UNIVERSIDADE FEDERAL RURAL DA AMAZÔNIA

Classes e Objetos

Prof. MSc. Tiago Araújo

tiagodavi70@gmail.com

Sumário

Estrutura das classes
- Campos, Métodos e Construtores

Objetos

Passagem de parâmetros para métodos

Modificadores de acesso

Membros de classe (estáticos)

Pacotes

Estrutura de uma classe

Declaração minimalista de uma classe

```
class MyClass {
    // campos
    // construtores
    // métodos
}
```

Estrutura de uma classe

Em geral, a declaração de classes pode conter os seguintes componentes, nesta ordem

- Modificador de acesso (public ou ausente)
- Nome da classe, com a primeira letra em maiúscula por convenção
- O nome da sua classe pai (se houver), precedido pela palavra chave *extends*
- Uma classe só pode herdar de uma classe pai (ou superclasse)
- Uma lista de nomes de interfaces que a classe implementa (se houver), separadas por vírgula, precedida da palavra chave *implements*
- Uma classe pode implementar várias interfaces
- O corpo da classe, cercado por chaves {}

```
class MyClass extends MySuperClass implements MyInterface {
    // fields
    // constructors
    // methods
}
```

Campos

Como vimos anteriormente, variáveis de instância são também chamados de campos (ou atributos)

Cada campo em uma classe é composto de três componentes

- Modificador de acesso (public, private, protected, ausente)
- Tipo do campo
- Nome do campo

Campos

Suponha a classe Bicicleta

```
public class Bicicleta {
     // campos
     public int cadencia;
     public int marcha;
     public int velocidade;

     // construtores
     // métodos
}
```

Os campos definidos são todos do tipo int, mas poderiam ser objetos ou arrays.

Campos

Pela ideia do encapsulamento, é melhor definir os campos como privados e prover métodos de acesso a eles

```
public class Bicicleta {
      // campos
      public int cadencia;
      public int marcha;
      public int velocidade;

      // métodos
      public int getCadencia() {
        return cadencia;
      }
      public void setCadencia(int novoValor) {
         cadencia = novoValor;
      }
      public int getMarcha() {
        return marcha;
      }
      public void setMarcha(int newValue) {
        marcha = newValue;
      }
      public int getVelocidade() {
        return velocidade;
      }
      public void freiar(int decremento) {
        velocidade -= decremento;
      }
      public void acelerar(int incremento) {
        velocidade += incremento;
      }
}
```

Métodos

De forma geral, a declaração de métodos dentro de uma classe possui seis componentes, nesta ordem:

- Modificador de acesso (public, private, ...)
- Tipo de retorno (qualquer tipo) ou void
- Nome do método
  - Assim como para campos e nomes das classes, também há convenções para nomes dos métodos (adiante)
- Lista de parâmetros entre parênteses, separados por vírgula. Cada parâmetro deve ser precedido pelo seu tipo. Se não há parâmetros, deve haver parênteses vazio.
- Lista de exceções que o método pode lançar
- Corpo do método, entre chaves {}

Métodos

Apesar de podermos dar qualquer nome para os métodos, é sensato seguir algumas convenções

A convenção de maiúsculas e minúsculas segue a mesma convenção para nome de variáveis

Primeira palavra minúscula e demais com primeira letra maiúscula

A primeira palavra do nome do método deve ser um verbo. As demais palavras podem ser adjetivos, substantivos, etc.

Exemplos

- getValue()
- compareTo(Object obj)
- isEmpty()

Métodos

A assinatura de um método é determinada pelo nome do método e sua lista de parâmetros

Tipo de retorno e nome dos parâmetros não importa

Importante, pois Java suporta sobrecarga de métodos

Exemplos de assinaturas diferentes de um mesmo método

- draw(int)
- draw(double)
- draw(String, int)

Construtores

Construtores são utilizados para inicializar os objetos da classe

São declarados de forma similar aos métodos

- Contudo, devem ter o nome da classe
- Não possuem tipo de retorno (nem void)

Por exemplo, a classe Bicicleta poderia ter o seguinte construtor

```
public Bicicleta(int startCadence, int startSpeed, int startGear) {
    marcha = startGear;
    cadencia = startCadence;
    velocidade = startSpeed;
}
```

Construtores

Para criar um novo objeto (instância) da classe Bicicleta, devemos usar o operador **new**

O comando `new Bicicleta(30, 0, 8)` aloca a posição de memória necessária ao objeto e inicializa os campos

```
public Bicicleta(int startCadence, int startSpeed, int startGear) {
    marcha = startGear;
    cadencia = startCadence;
    velocidade = startSpeed;
}
```

```
Bicicleta minhaBike = new Bicicleta(30, 0, 8);
```

```
public class Bicicleta {
     // campos
     public int cadencia;
     public int marcha;
     public int velocidade;

     // construtores
     public Bicycle(int startCadence, int startSpeed, int startGear) {
       marcha = startGear;
       cadencia = startCadence;
       velocidade = startSpeed;
     }

     public Bicycle() {
       marcha = 1;
       cadencia = 10;
       velocidade = 0;
     }

     // métodos
}
```

Objetos

Um programa típico em Java cria diversos objetos, de vários tipos

A interação entre os objetos se dá pela chamada dos métodos

- Chamada de métodos caracteriza a troca de mensagens entre os objetos

As interações entre os objetos é responsável pela execução das tarefas do programa

Objetos

A criação de um objeto envolve três passos

- Declaração: associação de um nome de variável a um tipo de objeto
- Instanciação: a palavra-chave new cria uma nova instância (objeto)
- Inicialização: a operação new é seguida de uma chamada a um dos construtores da classe, que inicializa o objeto

Exemplos

```
Point origemUm = new Point(23, 94);
Rectangle rectUm = new Rectangle(origemUm);
Rectangle rectDois = new Rectangle(50, 100);
```

Objetos

A declaração de variáveis primitivas já alocam a quantidade de memória necessária para aquele tipo

Isso não acontece para variáveis do tipo objeto ou arrays

- Variáveis do tipo referência

A simples declaração de um objeto não cria o objeto

Ao tentar usar um objeto não criado, ocorre um erro de compilação

Variáveis que não foram inicializadas são como ponteiros (implícitos) que não referenciam nenhum objeto

```
int x;
Point origemUm;
```

Passagem de Parâmetros para Métodos

Passagem de tipos primitivos para métodos e construtores são sempre por valor

- Cópia dos valores é colocada nos parâmetros

Passagem dos tipos de referência também é feita por valor

- Isso porque as variáveis deste tipo possuem como valor a referência para um objeto/array
- Porém, com a referência ao objeto/array é possível alterar os campos do objeto, caso se tenha acesso suficiente
- Mas ao retornar do método, a referência nunca é perdida

Palavra-chave this

Dentro de métodos de instâncias e construtores, o this representa o objeto (instância) atual

- Não faz sentido para métodos de classe (static)

Uma aplicação muito comum do this é para diferenciar os campos de um objeto dos parâmetros de um método ou construtor

- Como ele, é possível acessar qualquer membro da instância atual, de modo a não gerar ambiguidade

Palavra-chave this

O que aconteceria se o nome dos parâmetros do construtores fossem os mesmos nomes dos campos da classe?

- Com o this, é possível acessar o campo do objeto, mesmo quando uma variável local ou parâmetro obscurece o campo

```
public class Ponto {
    public int x = 0;
    public int y = 0;

    //construtor
    public Ponto(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

```
public class Ponto {
    public int x = 0;
    public int y = 0;

    //construtor
    public Ponto(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

Palavra-chave this

Outra aplicação do this é para chamar um construtor dentro de outro construtor da mesma classe

IMPORTANTE: a chamada de outro construtor com o this deve ser a primeira coisa dentro de um construtor

- Primeira linha

Classes e Objetos cont.

Modificadores de Acesso

Existem quatro tipos de modificadores de acesso

- public: todas as classes tem acesso
- private: apenas a classe atual tem acesso
- protected: classes do seu pacote e subclasses (que podem estar fora do seu pacote) tem acesso
- ausente (package-private): classes do seu pacote tem acesso

Classes podem ser declaradas apenas como public ou package-private

Membros da classe (campos e métodos) podem ter qualquer um dos quatro tipos de acesso

Modificadores de Acesso

Os tipos de acesso são importantes em duas situações para o programador

- Saber quais membros das classes externas ao seu projeto (API Java, por exemplo) suas classes poderão acessar
- Quando escrevemos uma classe, precisamos definir os níveis de acesso para as outras classes

Modificadores de Acesso

Dicas para a escolha do nível de acesso

Use o nível mais restrito possível (private), a menos que você tenha uma boa razão para não fazê-lo

Evite campos públicos, exceto para constantes

- Campos públicos limitam a flexibilidade do código, deixando a implementação mais presa a um contexto
- Quando não permitimos o acesso direto às variáveis, podemos alterar mais facilmente alguma funcionalidade

Uma boa escolha dos níveis de acesso evita erros de mau uso

Membros de classe

Quando um objeto é criado (instanciado), cada um terá seu conjunto de variáveis de instância

- Essas variáveis serão alocadas para cada objeto diferente

As variáveis de classe são comuns a todos os objetos

- Uma única variável na memória, cuja referência é compartilhada por todas as instâncias

Variáveis de classe são declaradas utilizando a palavra chave static

- Campos estáticos

Assim como campos, é possível definir métodos de classe (ou métodos estáticos)

Membros de classe

Constantes

A combinação dos modificadores static e final é usada para criar constantes

Modificador final indica que a variável não pode ser alterada

- Seu valor deve ser definido junto com a declaração
- Tentar alterar uma constante gera erro de compilação

Por que usar modificar static para definir constantes?

- Cada instância teria uma variável constante

Executando programas

Todo projeto em Java precisa de um método público e estático chamado main

Este método deve ser declarado dentro de alguma classe

Contudo, por ser estático, não há necessidade de uma instância para chamá-lo

- No início do programa, não há instâncias de nenhuma classe

```
public class Principal {
    public static void main(String[] args) {
        // O programa começa aqui
    }
}
```

Pacotes

A estrutura de pacotes é importante para organizar classes relacionadas de alguma forma

Lembre-se que os modificadores protected e package-private estão relacionados aos pacotes

Facilita para o programador na hora de buscar por classes que realizam determinada tarefa

Exemplos:

- java.lang
- java.io

Duas (ou mais) classes podem ter o mesmo nome se estiverem em pacotes diferentes

Pacotes

O pacote do qual a classe faz parte precisa ser declarado no início do código-fonte que define a classe

Um pacote também reflete a estrutura de diretórios do código fonte

- Essa estrutura deve ser respeitada, caso contrário o código não compila

```
package veiculo;

public class Bicicleta {
    // corpo da classe
}
```

Tipo *enum*

Tipo especial de dado, que define os possíveis valores associados ao tipo

- Variáveis de um tipo enum só pode assumir os valores pré-definidos
- Por se tratarem de constantes, são declarados em maiúscula
- Valores de enum podem ser usados em switch

```
public enum Dia {
     DOMINGO, SEGUNDA, TERÇA, QUARTA,
     QUINTA, SEXTA, SABADO
}
```

Resumo

Estrutura das classes

Campos, Métodos e Construtores

Objetos

Passagem de parâmetros para métodos

Modificadores de acesso

Membros de classe

Pacotes

Tipo Enum