Análise Orientada a objetos

Aula 9

Prof. Me. Juliana Costa-Silva

Na aula de hoje...

- Classe Time1
- Controle de acesso
- Construtor
- Sobrecarga de Construtor
- Get e Set
- Classes
- Herança
- Reescrita de métodos
- Atividade
- Leitura recomendada

Time1.java

```
1  public class Time1 {
2      private int hora; //0 - 23
3      private int minuto; // 0 - 59
4      private int segundo; // 0 - 59
5
6      public void setTime(int h, int m, int s){
7          hora = ((h >= 0 && h < 24) ? h : 0);
8          minuto = ((m >= 0 && m < 60) ? m : 0);
9          segundo = ((s >= 0 && s < 60) ? s : 0);
10     }
```

Time1.java - Parte II

```
1  public String toUniversalString(){
2      return String.format("%02d:%02d:%02d", hora,
           minuto, segundo);
3  }
4
5  public String toString(){
6      return String.format("%d:%02d:%02d %s",
7       ((hora == 0 || hora == 12) ? 12 : hora % 12),
8       minuto, segundo, (hora < 12 ? "AM":"PM"));
9
10     }
11 }
```

Entendendo...

O que fizemos nesse trecho?

```
1  public class Time1 {
2      private int hora; //0 - 23
3      private int minuto; // 0 - 59
4      private int segundo; // 0 - 59
```

Entendendo...

O que faz esse método?

```
1     public void setTime(int h, int m, int s){
2         hora = ((h >= 0 && h < 24) ? h : 0);
3         minuto = ((m >= 0 && m < 60) ? m : 0);
4         segundo = ((s >= 0 && s < 60) ? s : 0);
5     }
```

Entendendo...

O que não entendemos nos métodos abaixo?

```
1 public String toUniversalString(){
2     return String.format("%02d:%02d:%02d", hora,
          minuto, segundo);
3 }
```

Entendendo...

O que não entendemos nos métodos abaixo?

```
1 public String toUniversalString(){
2     return String.format("%02d:%02d:%02d", hora,
          minuto, segundo);
3 }
```

```
1 public String toString(){
2     return String.format("%d:%02d:%02d %s",
3      ((hora == 0 || hora == 12) ? 12 : hora % 12),
4      minuto, segundo, (hora < 12 ? "AM":"PM"));
5
6     }
```

Anote as explicações, como comentários no código

Testando...

Crie uma classe Time1Test e declare um método main() nessa classe. E faça testes!

```
1  public class Time1Test {
2
3    public static void main(String[] args) {
4      Time1 time = new Time1();
5      System.out.print("Valor universal inicial de time
           : ");
6      System.out.println(time.toUniversalString());
7      System.out.print("Valor default inicial de time:
           ");
8      System.out.println(time.toString() + "\n");
```

Não se esqueça de fechar as chaves da classe e da main()!!

Testando II

```
1       //Altera a hora e gera saidas atualizadas
2       time.setTime(13, 27, 6);
3       System.out.print("Valor universal depois de
            setTime: ");
4       System.out.println(time.toUniversalString());
5       System.out.print("Valor default depois de setTime
            : ");
6       System.out.println(time.toString() + "\n");
```

Controle de acesso

Altere o valor dos atributos na main() da classe TimeTest.java

```
1   }
```

Controle de acesso

Altere o valor dos atributos na main() da classe TimeTest.java

```
1  }
```

Exception in thread “main” java.lang.RuntimeException: Uncompilable source code hora has private access in Time1Test

Controle de acesso

Altere o valor dos atributos na main() da classe TimeTest.java

```
1   }
```

Exception in thread “main” java.lang.RuntimeException: Uncompilable source code hora has private access in Time1Test

Como editar os valores?

Os valores serão editados, somente utilizando o construtor da classe, ou com a chamada dos métodos que fazem a edição, nesse caso método setTime.
Isso se deve ao operador de visibilidade private.

Construtor de classe

E se houver a necessidade de inicializarmos um objeto com um valores, sem utilizar um método para alterar os valores de atributos?

Construtor de classe

Declare um CONSTRUTOR para a classe:

```
1  public class Time2 {
2
3      private int hora; //0 - 23
4      private int minuto; // 0 - 59
5      private int segundo; // 0 - 59
6
7      public Time2(int h, int m, int s) {
8          this.hora = h;
9          this.minuto = m;
10         this.segundo = s;
11     }
```

Construtor de classe

Declare um CONSTRUTOR para a classe:

```
1  public class Time2 {
2
3      private int hora; //0 - 23
4      private int minuto; // 0 - 59
5      private int segundo; // 0 - 59
6
7      public Time2(int h, int m, int s) {
8          this.hora = h;
9          this.minuto = m;
10         this.segundo = s;
11     }
```

Utilizando o CONSTRUTOR da classe Time2:

Construtor de classe

Construtor é o método que instancia um objeto, quando utilizamos *new* String(), estamos utilizando o construtor da classe String.

Construtor de classe

Construtor é o método que instancia um objeto, quando utilizamos *new* String(), estamos utilizando o construtor da classe String.

- Construtores possuem o mesmo nome da classe;

Construtor de classe

Construtor é o método que instancia um objeto, quando utilizamos *new* String(), estamos utilizando o construtor da classe String.

- Construtores possuem o mesmo nome da classe;
- É indicado que toda classe possua ao menos um construtor;

Construtor de classe

Construtor é o método que instancia um objeto, quando utilizamos *new* String(), estamos utilizando o construtor da classe String.

- Construtores possuem o mesmo nome da classe;
- É indicado que toda classe possua ao menos um construtor;
- O construtor não precisa necessariamente inicializar todos os atributos da classe.

Construtor de classe

Construtor é o método que instancia um objeto, quando utilizamos *new* String(), estamos utilizando o construtor da classe String.

- Construtores possuem o mesmo nome da classe;
- É indicado que toda classe possua ao menos um construtor;
- O construtor não precisa necessariamente inicializar todos os atributos da classe.
- Uma classe pode ter vários Construtores.

Construtor de classe

Vejamos como criar vários Construtores em uma classe

```
1      public Time2(int h) {
2          this.hora = h;
3      }
4
5      public Time2(int h, int m) {
6          this.hora = h;
7          this.minuto = m;
8      }
```

Construtor de classe

Vejamos como utilizar vários Construtores em uma classe

```
1
2  public class Time2Test {
3      public static void main(String[] args) {
4          String hora = JOptionPane.showInputDialog("
               Digite a horario 1");
```

Manipulando atributos

E se precisarmos alterar apenas o valor do atributo hora, sem editar os outros valores?

Manipulando atributos

E se precisarmos alterar apenas o valor do atributo hora, sem editar os outros valores?

```
1       // Declare os metodos na classe Time2
2       public int getHora() {
3           return hora;
4       }
5
6       public void setHora(int h) {
7           hora = ((h >= 0 && h < 24) ? h : 0);
8       }
```

Para manipular valores de atributos, a classe deve possuir métodos get e set para cada um de seus atributos.

Manipulando atributos

E se precisarmos alterar apenas o valor do atributo hora, sem editar os outros valores?

```
1       // Declare os metodos na classe Time2
2       public int getHora() {
3           return hora;
4       }
5
6       public void setHora(int h) {
7           hora = ((h >= 0 && h < 24) ? h : 0);
8       }
```

Para manipular valores de atributos, a classe deve possuir métodos get e set para cada um de seus atributos.

Podemos receber valores de hora sem validar?

Desenvolva os métodos get e set da classe Time2, com validação.

Conta.java

E se quisermos colocar um valor padrão (*default*) para nossos atributos?

```
1 class Conta {
2     int numero = 1234;
3     String dono = "Duke";
4     String cpf = "123.456.789-10";
5     double saldo = 1000;
6     double limite = 1000;
7 }
```

Conta.java

E se quisermos colocar um valor padrão (*default*) para nossos atributos?

```
1 class Conta {
2     int numero = 1234;
3     String dono = "Duke";
4     String cpf = "123.456.789-10";
5     double saldo = 1000;
6     double limite = 1000;
7 }
```

Nesse caso os atributos de um objeto são populados com os valores definidos na declaração dos atributos, e ao criarmos um objeto ele já esta populado.

Cliente.java e Conta.java

Um atributo também pode ser uma referência para outra classe. Vejamos a classe cliente:

```
1 class Cliente {
2     String nome;
3     String sobrenome;
4     String cpf;
5 }
```

```
1   public class Conta {
2       int numero;
3       double saldo;
4       double limite;
5       Cliente titular;
6   }
```

Teste.java

```
1 class Teste {
2     public static void main(String[] args) {
3         Conta minhaConta = new Conta();
4         Cliente c = new Cliente();
5         minhaConta.titular = c;
6         // ...
7     }
8 }
```

Teste.java

```
1 class Teste {
2     public static void main(String[] args) {
3         Conta minhaConta = new Conta();
4         Cliente c = new Cliente();
5         minhaConta.titular = c;
6         // ...
7     }
8 }
```

Aqui, simplesmente houve uma atribuição.

Teste.java

```
1 class Teste {
2     public static void main(String[] args) {
3         Conta minhaConta = new Conta();
4         Cliente c = new Cliente();
5         minhaConta.titular = c;
6         // ...
7     }
8 }
```

Aqui, simplesmente houve uma atribuição.
O valor da variável c é copiado para o atributo titular do objeto ao qual minhaConta se refere.

Teste.java

```
1 class Teste {
2     public static void main(String[] args) {
3         Conta minhaConta = new Conta();
4         Cliente c = new Cliente();
5         minhaConta.titular = c;
6         // ...
7     }
8 }
```

Aqui, simplesmente houve uma atribuição.
O valor da variável c é copiado para o atributo titular do objeto ao qual minhaConta se refere.
Em outras palavras, minhaConta agora tem uma referência ao mesmo Cliente que c se refere, e pode ser acessado através de minhaConta.titular.

Funcionario.java

Vejamos a classe que registra os funcionários do banco.

```
1  public class Funcionario {
2
3      String nome;
4      String cpf;
5      double salario;
```

Funcionario.java

Vejamos a classe que registra os funcionários do banco.

```
1  public class Funcionario {
2
3      String nome;
4      String cpf;
5      double salario;
```

Crie os métodos get e set para cada atributo da classe.

Gerente.java

Vejamos a classe que registra os Gerentes do banco.

```
1  public class Gerente {
2      String nome;
3      String cpf;
4      double salario;
5      int senha;
6
7      public boolean autentica(int senha) {
8          if (this.senha == senha) {
9              System.out.println("Acesso Permitido!");
10             return true;
11         } else {
12             System.out.println("Acesso Negado!");
13             return false;
14         }
15     }
16 }
```

Gerente.java

Vejamos a classe que registra os Gerentes do banco.

```
1  public class Gerente {
2      String nome;
3      String cpf;
4      double salario;
5      int senha;
6
7      public boolean autentica(int senha) {
8          if (this.senha == senha) {
9              System.out.println("Acesso Permitido!");
10             return true;
11         } else {
12             System.out.println("Acesso Negado!");
13             return false;
14         }
15     }
16 }
```

Pensando...

- E se tivéssemos mais algum tipo de funcionário?

Pensando...

- E se tivéssemos mais algum tipo de funcionário?
- Se for necessário guardar mais alguma informação sobre todos os funcionários? Teríamos que editar todas as classes (Funcionario, Gerente, Tipo3...)

Pensando...

- E se tivéssemos mais algum tipo de funcionário?
- Se for necessário guardar mais alguma informação sobre todos os funcionários? Teríamos que editar todas as classes (Funcionario, Gerente, Tipo3...)
- Existe uma maneira de relacionarmos uma classe de tal maneira que uma delas **herda** tudo que a outra tem.

Pensando...

- E se tivéssemos mais algum tipo de funcionário?
- Se for necessário guardar mais alguma informação sobre todos os funcionários? Teríamos que editar todas as classes (Funcionario, Gerente, Tipo3...)
- Existe uma maneira de relacionarmos uma classe de tal maneira que uma delas **herda** tudo que a outra tem.
- Isto é uma relação de classe mãe e classe filha.

Pensando...

- E se tivéssemos mais algum tipo de funcionário?
- Se for necessário guardar mais alguma informação sobre todos os funcionários? Teríamos que editar todas as classes (Funcionario, Gerente, Tipo3...)
- Existe uma maneira de relacionarmos uma classe de tal maneira que uma delas **herda** tudo que a outra tem.
- Isto é uma relação de classe mãe e classe filha.
- No nosso caso, gostaríamos de fazer com que o Gerente tivesse tudo que um Funcionario tem, gostaríamos que ela fosse uma extensão de Funcionario.

Pensando...

- E se tivéssemos mais algum tipo de funcionário?
- Se for necessário guardar mais alguma informação sobre todos os funcionários? Teríamos que editar todas as classes (Funcionario, Gerente, Tipo3...)
- Existe uma maneira de relacionarmos uma classe de tal maneira que uma delas **herda** tudo que a outra tem.
- Isto é uma relação de classe mãe e classe filha.
- No nosso caso, gostaríamos de fazer com que o Gerente tivesse tudo que um Funcionario tem, gostaríamos que ela fosse uma extensão de Funcionario.
- Fazemos isto através da palavra chave *extends*.

Herança

```
1  package noExtend;
2
3  public class Gerente {
4      String nome;
5      String cpf;
6      double salario;
7      int senha;
8
9      public boolean autentica(int senha) {
10         if (this.senha == senha) {
11             System.out.println("Acesso Permitido!");
12             return true;
13         } else {
```

Utilizando a nova classe Gerente

```
1
2  class TestaGerente {
3
4      public static void main(String[] args) {
5          Gerente gerente = new Gerente();
6          gerente.setNome("Joao da Silva");
7          gerente.setSenha(4231);
```

Utilizando a nova classe Gerente

```
1
2  class TestaGerente {
3
4      public static void main(String[] args) {
5          Gerente gerente = new Gerente();
6          gerente.setNome("Joao da Silva");
7          gerente.setSenha(4231);
```

Super e Sub classe

A nomenclatura mais encontrada é que Funcionario é a superclasse de Gerente, e Gerente é a subclasse de Funcionario. Dizemos também que todo Gerente é um Funcionário.

Funcionario.java

Como controlar a bonificação dos funcionários?

Funcionario.java

Como controlar a bonificação dos funcionários?
Vamos criar o método bonificação:

```
1       public double getBonificacao() {
2           return this.salario * 0.10;
3       }
```

Funcionario.java

Como controlar a bonificação dos funcionários?
Vamos criar o método bonificação:

```
1       public double getBonificacao() {
2           return this.salario * 0.10;
3       }
```

Porém queremos que Gerentes tenham uma bonificação de 15% e os outros funcionários uma bonificação de 10%.

Funcionario.java

Como controlar a bonificação dos funcionários?
Vamos criar o método bonificação:

```
1       public double getBonificacao() {
2           return this.salario * 0.10;
3       }
```

Porém queremos que Gerentes tenham uma bonificação de 15% e os outros funcionários uma bonificação de 10%. **Teste a bonificação de Gerente e Funcionário.**

Reescrita de métodos

Na classe Gerente.java reescreva o método getBonificacao().

```
1        }
2    }
```

Execute o teste novamente...

Atividade

1. Crie uma classe Conta, que possua um saldo, e os métodos para pegar saldo, depositar, e sacar.
2. Crie os métodos get e set para cada atributo.
3. Adicione um método na classe Conta, que atualiza o saldo da conta de acordo com uma taxa percentual fornecida.
4. Crie duas subclasses da classe Conta : ContaCorrente e ContaPoupanca. Ambas terão o método atualiza reescrito: A ContaCorrente deve atualizar-se com o dobro da taxa e a ContaPoupanca deve atualizar-se com o triplo da taxa.
5. Além disso, a ContaCorrente deve reescrever o método deposita, afim de retirar uma taxa bancária de dez centavos de cada depósito.
6. Crie uma classe com método main e instancie essas classes, atualize-as e veja o resultado.

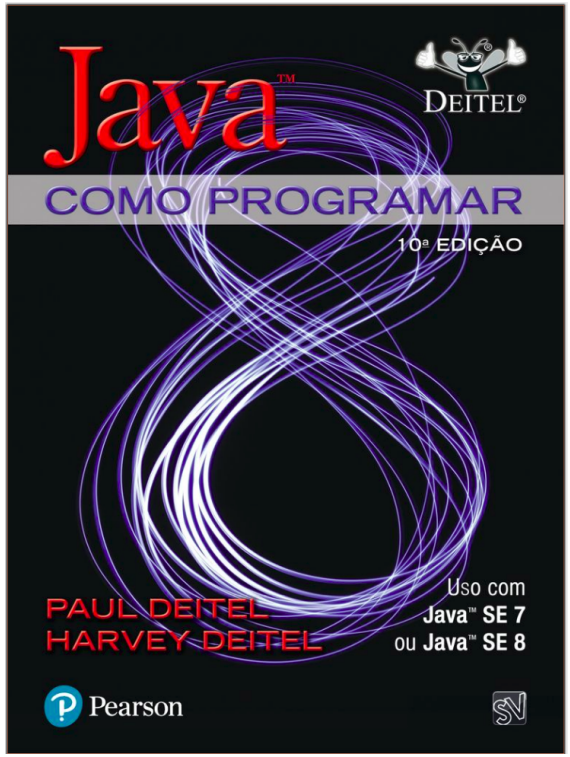
Leitura complementar

Para mais informações sobre JAVA, leia:

Java: Como programar

Capítulo 3:

[Deitel, 2010]



Referências

Deitel, Paul J.; Deitel, H. M. (2010).
Java: Como programar. 8ª Edição.
Pearson.