Programação Orientada a Objetos

Instanciação e Encapsulamento

Prof. Rui César das Neves, Prof. José Cordeiro

Departamento de Sistemas e Informática
Escola Superior de Tecnologia de Setúbal – Instituto Politécnico de Setúbal
2014/2015

Sumário

- Construtores
  - Instanciação e Inicialização de Objetos
  - Palavra reservada “this”
- Encapsulamento
  - Encobrimento de Dados (Data Hiding)
  - Encobrimento do Comportamento Interno

Construtores

- Como já vimos os métodos **construtores** destinam-se a criar objectos (instâncias da classe)…
  - com a possibilidade de atribuir valores iniciais aos atributos.

- Em Java, caso não seja definido nenhum construtor para uma Classe X, é definido automaticamente o construtor X(), sem argumentos, utilizado para criar instâncias da classe.

```
public class Relogio {
    private int _horas;
    private int _minutos;
    private int _segundos;

    public Relogio(){
       _horas = 0;
       _minutos = 0;
       _segundos = 0;
    }
     public Relogio(int horas,
           int minutos,
           int segundos){
       _horas = horas;
       _minutos = minutos;
       _segundos = segundos;
    }

    //[...]

}
```

Construtores - new

- **new** – palavra reservada usada para criar objetos.

```
public class Programa {
    public static void main ( String[] args )
    {
        Relogio timex; // declaração
        Relogio rolex; // declaração
        timex = new Relogio (); // instanciação
        rolex = new Relogio (10, 25, 15); // instanciação

        System.out.println(timex.getHoras() + ":" +
                           timex.getMinutos() + ":" +
                           timex.getSegundos());
         System.out.println(rolex.getHoras() + ":" +
                            rolex.getMinutos() + ":" +
                            rolex.getSegundos());
    }
}
```

Construtores - this

- Note-se que até ao momento temos distinguido os atributos internos do objeto dos parâmetros passados na lista de parâmetros recorrendo a um "_" quando …
- Considerando que tanto o atributo "_horas" como o parâmetro "horas" têm o mesmo valor semântico, e até gostaríamos de o transmitir ao potencial utilizador do método, uma alternativa seria terem exatamente o mesmo identificador …

```
public class Relogio {

    private int _horas;
    Private int _minutos;
    private int _segundos;

    public Relogio(){
        _horas = 0;
        _minutos = 0;
        _segundos = 0;
    }

     public Relogio(int horas,
                    int minutos,
                    int segundos){
        _horas = horas;
        _minutos = minutos;
        _segundos = segundos;
    }

    //[...]

}
```

Construtores - this

- `this` – palavra reservada do Java que refere, dentro de um método da classe X, o objecto da própria classe X para o qual esse método foi chamado.

- Usamos a palavra reservada "this" para nos referirmos aos membros do objeto sempre que possa haver ambiguidade entre identificadores.

```
public class Relogio {

    private int horas;
    private int minutos;
    private int segundos;

    public Relogio(){
        horas = 0;
        minutos = 0;
        segundos = 0;
    }

    public Relogio(int horas,
                   int minutos,
                   int segundos){
        this.horas = horas;
        this.minutos = minutos;
        this.segundos = segundos;
    }

    //[...]

}
```

Construtores - this

- Mas a palavra reservada “this” tem outra poderosa utilização:
  - Quando pretendemos reutilizar um construtor já definido:
- O "**this**" como chamada a outro construtor aparece obrigatoriamente como **primeira instrução do construtor**

```
public class Relogio {

    private int horas;
    private int minutos;
    private int segundos;

    public Relogio(){
        this(0,0,0);
     }

    public Relogio(int horas,
                   int minutos){
        this(horas, minutos, 0);
     }

    public Relogio(int horas,
                   int minutos,
                   int segundos){
        this.horas = horas;
        this.minutos = minutos;
        this.segundos = segundos;
     }

    // […]
}
```

Encobrimento e Encapsulamento

Encobrimento da Informação (Data Hidding)

- Em POO os atributos (variáveis) do objeto estão escondidos do exterior e só são acedidos através dos métodos da classe.

- Mas o conceito de objeto leva mais longe o encobrimento da informação ao permitir definir também métodos internos ao objeto, métodos desconhecidos do exterior, encobrindo assim do exterior o comportamento interno do objeto.

Encapsulamento

- O objeto **encapsula** o estado (atributos/dados) e comportamento (métodos/funções).

Encapsulamento

- **Vantagens**:
  - Evita a propagação de erros por todo o restante programa
    - Por exemplo, alterações ao valor dos atributos só são feitas pelos métodos da própria classe evitando que a atribuição de valores errados possa estar dispersa por acessos provenientes de qualquer outra entidade no código do programa.
  - Evita a propagação por todo o restante programa de erros que surjam na sequência de alterações no código das classes.
    - Nomeadamente evita que alterações nos formatos de dados do objeto possam afetar o restante programa.
    - As alterações no código do objeto podem ser completamente desconhecidas para o exterior.

Encapsulamento - Java

- Em Java do ponto de vista da visibilidade e acessibilidade os membros de uma classe podem ser:

  - Privados (**`private`**) - atributos e métodos internos, i.e., membros vísiveis e acessíveis apenas no interior da classe;

  - Públicos (**`public`**) - métodos disponibilizados para o exterior, i.e. membros acessíveis a objetos de quaisquer outras classes;

  - Outros - 2 (**`protected`** e `*nenhum*`) serão abordados mais à frente quando for tratado o conceito de herança.

Encobrimento e Encapsulamento

- Regra: **Se pode ser** privado, **deve ser privado!**

  - Todos os **atributos** de uma classe **são privados**, i.e. não são sequer vísiveis e muito menos acessíveis a partir do "exterior" do objeto.

```
Pessoa bruno = new Pessoa();
bruno.altura = 1830;                // possível mas ERRADO!
```

  - **Atributos** só são **acedidos/modificados** pelos **inspectores/modificadores**!

```
Pessoa bruno = new Pessoa()         // OK!
bruno.setAltura(1830);              // OK!
```

  - Métodos que não interajam com o exterior são privados:

```
public digerir() { […] ;}           // possível mas ERRADO!
public crescer() { […] ;}           // possível mas ERRADO!
private digerir() { […] ;}          // OK!
private crescer() { […] ;}          // OK
```

Encapsulamento em Acção

- Imagine que necessitamos de ter um relógio com maior precisão e que indique também as decimas e centésimas de segundo. Teremos de alterar o tipo dos segundos de int para double ou float.

```
public class Relogio {
  private int horas; private int minutos; private int segundos;
  public Relogio() { this(0,0,0); }
  public Relogio (int horas, int minutos, int segundos) {
    this.horas=horas; this.minutos=minutos; this.segundos=segundos; }
  public int getHoras () { return horas; }
  public int getMinutos () { return minutos; }
  public int getSegundos (){ return segundos; }
  public void setHoras (int horas) { this.horas = horas; }
  public void setMinutos (int minutos) { this.minutos = minutos; }
  public void setSegundos (int segundos){ this.segundos = segundos; }
  public void avancarMinutos (int maisMinutos) {
    horas += (minutos + maisMinutos) / 60;
    horas = horas % 24;
    minutos = (minutos + maisMinutos) % 60;
  }
}
```

- Vamos introduzir essa “correção” nos dados e ver como a alteração se reflete no programa.

Encapsulamento em Acção

```
public class Relogio {
  private int horas; private int minutos; private double segundos;
  public Relogio() { this(0,0,0.0); }
  public Relogio (int horas, int minutos, double segundos){
    this.horas = horas; this.minutos = minutos; this.segundos = segundos; }
  public int getHoras () { return horas; }
  public int getMinutos () { return minutos; }
  public int getSegundos () { return (int)segundos; }
  public double getSegundosReais () { return segundos; }
  public void setHoras (int horas) { this.horas = horas;}
  public void setMinutos (int minutos) { this.minutos = minutos; }
  public void setSegundos (int segundos) { this.segundos = segundos; }
  public void setSegundos (double segundos) { this.segundos = segundos; }
  public void avancarMinutos (int maisMinutos) {
    horas += (minutos + maisMinutos) / 60;
    horas = horas % 24;
    minutos = (minutos + maisMinutos) % 60;
  }
}
```

- Mas vamos ver como é que a alteração se reflete no "exterior", no programa em sí.

Encapsulamento em Acção

Java - OO

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

    timex.setHoras( 10 );
    timex.setMinutos( 25 );
    timex.setSegundos( 55 );

    System.out.println(timex.getHoras() + ":" +
                       timex.getMinutos() + ":" +
                       timex.getSegundos());

    timex.avancarMinutos( 55 );

    System.out.println(timex.getHoras() + ":" +
                       timex.getMinutos() + ":" +
                       timex.getSegundos());
  }
}
```

Encobrimento e Encapsulamento

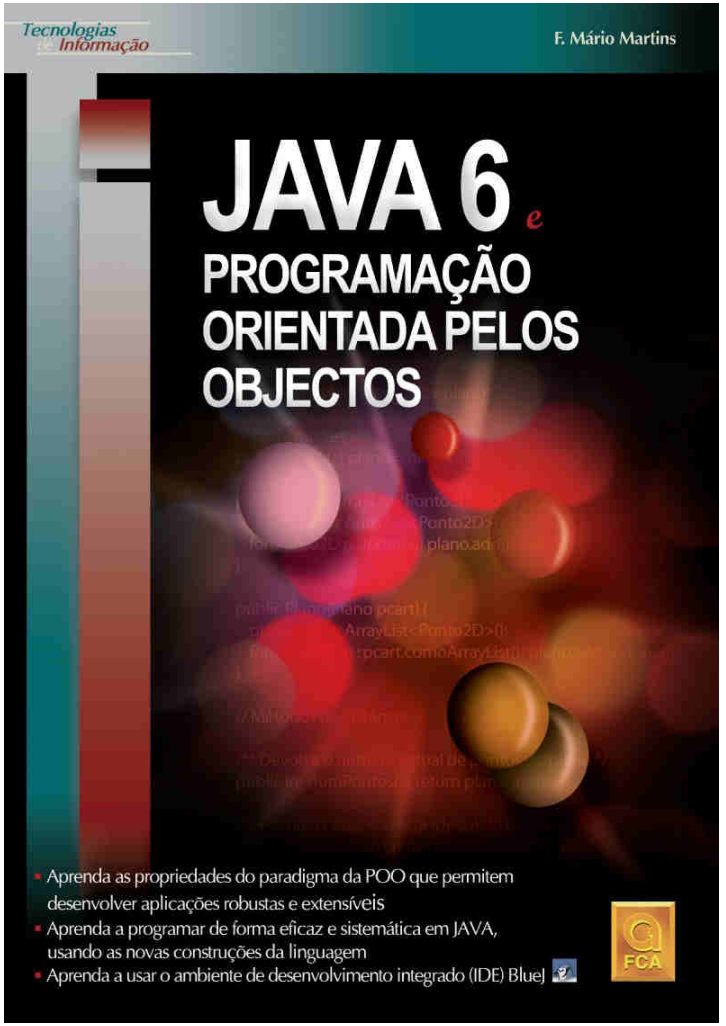
- Analisando os resultados da alteração do tipo de dados:
  - No programa em Java não há alterações fora da classe!
    - As alterações no interior da classe ficaram **encapsuladas** dentro da classe.

Encapsulamento

- Resumindo:
- O Conceito de “Encapsulamento”
  - Está no centro do paradigma OO
  - É ele que assegura:
    - A “Abstracção de dados”
    - A “modularidade”
    - A “independência de contexto”
- Simplificadamente resume-se a …
  - **disponibilizar** ao exterior do objecto apenas **o comportamento mínimo** que assegure o funcionamento e utilidade do objecto.

Leitura Complementar

- Encapsulamento:
  - Página 16
- Capítulo 3
  - Páginas 65 a 112



Exemplo

- Implementar a Classe Relogio mas utilizando apenas um atributo que armazena o número de segundos desde a meia noite.
- Vantagens:
  - Poupança de espaço
  - Facilidade de implementação do método "passarDoTempo" ("tick")
  - Facilidade de comparação de tempos entre dois relógios
- Pedagogicamente
  - Mostra inspectores e modificadores que não são directamente relacionados com os atributos internos mas sim com o conceito que temos no mundo real
  - Mostra que podemos modificar a parte privada sem ter impacto na parte pública
  - Fazer a validação de todos os valores, não permitindo a criação de estados inconsistentes