ECM251 – Linguagens de Programação I

Aula 07 – L1/1 e L2/1

Engenharia da Computação – 3ª série

Associação e Composição de Objetos (L1/1 – L2/1)

2024

Horário

Terça-feira: 2 x 2 aulas/semana

- L1/1 (07h40min-09h20min): *Prof. Calvetti*;
- L1/2 (09h30min-11h10min): *Prof. Calvetti*;
- L2/1 (07h40min-09h20min): *Prof. Igor Silveira*;
- L2/2 (11h20min-13h00min): *Prof. Calvetti*.

Associação e Composição de Objetos

Tópico

- Revisão: Estado

Estado

- 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 tem estado, só objetos, pois somente os objetos podem receber valores em seus atributos.

Associação e Composição de Objetos

Tópico

- Revisão: Construtores

Construtores

- Os construtores são métodos especiais, usados para instanciar uma classe, "construindo" um novo objeto;
- O método construtor tem 2 características que o distingue dos outros métodos:
  - Tem exatamente o mesmo nome da classe onde está;
  - Não tem retorno.
- Não é obrigatório criar um construtor para uma classe;
- Quando são criados, devem ser usados para atribuir valores iniciais para os atributos da classe, isto é, configurar o estado inicial do objeto.

Associação e Composição de Objetos

Exemplo

```
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 outro método. Veja que o construtor é usado para inicializar as variáveis com os parâmetros recebidos ou com valores padrão, com o caso do atributo formando. O construtor sempre é public.

Construtores

- Os construtores, em vez de atribuir diretamente os valores recebidos como parâmetro às variáveis de instância, podem chamar os métodos modificadores para fazer isso;
- Uma classe pode ter múltiplos construtores, para isto bastando que variem os seus tipos e as suas quantidade de parâmetros, o que se define como **sobrecarga**;
- Na verdade, não só os construtores, mas qualquer método pode ser sobrecarregado.

Associação e Composição de Objetos

Construtores

```
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.

Associação e Composição de Objetos

Tópico

- Revisão: Instanciação de Objetos

Instanciação de Objetos

- Para se criar uma nova instância de uma classe, deve-se chamar o seu construtor, utilizando o comando ***new***:

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

Tópico

- Encapsulamento – Identificadores de Acesso em Java

Encapsulamento – Identificadores de Acesso em Java

- Restringem o acesso externo aos atributos e métodos de uma classe, e estão ordenados do mais restritivo para o mais aberto:
  - ***private***: atributos e métodos que não são “vistos” fora da classe a que pertencem;
  - ***default***: atributos e métodos que são "vistos" dentro do mesmo pacote em que se localizam;
  - ***protected***: atributos e métodos que são “vistos” fora da classe a que pertencem, porém somente pelas classes da mesma hierarquia e/ou do mesmo pacote;
  - ***public***: atributos e métodos que são “vistos” por todas as classes fora da classe a que pertencem.

Tópico

- O comando ***this***

O comando ***this***

- O comando ***this*** significa “este objeto”, “esta instância” e é usada para deixar claro que um determinado atributo ou determinado método que se está invocando pertence a esse objeto;
- Seu uso é opcional;
- É indicado o quando se utilizam parâmetros com os mesmos nomes dos atributos;
- Neste caso, se não for utilizado o ***this***, o Java irá considerar toda referência à variável como sendo ao parâmetro e não ao atributo;

Associação e Composição de Objetos

Exemplo

```
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.

Associação e Composição de Objetos

Tópico

- Escopo de Variáveis

Escopo de Variáveis

- Uma variável pode ser de instância ou local, onde os 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.

Exemplo

```
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.

Tópico

- Uso de Método e sua definição

Uso de Método e sua definição

- Quando são definidos os métodos, deve-se colocar o cabeçalho completo, composto por:
  - Modificador de acesso;
  - Tipo de retorno;
  - Nome do método;
  - Definição dos parâmetros, composta por tipo e nome;
  - Implementação do método;
  - Quando o método for invocado (chamado) em outro objeto, utiliza-se, somente, a assinatura do método:
  - Nome do método;
  - Valores nos parâmetros, ou passagem de parâmetros.

Uso de Método e sua definição

```
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;
}
```

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

Tópico

- Refatoração

Refatoração

- Refatorar é melhorar o código existente sem criar novas funcionalidades;
- Um código deve ser refatorado 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 subdividido;
  - Tenha uma classe com responsabilidades estranhas, que devem ser colocadas em outra classe mais apropriada;
  - Tem um código muito complexo que possa ser simplificado.

Associação e Composição de Objetos

Exemplo

```
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;
    }
}
```

Associação e Composição de Objetos

Exemplo

```
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");
        }
    }
}
```

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

Associação e Composição de Objetos

Exemplo

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

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

Associação e Composição de Objetos

Exemplo

```
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());
    }
}
```

O método main( ) fica bem mais simples.

Tópico

- Objetos conversando com outros objetos

Objetos conversando com outros objetos

- Primeiramente, o que significa um objeto conversando com outro?

  Quando um objeto chama os 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;

Objetos conversando com outros objetos

- 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 conversar, nem recebe como parâmetro em um método, mas instancia o objeto em uma variável local.

Associação e Composição de Objetos

Exemplo

```
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 a Aluno e com a Disciplina. Então ela tem 2 atributos, um do tipo Aluno e outro do tipo Disciplina, e instancia estas duas classes em seu construtor, chamando os respectivos construtores de cada classe. Agora ela tem apontadores para estes 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.

Associação e Composição de Objetos

Exemplo

```
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 destas 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

Associação e Composição de Objetos

Exemplo

```
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 destas 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.

Associação e Composição de Objetos

Exemplo

```
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 destas duas classes como parâmetro em seu método imprime, mas ela própria instancia as duas classe em variáveis locais dentro do método imprime.

Tópico

- Composição de Objetos

Composição de Objetos

- Primeiramente, o que significa um objeto conversando com outro?

  Quando um objeto chama os 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;

Exemplo

- Um relógio:

Abstração

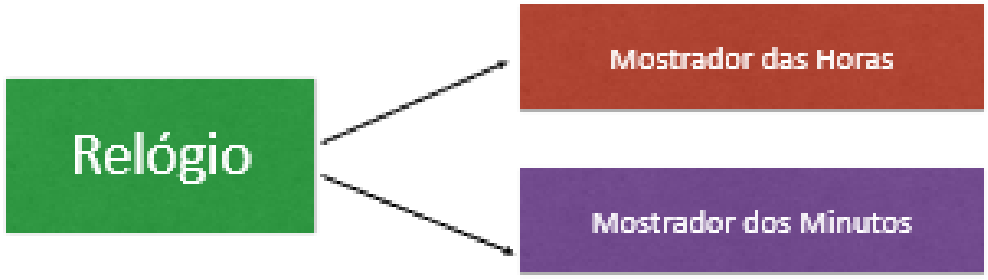
- Usando o poder da abstração, pode-se reduzir o relógio ao seu mostrador;
- Abstraindo um pouco mais, pode-se notar que são, na verdade, dois mostradores:
  - Mostrador das horas, que varia entre 0 e 23;
  - Mostrador dos minutos, que varia entre 0 e 59.
- Porém, os dois fazem exatamente a mesma coisa:
- Incrementam suas contagens de 1 em 1, até chegarem nos limites de cada um, ou seja, 23 para o mostrador de horas e 59 para o mostrador de minutos, na próxima contagem, zerando cada um deles.

Tópico

- Modularização

Modularização

- Pode-se, então, “desenhar” a 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 zeram, além da responsabilidade de unir os valores dos dois mostradores e apresentar o horário para quem perguntar.



Tópico

- Responsabilidades

Responsabilidades

- A “responsabilidade” de um objeto se resume ao que ele “sabe”, por seus atributos e ao que ele “faz”, por seus métodos;
- Não se atribui a um objeto uma responsabilidade que não faça sentido que ele tenha;
- Por exemplo, não seria adequado que qualquer aluno fosse obrigado a saber as notas de todos os outros alunos em uma determinada disciplina;
- Então, não deve ser criado um método ***listarNotasDaTurma( )*** em um objeto do tipo **Aluno**, mas sim em um objeto do tipo **Professor;**
- Isso torna os objetos coesos, isto é, coerentes e este é um conceito muito importante para se construir um bom sistema.

Associação e Composição de Objetos

Exemplo

```
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 }
```

Associação e Composição de Objetos

Exemplo

```
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 }
```

Associação e Composição de Objetos

Exemplo

```
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());
        }
    }
}
```

Exemplo

- Chamada interna de método, ou seja, o objeto chama um método definido nele mesmo: Relógio, linhas 9 e 16;
- Chamada externa de método, ou seja, o objeto chama um método de outro objeto: Relógio, linhas 12, 13, 14 e 18;
- Palavra chave ***this***: Mostrador, linha 9;
- Construtores: Mostrador, linha 8 a 11, Relógio, linha 6 a 10.

Exercício 1

- Executar o código do “Relógio” apresentado anteriormente.

Exercício 2

- Implementar o "Relógio" apresentado anteriormente, alterando seu código de maneira a invocar o método *relogio.ticTac()* toda vez que o usuário digitar a palavra "***tictac***" no teclado.

Exercício 3

- Implementar o "Relógio" apresentado anteriormente, alterando seu código de maneira a invocar o método *relogio.ticTac()* a cada minuto, tempo este controlado pelo relógio interno do computador.

Exercício 4

- Implementar o "Relógio" apresentado anteriormente, alterando seu código de maneira a apresentar em seu mostrador a hora no formato "HH:MM:SS" e invocar o método *relogio.ticTac()* a cada segundo, tempo este controlado pelo relógio interno do computador.

Exercício 5

- Implementar o "Relógio" apresentado anteriormente, alterando seu código de maneira a apresentar em seu mostrador a hora no formato "HH:MM:SS – AM/PM" e invocar o método *relogio.ticTac()* a cada segundo, tempo este controlado pelo relógio interno do computador.

Exercícios Extras

- Propostos pelo professor em aula, utilizando os conceitos abordados neste material...

Bibliografia Básica

- MILETTO, Evandro M.; BERTAGNOLLI, Silvia de Castro. Desenvolvimento de software II: introdução ao desenvolvimento web com HTML, CSS, javascript e PHP (Tekne). Porto Alegre: Bookman, 2014. E-book. Referência Minha Biblioteca: https://integrada.minhabiblioteca.com.br/#/books/9788582601969

- WINDER, Russel; GRAHAM, Roberts. Desenvolvendo Software em Java, 3ª edição. Rio de Janeiro: LTC, 2009. E-book. Referência Minha Biblioteca: https://integrada.minhabiblioteca.com.br/#/books/978-85-216-1994-9

- DEITEL, Paul; DEITEL, Harvey. Java: how to program early objects. Hoboken, N. J: Pearson, c2018. 1234 p. ISBN 9780134743356.

Continua…

Bibliografia Básica (*continuação*)

- HORSTMANN, Cay S; CORNELL, Gary. Core Java. SCHAFRANSKI, Carlos (Trad.), FURMANKIEWICZ, Edson (Trad.). 8. ed. São Paulo: Pearson, 2010. v. 1. 383 p. ISBN 9788576053576.
- LIANG, Y. Daniel. Introduction to Java: programming and data structures comprehensive version. 11. ed. New York: Pearson, c2015. 1210 p. ISBN 9780134670942.
- TURINI, Rodrigo. Desbravando Java e orientação a objetos: um guia para o iniciante da linguagem. São Paulo: Casa do Código, [2017]. 222 p. (Caelum).

Bibliografia Complementar

- HORSTMANN, Cay. Conceitos de Computação com Java. Porto Alegre: Bookman, 2009. E-book. Referência Minha Biblioteca: https://integrada.minhabiblioteca.com.br/#/books/9788577804078

- MACHADO, Rodrigo P.; FRANCO, Márcia H. I.; BERTAGNOLLI, Silvia de Castro. Desenvolvimento de software III: programação de sistemas web orientada a objetos em java (Tekne). Porto Alegre: Bookman, 2016. E-book. Referência Minha Biblioteca: https://integrada.minhabiblioteca.com.br/#/books/9788582603710

- BARRY, Paul. Use a cabeça! Python. Rio de Janeiro: Alta Books, 2012. 458 p.
  ISBN 9788576087434.

Continua...

Bibliografia Complementar (continuação)

- LECHETA, Ricardo R. Web Services RESTful: aprenda a criar Web Services RESTfulem Java na nuvem do Google. São Paulo: Novatec, c2015. 431 p.
  ISBN 9788575224540.

- SILVA, Maurício Samy. JQuery: a biblioteca do programador. 3. ed. rev. e ampl. São Paulo: Novatec, 2014. 544 p.
  ISBN 9788575223871.

- SUMMERFIELD, Mark. Programação em Python 3: uma introdução completa à linguagem Phython. Rio de Janeiro: Alta Books, 2012. 506 p.
  ISBN 9788576083849.

Continua...

Bibliografia Complementar (continuação)

- YING, Bai. Practical database programming with Java. New Jersey: John Wiley & Sons, c2011. 918 p.
- ZAKAS, Nicholas C. The principles of object-oriented JavaScript. San Francisco, CA: No Starch Press, c2014. 97 p. ISBN 9781593275402.

FIM

Engenharia da Computação – 3ª série

Associação e Composição de Objetos (L1/2 – L2/2)

2024

Horário

Terça-feira: 2 x 2 aulas/semana

- L1/1 (07h40min-09h20min): *Prof. Calvetti*;
- L1/2 (09h30min-11h10min): *Prof. Calvetti*;
- L2/1 (07h40min-09h20min): *Prof. Igor Silveira*;
- L2/2 (11h20min-13h00min): *Prof. Calvetti*.

Exemplo

Crie as classes conforme a descrição abaixo:

a. Crie a classe **Turma** com os atributos privados: **codigo**, do tipo *String*, e **ano**, do tipo *int*;

b. Crie um construtor para essa classe que receba parâmetros, para inicializar os atributos, e seus respectivos métodos de acesso e modificadores;

c. Altere a classe **Aluno**, criada na aula passada, para que tenha, também, um atributo privado **turma**, do tipo **Turma**;

Exemplo

Crie as classes conforme a descrição abaixo:

d. Altere o construtor dessa classe para receber um parâmetro, que inicialize o novo atributo, e crie o método de acesso e o modificador para este novo atributo;

e. Crie métodos ***getDados()***, em ambas as classes, que retornam **Strings** com os valores dos atributos;

f. Altere a classe **TesteAluno**, feita na aula passada, para tratar este novo atributo da classe **Aluno**.

Associação e Composição de Objetos

Exemplo

Solução: Classe Turma

```
public class Turma {
    //atributos
    private String codigo;
    private int ano;
    //construtor
    public Turma(String codigo, int ano) {
        this.codigo = codigo;
        this.ano = ano;
    }
    //metodos de acesso
    public String getCodigo() {
        return codigo;
    }
    public int getAno() {
        return ano;
    }
    //metodos modificadores
    public void setCodigo(String codigo) {
        this.codigo = codigo;
    }
    public void setAno(int ano) {
        this.ano = ano;
    }
    //metodo getDados
    public String getDados() {
        return "Turma [codigo=" + codigo + ", ano=" + ano + "]";
    }
}
```

Associação e Composição de Objetos

Exemplo

Solução: Classe Aluno

```
public class Aluno {
    // atributos
    private String nome;
    private int idade;
    private double peso;
    private boolean formando;
    private char sexo;
    private Turma turma;
    //construtor
    public Aluno(String nome, int idade, double peso, char sexo, Turma turma) {
        this.nome = nome;
        this.idade = idade;
        this.peso = peso;
        this.formando = false;
        this.sexo = sexo;
        this.turma = turma;
    }
    //metodos de acesso
    public String getNome() {
        return nome;
    }
    public int getIdade() {
        return idade;
    }
    public double getPeso() {
        return peso;
    }
    public boolean getFormando() {
        return formando;
    }
    public char getSexo() {
        return sexo;
    }
    public Turma getTurma() {
        return turma;
    }
    //metodos modificadores
    public void setNome(String nome) {
        this.nome = nome;
    }
    public void setIdade(int idade) {
        this.idade = idade;
    }
    public void setPeso(double peso) {
        this.peso = peso;
    }
    public void setFormando(boolean formando) {
        this.formando = formando;
    }
    public void setSexo(char sexo) {
        this.sexo = sexo;
    }
    public void setTurma(Turma turma) {
        this.turma = turma;
    }
    //metodo getDados
    public String getDados() {
        return "Aluno [nome=" + nome + ", idade=" + idade + ", peso=" + peso
                + ", formando=" + formando + ", sexo=" + sexo + ", turma="
                + turma.getDados() + "]";
    }
}
```

Associação e Composição de Objetos

Exemplo

Solução: Classe TesteAluno

```
import javax.swing.JOptionPane;
public class TesteAluno {
    // cadastrar um novo aluno no metodo main
    public static void main(String[] args) {
        // coletando os dados do aluno a ser cadastrado
        String nome = JOptionPane.showInputDialog("Nome");
        int idade = Integer.parseInt(JOptionPane.showInputDialog("Idade"));
        double peso = Double.parseDouble(JOptionPane.showInputDialog("Peso"));
        // pega o primeiro caracter da String e retorna como char
        char sexo = JOptionPane.showInputDialog("Sexo M/F").charAt(0);
        String codigo = JOptionPane.showInputDialog("Codigo da Turma");
        int ano = Integer.parseInt(JOptionPane.showInputDialog("Ano da Turma"));
        //cria a turma
        Turma turma = new Turma(codigo, ano);
        // cria um objeto aluno
        Aluno aluno = new Aluno(nome, idade, peso, sexo, turma);
        // nao precisa mais montar a string de saida, e so chamar o metodo getDados
        // mostra o aluno
        JOptionPane.showMessageDialog(null, aluno.getDados());
        // altera informacoes; nao precisa criar todas as variaveis novamente
        idade = Integer.parseInt(JOptionPane.showInputDialog("Idade"));
        peso = Double.parseDouble(JOptionPane.showInputDialog("Peso"));
        // tem que digitar true ou false
        boolean formando = Boolean.parseBoolean(JOptionPane.showInputDialog("E' formando?true/false"));
        // muda usando os metodo modificadores
        aluno.setIdade(idade);
        aluno.setPeso(peso);
        aluno.setFormando(formando);
        // mostra novamente o cadastro do aluno
        // mostra o aluno
        JOptionPane.showMessageDialog(null, aluno.getDados());
    }
}
```

Note que há trechos com código repetido foram substituídos pelos métodos *getDados*. Este é um exemplo de modularização. Outro exemplo é o próprio fato de separar o código em 3 classes diferentes, cada uma com papeis distintos.

Exercícios

1. Crie as classes conforme abaixo:

   a. Crie a classe **Professor** com seu construtor, métodos de acesso e modificadores e os atributos privados **nome**, do tipo *String*, **idade**, do tipo *int*. Crie o método ***getDados()*** que retorna os valores dos atributos;

   b. Crie a classe **Disciplina** com seu construtor, métodos de acesso e modificadores e os atributos privados **nome**, do tipo *String*, **pratica**, do tipo *boolean*. Crie o método ***getDados()*** que retorna os valores dos atributos;

Exercícios

1. Crie as classes conforme abaixo:

   c. Crie a classe **Atribuicao** com seu construtor, métodos de acesso e modificadores e os atributos privados **professor**, do tipo ***Professor***, e **disciplina**, do tipo ***Disciplina***. Crie o método ***getDados()*** que retorna os valores dos atributos;

   d. Crie a classe **TesteAtribuicao** com o método ***main()*** que instancia um **Professor**, uma **Disciplina** e uma **Atribuicao**. Imprima dos dados da **Atribuicao**.

Exercícios

2. Crie as classes **Cliente**, **ContaCorrente** e **Agencia** conforme abaixo:

   a) A classe **Cliente** possui os atributos **nome** e **cpf**, ambos do tipo *String*, e um atributo **conta** do tipo ***ContaCorrente***. Crie um construtor que receba os atributos como parâmetros e os métodos de acesso e os modificadores;

Exercícios

2. Crie as classes **Cliente**, **ContaCorrente** e **Agencia** conforme abaixo:

   b) A classe **ContaCorrente** tem os atributos **numero** e **digito**, ambos *int*, o atributo **agencia** do tipo ***Agencia*** e o atributo **saldo** do tipo *double*. Crie um construtor que receba os atributos como parâmetros e os métodos de acesso e os modificadores. Crie também um método ***depositar()*** que receba um parâmetro *double* com o valor do depósito e aumente o saldo da conta. Crie também um método ***sacar()*** que receba um parâmetro *double* com o valor do saque e diminua o saldo da conta;

Exercícios

2. Crie as classes **Cliente**, **ContaCorrente** e **Agencia** conforme abaixo:

   A conta não pode ficar negativa. Neste caso, deve ser dada uma mensagem que o saque não foi efetuado e o retorno deve ser zero. Caso contrário o retorno deve ser o valor sacado. Crie também um método ***consultarSaldo()*** que não receba parâmetros e retorne o saldo. Crie, finalmente, um método ***imprimirSaldo()*** que imprima o numero da conta corrente com dígito, o número da agência com dígito e o saldo da conta corrente;

Exercícios

2. Crie as classes **Cliente**, **ContaCorrente** e **Agencia** conforme abaixo:

   c) Ainda na classe **ContaCorrente**, o número da conta deve ter no máximo 4 dígitos e ser positivo. O digito da conta deve ser validado a partir do seguinte algoritmo de **módulo 11**: multiplique o primeiro dígito da conta por 4, o segundo por 6, o terceiro por 8 e o quarto por 2; some tudo e calcule o resto da divisão (módulo) da soma por 11. Este é o valor do dígito.

   Obs.: se o resultado for 10 o dígito é 0;

Exercícios

2. Crie as classes **Cliente**, **ContaCorrente** e **Agencia** conforme abaixo:

   d) A classe **Agencia** tem os atributos **nome** do tipo *String*, **numero** e **digito** do tipo *int*. Crie um construtor que receba os atributos como parâmetros e os métodos de acesso e os modificadores. O **numero** e o **digito** da **Agencia** devem seguir os mesmos padrões do número e do dígito da conta corrente;

Exercícios

2. Crie as classes **Cliente**, **ContaCorrente** e **Agencia** conforme abaixo:

   e) Para testar, faça uma classe **CaixaEletronico**, que irá conter o método ***main()***. Nele, instancie um cliente com os seguintes dados:

   Nome: Ademar Apior

   CPF: 123231518-12

   Conta Corrente: 1234 Dígito: 4

   Agencia: 7890 Dígito: 5

   Saldo Inicial: 150.00

Exercícios

2. Crie as classes **Cliente**, **ContaCorrente** e **Agencia** conforme abaixo:

   Operações:

   - sacar 140.0 (sucesso);
   - consultar saldo (resultado é 10.0);
   - sacar 200.0 (falha);
   - consultar saldo (resultado é 10.0);
   - depositar 25.45 (sucesso);
   - imprimir saldo (além dos dados de cliente, conta e agencia, o saldo deve ser 35.45).

Exercícios

3. Criar as classes **CondicionadorDeAr** e **Termostato**.

   a) A classe **CondicionadorDeAr** tem um atributo **termostato** do tipo ***Termostato*** e um atributo *boolean* chamado **ligado**. O construtor de **CondicionadorDeAr** não recebe nenhum parâmetro, mas instancia um termostato e o coloca ligado em ***false***. Crie um método de acesso para **ligado** e outro para **termostato**. Não precisa fazer os métodos modificadores. Crie um método ***ligar()***, que muda **ligado** para ***true***, e um ***desligar()***, que muda **ligado** para ***false***;

Exercícios

3. Criar as classes **CondicionadorDeAr** e **Termostato**.

   Crie um método ***aumentarTemperatura()***, que aumenta a temperatura do termostato em um grau cada vez que é chamado, até o limite de 28 graus. Crie um método ***reduzirTemperatura()***, que reduz a temperatura em um grau cada vez que é chamado, até o limite de 15 graus. Crie um método ***imprimirTemperatura()***, que imprime a temperatura atual. Não se esqueça de verificar se o condicionador está ligado antes de aumentar ou diminuir a temperatura ou imprimi-la;

Exercícios

3. Criar as classes **CondicionadorDeAr** e **Termostato**.

   b) A classe **Termostato** tem um atributo **temperatura**. Seu construtor não recebe parâmetros, mas instancia a temperatura em 20 graus. Crie um método de acesso e outro modificador. Estes métodos devem respeitar os limites estabelecidos no item anterior;

   c) Crie a classe **Usuario**, com o método ***main()***. Neste método você deve instanciar um **CondicionadorDeAr**, aumentar a temperatura para 30 graus (receber mensagem de erro), reduzir a temperatura para 10 graus (receber mensagem de erro). Aumentar a temperatura para 25 graus e imprimir a temperatura.

Atividade

- Individualmente, resolver os exercícios propostos e apresentar à sala, explicando-a, na próxima aula L1/2 e L2/2, a solução daquele solicitado pelo professor.

Bibliografia (apoio)

- LOPES, ANITA. GARCIA, GUTO. Introdução à Programação: 500 algoritmos resolvidos. Rio de Janeiro: Elsevier, 2002.
- DEITEL, P. DEITEL, H. Java: como programar. 8 Ed. São Paulo: Prentice-Hall (Pearson), 2010.

FIM