores padrão
  - Travar o programa com uma referência null
  - Entender o construtor padrão
  - Escrever um construtor que aceite argumentos
  - Inicializar campos com um construtor
  - Usar this como uma referência a objeto


ORACLE
Academy