Associação e Composição de Objetos
Uso e definição de métodos

Programação Orientada a Objetos - Aula 06
Professores: Bonato, Hamilton e Andreia

fonte: Barnes & Kölling, Programação orientada a objetos com Java - Uma introdução prática usando o BlueJ, 4a edição, ed. Prentice Hall

Construtores (revisão)

- Os construtores são blocos especiais usados para instanciar uma classe, “construindo” um novo objeto
- Têm 2 características que o distinguem dos métodos comuns
  - Têm exatamente o mesmo nome da classe
  - Não têm retorno
- Não é obrigatório criar um construtor para uma classe
- Quando criamos devemos usá-lo para atribuir valores iniciais para os atributos da classe, isto é, configurar o estado inicial do objeto

Estado (revisão)

- O estado de um objeto é representado pelos valores dos seus atributos em um determinado momento
- A mudança de um atributo muda o estado
- Classes não têm estado, só objetos, pois somente os objetos podem receber valores

```
public class Aluno
{
    private String nome;
    private int idade;
    private double peso;
    private boolean formando;
    private char sexo;

    public Aluno(String n, int i, double p, char s){
        nome = n;
        idade = i;
        peso = p;
        sexo = s;
        formando = false;
    }
}
```

Exemplo de construtor da classe Aluno. Note que não há o tipo de retorno entre o public e o nome do método. Somente o construtor pode ser assim, nenhum método. Veja que o construtor é usado para inicializar as variáveis com os parâmetros recebidos ou com valores padrão, como o caso do atributo formando. O construtor sempre é public.

Mais Construtores

- Os construtores podem, em vez de atribuir diretamente os valores recebidos como parâmetro às variáveis de instância, chamar os métodos modificadores para fazerem isso.
- Uma classe pode ter múltiplos construtores; para isso basta que se variem os tipos e a quantidade de parâmetros. Isso chama-se sobrecarga.
- Qualquer método pode ser sobrecarregado.

```
public class Aluno
{
    private String nome;
    private int idade;
    private double peso;
    private boolean formando;
    private char sexo;

    public Aluno(String n, int i, double p, char s){
        setNome(n);
        setIdade(i);
        setPeso(p);
        setSexo(s);
        setFormando(false);
    }
    //modificadores
    public void setNome(String n){
        nome = n;
    }
```

```
public void setIdade(int i){
    if(i > 0){
        idade = i;
    }
}
public void setPeso(double p){
    peso = p;
}
public void setFormando(boolean f){
    formando = f;
}
public void setSexo(char s){
    sexo = s;
}
```

Veja esta versão modificada do construtor da classe aluno; desta vez, em vez de fazer nome = n, ele faz setNome(n); e faz o mesmo para todos os atributos; isso é especialmente vantajoso quando há validação de dados, pois concentra-se a regra de validação em um único ponto; note que o parâmetro que configura a idade, por exemplo, está sendo validado para que a idade seja um número positivo.

Instanciação de Objetos

- Para se criar uma nova instância de uma classe chama-se o construtor desta classe com a palavra new

```
Aluno aluno = new Aluno("João da Silva", 19, 72.5, 'M');
```

O operador this

- A palavra this significa "este objeto", "esta instância" e é usada para deixar claro que um determinado atributo ou determinado método que está sendo invocado pertencem a este objeto.
- Seu uso é opcional em alguns casos
- É mais usado quando usamos parâmetros com os mesmos nomes de atributos; neste caso, se não usarmos o this, o Java irá considerar toda referência à variável como sendo ao parâmetro, e não ao atributo; veja o exemplo.

```java
public class Turma
{
    private Aluno aluno;
    private Disciplina disciplina;

    public Turma(){
        aluno = new Aluno("João da Silva", 19, 72.6, 'M');
        disciplina = new Disciplina("LogProg");
    }

    public Turma(Aluno aluno, Disciplina disciplina){
        this.aluno = aluno;
        this.disciplina = disciplina;
    }
}
```

Turma tem dois construtores, um que recebe dois parâmetros e um que não recebe nenhum. Quem usar o construtor Turma() irá criar, como padrão, uma turma de LogProg com um aluno João da Silva. Veja o uso da palavra new. Quem usar o construtor Turma(Aluno, Disciplina) terá que primeiro criar os objetos Aluno e Disciplina para passá-los como parâmetro no construtor.
Note o uso da palavra this. Se fizermos aluno = aluno, o Java não irá atribuir nada ao atributo aluno, mas fará parâmetro aluno = parâmetro aluno; quando uso this.aluno deixo claro que estou me referindo ao atributo aluno. Veja o tema escopo de variáveis no próximo slide.

Escopo de variáveis

- As variáveis podem ser de instância ou locais (parâmetros são variáveis locais)
- A variável de instância existe a partir do momento da instanciação do objeto e pode ser acessada por qualquer método da classe
- As variáveis locais existem a partir de sua criação dentro do método e dentro do local que foram criadas ou dentro do método inteiro se forem parâmetros.

```
public double calculaIMC(double altura){
    double imc = altura/this.peso*this.peso;
    return imc;
}
```

No exemplo, a variável peso é de instância (atributo). Ela vale em todo o objeto. A variável altura é um parâmetro, variável local que vale o método todo. A variável imc é uma variável local que passa a existir somente abaixo da linha em que foi criada.

As variáveis altura e imc desaparecem logo após que o método termina, depois do return. A variável peso continuará existindo.

Quanto à precedência, o Java considera a variável local mais importante que o parâmetro. Por isso quando usamos um atributo com o mesmo nome de uma variável local em um método, a variável local se sobrepõe ao atributo.

Não é possível ter variáveis locais e parâmetros com o mesmo nome.

Definição e chamada de métodos

- Quando você define um método, você tem que colocar o cabeçalho completo:
  - modificador de acesso
  - tipo de retorno
  - nome do método
  - definição dos parâmetros (tipo e nome)
  - implementação do método
- Quando você chama métodos em outro objeto, você usa só a assinatura do método:
  - nome do método
  - passa valores nos parâmetros

```java
public class Aluno
{
    private String nome;
    private int idade;
    private double peso;
    private boolean formando;
    private char sexo;

    public Aluno(String n, int i, double p, char s){
        setNome(n);
        setIdade(i);
        setPeso(p);
        setSexo(s);
        setFormando(false);
    }
    //modificadores
    public void setNome(String n){
        nome = n;
    }
```

```java
public void setIdade(int i){
    if(i > 0){
        idade = i;
    }
}
public void setPeso(double p){
    peso = p;
}
public void setFormando(boolean f){
    formando = f;
}
public void setSexo(char s){
    sexo = s;
}
```

Olhe novamente os métodos set da classe Aluno; são todas definições de método:
public void setNome(String n)
public void setFormando(boolean f)
Veja agora, dentro do construtor, os métodos sendo chamados:
setNome(n) - a variável n sendo passada como parâmetro
setFormando(false) - o valor literal false sendo passado como parâmetro

Objetos falando com objetos

- Primeiro, o que significa um objeto falar com outro?
  - **É um objeto chamar métodos de outro objeto.**
- Para que isso aconteça, é preciso que um objeto conheça o outro, isto é, que um objeto tenha uma variável que aponte para outro objeto, podemos dizer que referencie outro.
- Isso acontece de 4 maneiras:

1. o objeto tem atributos do tipo do objeto que ele quer conversar e instancia este objeto no construtor
2. o objeto tem atributos do tipo do objeto que ele quer conversar e recebe instâncias deste objeto como parâmetro no construtor
3. o objeto não tem atributos do tipo do objeto que ele quer conversar mas recebe uma instância em um método como parâmetro
4. o objeto não tem atributos do tipo do objeto que ele quer nem recebe como parâmetro em um método, mas instancia o objeto em uma variável local

```
public class Turma
{
    private Aluno aluno;
    private Disciplina disciplina;

    public Turma(){
        aluno = new Aluno("João da Silva", 19, 72.6, 'M');
        disciplina = new Disciplina("LogProg");
    }

    public void imprime(){
        String nomeAluno = aluno.getNome();
        int idadeAluno = aluno.getIdade();
        System.out.println("Nome do aluno: " + nomeAluno);
        System.out.println("Idade: " + idadeAluno);
        System.out.println("Nome da disciplina: "+
            disciplina.getNome());
    }
}
```

Este é o primeiro caso. A classe Turma precisa falar com Aluno e com Disciplina. Então ela tem 2 atributos, um do tipo Aluno e outro do tipo Disciplina, e instancia dois objetos em seu construtor, chamando os respectivos construtores de cada classe. Agora ela tem apontadores (referências) para esses objetos e pode chamar métodos dentro do seu método imprime.

Aproveitando, toda vez que você precisar imprimir alguma coisa no console, use o método System.out.println(String). Ele sempre recebe uma String como parâmetro e, tudo o que você concatenar com uma String, usando o +, vira uma String.

```
public class Turma
{
    private Aluno aluno;
    private Disciplina disciplina;

    public Turma(Aluno aluno, Disciplina disciplina){
        this.aluno = aluno;
        this.disciplina = disciplina;
    }

    public void imprime(){
        String nomeAluno = aluno.getNome();
        int idadeAluno = aluno.getIdade();
        System.out.println("Nome do aluno: " + nomeAluno);
        System.out.println("Idade: " + idadeAluno);
        System.out.println("Nome da disciplina: "+
            disciplina.getNome());
    }
}
```

Este é o segundo caso. A classe Turma tem 2 atributos, um do tipo Aluno e outro do tipo Disciplina, mas recebe instâncias dessas duas classes como parâmetro em seu construtor. Isso quer dizer que quem instanciar a classe Turma tem que antes instanciar um objeto Aluno e um objeto Disciplina e depois passá-los como parâmetro no construtor da classe Turma.

Outra coisa: olhando novamente para o método imprimir, você pode criar variáveis locais, atribuir a elas o valor e depois passá-las como parâmetro para o System.out.println ou chamar os métodos diretamente do System.out.println

```
public class Turma
{
    public void imprime(Aluno aluno, Disciplina disciplina){
        String nomeAluno = aluno.getNome();
        int idadeAluno = aluno.getIdade();
        System.out.println("Nome do aluno: " + nomeAluno);
        System.out.println("Idade: " + idadeAluno);
        System.out.println("Nome da disciplina: "+
            disciplina.getNome());
    }
}
```

Este é o terceiro caso. A classe Turma não tem atributos nem do tipo Aluno nem do tipo Disciplina, mas recebe instâncias dessas duas classes como parâmetro em seu método imprime. Quem instanciar a classe Turma tem que antes instanciar um objeto Aluno e um objeto Disciplina e depois passá-los como parâmetro no método imprime.

```
public class Turma
{
    public void imprime(){
        Aluno aluno = new Aluno("João", 19, 72.5, 'M');
        Disciplina disciplina = new Disciplina("LogProg");
        String nomeAluno = aluno.getNome();
        int idadeAluno = aluno.getIdade();
        System.out.println("Nome do aluno: " + nomeAluno);
        System.out.println("Idade: " + idadeAluno);
        System.out.println("Nome da disciplina: "+
            disciplina.getNome());
    }
}
```

Finalmente, este é o quarto caso. A classe Turma não tem atributos nem do tipo Aluno nem do tipo Disciplina e nem recebe instâncias dessas duas classes como parâmetro em seu método imprime, mas ela própria instancia as duas classe em variáveis locais dentro do método imprime.

COMPOSIÇÃO DE OBJETOS

fonte: Barnes & Kölling, Programação orientada a objetos com Java - Uma introdução prática usando o BlueJ, 4a edição, ed. Prentice Hall

Um relógio

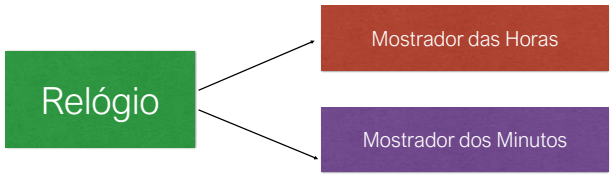
11:03

Abstração

- Usando o poder da abstração, posso reduzir o relógio ao seu mostrador.
- Abstraindo um pouco mais, noto que são na verdade dois mostradores:
  - O das horas, que varia entre 0 e 23.
  - O dos minutos, que varia entre 0 e 59.
- Porém, os dois fazem exatamente a mesma coisa: aumentam de 1 em 1 até chegarem no limite (59 ou 23) e depois zeram novamente.

Modularização

- Posso então desenhar minha solução da seguinte maneira:



  - Atribuindo a cada objeto mostrador a responsabilidade de saber seu valor e de zerar no momento certo.
  - Atribuindo ao objeto Relógio a responsabilidade de aumentar o valor do mostrador de horas cada vez que o mostrador de minutos zera. E também a responsabilidade de unir os valores dos dois mostradores e apresentar o horário para quem perguntar.

Responsabilidades

- A “responsabilidade” de um objeto resume-se ao que ele sabe (seus atributos) e ao que ele faz (seus métodos).
- Não atribua a um objeto uma responsabilidade que não faz sentido que ele tenha.
- Por exemplo, você acha justo que todo aluno seja obrigado a saber as notas de todos os outros alunos em uma determinada disciplina?
- Então não crie um método listarNotasDaTurma em um objeto do tipo Aluno, mas sim em um objeto do tipo Professor.
- Isso torna seus objetos coesos, isto é, coerentes. E este é um conceito muito importante para se construir um bom sistema.

```
1  public class Mostrador{
2
3      //armazena o valor do mostrador
4      private int valor;
5      //armazena o limite do mostrador
6      private int limite;
7
8      public Mostrador(int limite){
9          this.limite = limite;
10         valor = 0;
11     }
12     public int getValor(){
13         return valor;
14     }
15     public void incrementa(){
16         valor = (valor + 1)%limite;
17     }
18     public String mostra(){
19         if(valor<10){
20             return "0"+valor;
21         } else {
22             return ""+valor;
23         }
24     }
25 }
```

```
1  public class Relogio{
2     public Mostrador hora;
3     public Mostrador minuto;
4     public String mostrador;
5
6     public Relogio(){
7        hora = new Mostrador(24);
8        minuto = new Mostrador(60);
9        atualizaMostrador();
10    }
11    public void ticTac(){
12       minuto.incrementa();
13       if(minuto.getValor()==0){
14          hora.incrementa();
15       }
16       atualizaMostrador();
17    }
18    private void atualizaMostrador(){
19       mostrador = hora.mostra()+":"+minuto.mostra();
20    }
21    public String mostra(){
22       return mostrador;
23    }
24 }
```

```
public class Teste{
   public static void main(String[] args){
      Relogio relogio = new Relogio();

      for(int i = 0; i < 1440; i++){
         relogio.ticTac();
         System.out.println(relogio.mostra());
      }
   }
}
```

- Chamada interna de método (o objeto chama um método definido nele mesmo): Relógio, linhas 9 e 16.
- Chamada externa de método (o objeto chama um método de outro objeto): Relógio, linhas 12, 13, 14 e 19.
- Palavra chave this: Mostrador, linha 9
- Construtores: Mostrador, linha 8 a 11, Relógio, linha 6 a 10.

Refatoração

- Refatorar é melhorar o código existente sem criar novas funcionalidades.
- Você deve refatorar quando:
  - Encontra código repetido, que deve ser transformado em um método.
  - Tem um método muito grande, que talvez esteja com muitas responsabilidades e deva ser dividido.
  - Tem uma classe com responsabilidades estranhas, que devem ser colocadas na classe correta.
  - Tem um código muito complexo que possa ser simplificado.

```java
public class Disciplina {
    private String nome;
    private boolean pratica;

    public Disciplina(String nome, boolean pratica) {
        this.nome = nome;
        this.pratica = pratica;
    }
    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
    public boolean isPratica() {
        return pratica;
    }
    public void setPratica(boolean pratica) {
        this.pratica = pratica;
    }
}
```

```java
public class TesteDisciplina {
    public static void main(String[] args) {
        // cria objetos disciplina e professor
        Disciplina disciplina = new Disciplina("ProgComp", true);
        // imprime os dados
        System.out.println("Nome da Disciplina: " + disciplina.getNome());
        System.out.print("Disciplina Pratica: ");
        if (disciplina.isPratica()) {
            System.out.println("sim");
        } else {
            System.out.println("não");
        }
        //altera para nao pratica
        disciplina.setPratica(false);
        //imprime de novo
        System.out.print("Disciplina Pratica: ");
        if (disciplina.isPratica()) {
            System.out.println("sim");
        } else {
            System.out.println("não");
        }
    }
}
```

Note que há código repetido que pode ser eliminado se for criado um método na classe Disciplina que retorne seus dados.

Adicionando-se o método `getDados()` na classe Disciplina...

```
public String getDados() {
    String saida = "Nome da Disciplina: " + nome + "\nDisciplina Pratica: ";
    if (pratica) {
        saida += "sim";
    } else {
        saida += "não";
    }
    return saida;
}
```

O método main fica bem mais simples.

```
public class TesteDisciplina {
    public static void main(String[] args) {
        // cria objetos disciplina e professor
        Disciplina disciplina = new Disciplina("ProgComp", true);
        // imprime os dados
        System.out.println(disciplina.getDados());
        //altera para nao pratica
        disciplina.setPratica(false);
        //imprime de novo
        System.out.println(disciplina.getDados());
    }
}
```