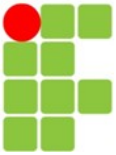


Definições de Classes

Conceitos

- Campos
- Construtores
- Métodos
- Métodos de acesso
- Métodos modificadores
- Definições de classe

Tipos primitivos e tipo de objeto

- Em Java tudo é objeto, exceto alguns valores "primitivos"
  - uma janela é objeto, um botão de uma interface gráfica com o usuário é um objeto, uma conexão com um banco de dados é um objeto, um programa é um objeto, uma palavra é um objeto, ou seja, quase tudo exceto os primitivos.
- Tipos primitivos
  - Inteiros: byte, short, int, long
  - Reais: float, double
  - Caracter: char
  - Lógico: boolean

Tipo	Tamanho (bits)	Valor default
byte	8	0
short	16	0
int	32	0
long	64	0
float	32	0.0
double	64	0.0
char	16	\u0000
boolean	-	false

Exemplos de tipos primitivos e literais

- Literais de caracter
  - char c = 'a';
  - char z = '\u0041'; // em Unicode
- Literais inteiros
  - int i = 10; short s = 15; byte b = 1;
  - long hexa = 0x9af0L; int octal = 0633;
- Literais de ponto-flutuante
  - float f = 123.0f;
  - double d = 12.3;
  - double g = .1e-23;

Exemplos de tipos primitivos e literais

- Literais booleanos
  - boolean v = true;
  - boolean v = false;
- Literais de String (não é tipo primitivo)
  - String s = “abc”;

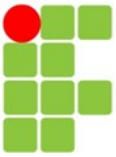
Definição de Classes

```
public class NomeClasse
{
        Campos

        Construtores

        Métodos

}
```



Campos

- Também chamados de **variáveis de instância**

- Declaração

```
tipo nomeCampo [= expressão];
```

- Declaração de Métodos

```
tipoRetorno nomeMétodo( [parâmetros] )
{
    Declaração de variáveis locais
    Corpo do método (lógica)
}
```

Inicialização de objetos

- Construtor da Classe
  - é chamado na criação do objeto **(new)**
  - tem o mesmo nome que a classe

```
class Teste {
   public Teste() { .... }
   public Teste(int i) { ..... }
    .....
}
```

Exemplo

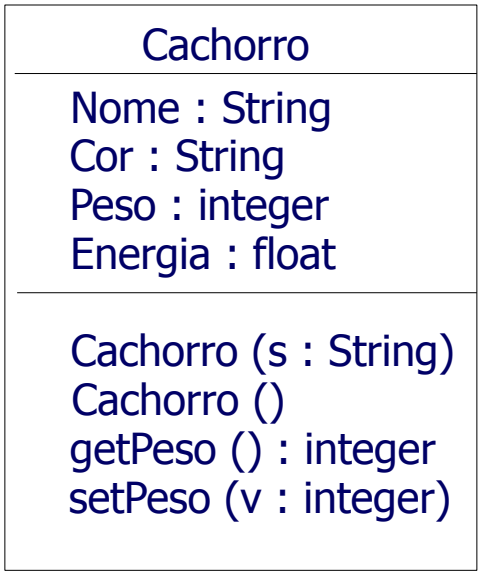
```
class Cachorro {

    // Campos
    private String nome;
    private String cor;
    private int    peso;
    private float  energia;

    // Construtores
    Cachorro(String s) { nome = s; }
    Cachorro() { nome = "Sem nome"; }

    // Métodos (comportamentos dos objetos da classe)
    void setPeso(int v) { peso = v; }
    int getPeso() { return peso; }

}
```



Máquinas de vender bilhetes

- Abra o projeto naive-ticket-machine
- Criar um objeto TicketMachine configurado com o preço do ticket de 500 centavos e examine seus métodos
- Experimente o método getPrice. Você deve ver um valor de retorno contendo o preço de bilhete que foi definido quando esse objeto foi criado
- Use o método insertMoney para simular a inserção de uma quantidade de dinheiro na máquina e então use getBalance para verificar se a máquina tem um registro da quantidade inserida
- Você pode inserir vários montantes distintos de dinheiro na máquina, do mesmo modo que você pode inserir várias moedas ou notas em máquinas reais
- Tente inserir a quantidade exata necessária para um bilhete
- Para emitir um bilhete, chame o método printTicket.
- Qual é o valor retornado se você verificar o balanço da máquina depois de imprimir um bilhete?

Máquinas de vender bilhetes

- Experimente a inserção de diferentes quantidades de dinheiro antes da impressão de bilhetes.
- Você percebeu alguma coisa estranha sobre o comportamento da máquina?
- O que acontece se você inserir muito dinheiro na máquina?
- O que acontece se você não inserir o suficiente e, em seguida, tentar imprimir um bilhete?
- Tente obter uma boa compreensão do comportamento de uma máquina do bilhete de interagindo com ele na bancada de objetos antes de começar a olhar como a classe TicketMachine é implementada
- Crie outra máquina de bilhetes com um preço diferente. Compre um bilhete dessa máquina. O bilhete impresso parece diferente?

Exercícios

- Escreva como você acha que seria os empacotadores externos das classes Student e LabClass - não se preocupe com a parte interna.
- Faça uma lista dos nomes dos campos, construtores e métodos na classe TicketMachine
- Você percebe alguma características do construtor que o torna significativamente diferente dos outros métodos da classe?

Métodos de acesso

- Métodos de acesso retornam informações sobre o estado de um objeto
- Os nomes de métodos de acesso possuem um prefixo get: `getPrice()`
- Terminam com o comando `return valor`

Exercícios

- Compare o método getBalance com o método getPrice. Quais são as diferenças entre eles?
- Se uma chamada para getPrice pode ser caracterizada como: "Qual o custo dos bilhetes?", como você caracterizaria uma chamada para getBalance?
- Definir um método de acesso, getTotal, que retorna o valor do campo total.
- Tente remover a instrução de retorno do corpo de getPrice. Qual mensagem de erro que você vê agora, quando você tentar compilar a classe?
- Compare as assinaturas dos métodos de getPrice e printTicket. Além de seus nomes, qual é a principal diferença entre eles?
- Os métodos insertMoney e printTicket têm instruções de retorno? O que você acha que isso pode ser? Você percebeu alguma coisa sobre os seus cabeçalhos que poderia sugerir porque não requerem instruções de retorno?

Métodos modificadores

- Métodos modificadores alteram o estado de um objeto
- Não possuem o comando `return valor;`
- Podem ter um comando `return;`
- Possuem o tipo de retorno `void`

Exercícios

- Crie uma máquina de bilhetes, com um preço do bilhete de sua escolha.
- Antes de fazer qualquer outra coisa, chame o método getBalance.
- Agora, chame o método insertMoney e forneça-lhe uma quantia positiva como o parâmetro real.
- Agora chame getBalance novamente.
- As duas chamadas para getBalance devem mostrar uma saída diferente porque a chamada para insertMoney teve o efeito de mudar o estado da máquina através do seu saldo de campo.

Imprimindo

```
/**
* Print a ticket and reduce the
* current balance to zero.
*/
public void printTicket()
{
// Simulate the printing of a ticket.
System.out.println("####################");
System.out.println("# The BlueJ Line");
System.out.println("# Ticket");
System.out.println("# " + price + " cents.");
System.out.println("####################");
System.out.println();
// Update the total collected with the balance.
total += balance;
// Clear the balance.
balance = 0;
}
```

Exercícios

- Adicione um método chamado prompt à classe TicketMachine. Isso deve ter um tipo de retorno void e também não deve aceitar parâmetros. O corpo do método deve imprimir algo como: “Por favor insira a quantidade correta do dinheiro”
- Adicione um método showPrice à classe TicketMachine. Ele deve ter um tipo de retorno void e sem parâmetros. O corpo do método deve imprimir algo como: “O preço do bilhete é de centavos xyz”, onde xyz deve ser substituído pelo valor do campo price
- Criar duas máquinas de bilhetes com preços diferentes. Fazer chamadas para seus métodos showPrice mostram a mesma saída ou saídas diferentes? Como você explica esse efeito?
- O que você acha que seria impresso se alterasse a quarta instrução printTicket de modo que price também estivesse entre aspas, como se segue?

```
System.out.println ("#" + "price" + "centavos.");
```

Exercícios

- Modificar o construtor de TicketMachine para que ele não possua nenhum parâmetro. Em vez disso, o preço dos bilhetes deve ser fixado em 1.000 centavos. Qual o efeito que isso tem quando você construir objetos máquina de bilhetes no BlueJ?

- Implementar um método chamado empty, que simula o efeito de remover todo dinheiro da máquina. Este método deve ter um tipo de retorno void, e seu corpo deve simplesmente definir o campo total para zero. Esse método precisa receber algum parâmetro? Teste o seu método, criando uma máquina, inserindo algum dinheiro, imprimindo alguns bilhetes, verificando o total e, em seguida, removendo o dinheiro da máquina. Este método é um modificador?

- Implementar um método setPrice, que é capaz de definir o preço dos bilhetes para um novo valor. O novo preço é passado como um valor de parâmetro para o método. Teste o seu método criando uma máquina, mostrando o preço de bilhetes, alterando o preço, e depois, mostrando o novo preço. É este um método modificador?

- Dê a classe dois construtores. Um deve aceitar um único parâmetro que especifica o preço, e o outro não deve aceitar nenhum parâmetro e definir o preço e configurar um valor padrão de preço a sua escolha. Teste a sua implementação através da criação de máquinas usando os dois construtores diferentes.

Reflexões

- Esta primeira versão da máquina de vender bilhetes contém diversas deficiências
- Você seria capaz de citá-las?

- Faça o exercício que está no endereço:

https://spreadsheets.google.com/a/iftm.edu.br/viewform?hl=pt_BR&formkey=dENpTWJ4VkdOa0x0Qy1WaTMyaTlRTnc6MQ#gid=0

- 1. Construa as seguintes classes:

  - Uma Pessoa tem um nome (String)
  - Uma Porta tem um estado aberto, que pode ser true ou false, e pode ser aberta ou fechada
  - Uma Casa tem um proprietário Pessoa e um endereço
  - Um Ponto tem coordenadas x e y inteiras
  - Um Circulo tem um Ponto e um raio inteiro
  - Um Pixel é um tipo de Ponto que possui uma cor

Exercícios

- 2. Escreva uma classe Ponto

  contém x e y que podem ser definidos em construtor

  métodos getX() e getY() que retornam x e y

  métodos setX(int) e setY(int) que mudam x e y

- 3. Escreva uma classe Circulo, que contenha

  raio inteiro e origem Ponto

  construtor que define origem e raio

  método que retorna a área

  método que retorna a circunferência

  use java.lang.Math.PI (Math.PI)

- 4. Crie um segundo construtor para Circulo que aceite

  um raio do tipo int e coordenadas x e y

Conceitos vistos neste módulo

- Campo
- Comentário
- Construtor
- Escopo
- Tempo de vida
- Método de acesso
- Método modificador
- println
- Variável local