Encapsulamento: Modularidade, Coesão e Acoplamento

Prof. Pedro Pongelupe

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Departamento de Ciência da Computação

Sumário

Encapsulamento: ocultando informações

O quê é??

Encapsular: "incluir ou proteger em uma cápsula ou como em uma cápsula "

- Ocultar e proteger o estado
- Ocultar o progresso
- Objetiva separar aspectos visíveis de um objeto ou classe de seus detalhes de implementação
- Interface:
  - tudo aquilo que o usuário do objeto vê/acessa.
  - maneira como o usuário utiliza o objeto

Encapsulamento: ocultando informações

- Permite alterar a implementação de um objeto sem impactos em outros módulos do sistema.
- Permite que seus dados sejam protegidos de acesso ilegal.
- Em geral, desejamos ocultar determinados dados e/ou métodos do cliente/usuário da aplicação.

Modificadores de acesso

- Modificadores de acesso controlam a visibilidade dos componentes na aplicação.
- Ao nível da classe: **public** ou *package-private* (sem modificador explícito).
  - Classe declarada como **public** é visível a todas as classes do programa.
  - Classe sem modificador de acesso é visível apenas em seu pacote.
- Ao nível dos membros (atributos e métodos): **public**, **private**, **protected**, ou *package-private* (sem modificador explícito).

Modificadores de acesso

O Java possui 4 modificadores de acesso ao nível dos membros:

- **private**: membros declarados com acesso privado são acessíveis apenas na própria classe.
- *package-private*: membros declarados sem modificador de acesso são acessíveis apenas às classes dentro do mesmo pacote.
- **protected**: membros declarados com acesso protegido são acessíveis às classes do pacote e adicionalmente por suas subclasses.
- **public**: membros declarados com acesso público são acessíveis de qualquer lugar do programa.

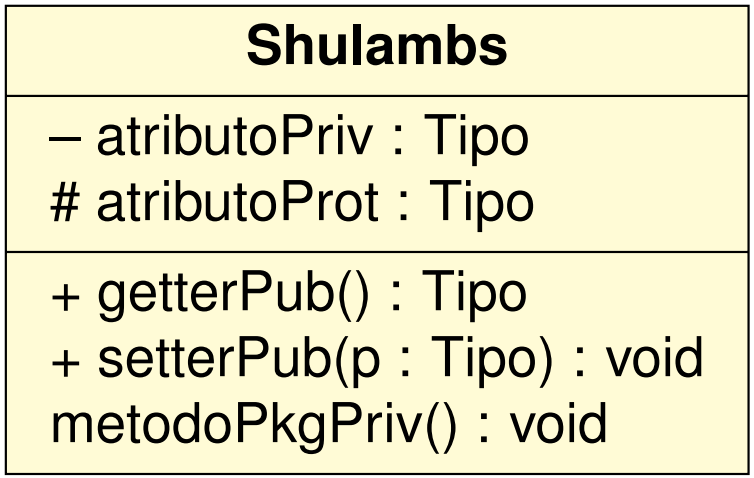
Modificadores de acesso Java

Modifier	Class	Package	Subclass	Global
public	sim	sim	sim	sim
protected	sim	sim	sim	não
<default>	sim	sim	não	não
private	sim	não	não	não

Princípios da ocultação de informação

- Use o nível de acesso mais restrito e que faça sentido para um membro particular.
- É **private**... ou 99% das vezes, mas aquele 1% é...
- Evite campos **public** exceto para constantes.

Encapsulamento na UML




```
class Shulambs {
    private Tipo atributoPriv;
    protected Tipo atributoProt;

    public Tipo getterPub() {
      ...
    }
    public void setterPub(Tipo p) {
      ...
    }

    void metodoPkgPriv() {
      ...
    }
}
```

Classe Aluno: encapsulamento

```
public class Aluno {
        private String nome;
        private String matricula;
        private int idade;
        private double[] notas;
        private double media;

  public Aluno(String nome, String matricula, int idade, double... not
  public Aluno(String nome, String matricula, int idade) {...}
  public Aluno(String nome, int idade) {...}
  public Aluno(String nome) {...}

  public double getMaiorNota() {...}
 }
```

Métodos de acesso (*getters* e *setters*)

- Métodos *get*: acessam o valor de um atributo privado.
  - Valores podem ser tratados antes de serem exibidos.

  Métodos *set*: atribuem um valor a um atributo privado.
  - Valores devem ser validados/tratados antes de serem atribuídos.

Classe Aluno: métodos de acesso (*getters* e *setters*)

```
public class Aluno {
...
public String getNome() { return nome; }
public void setNome(String nome) { this.nome = nome;}

public String getMatricula() { return matricula;}
public void setMatricula(String matricula) { this.matricula = matricula;}

public int getIdade() { return idade;}
public void setIdade(int idade) { this.idade = idade;}

public double[] getNotas() { return notas;}
public void setNotas(double[] notas) { this.notas = notas;}

public String getMedia() { return media;}
public void setMedia(double media) { this.matricula = matricula;}
...
}
```

Classe Produto: acessando membros encapsulados

```
public class Driver {
        public static void main(String[] args) {
                int n = 3;
                Aluno[] alunos = new Aluno[n];
//              Aluno aluno0 = new Aluno();
//              aluno0.nome = "Pedro";
//              aluno0.matricula = "577028";
//              aluno0.idade = 25;
//              alunos[0] = aluno0;

                alunos[0] = new Aluno("Pedro", "577028", 25);
                alunos[0].setNotas(new double[] {5, 6.8, 9});
                alunos[0].setMedia(10);
                alunos[1] = new Aluno("Aninha", "922955", 28, 10, 9.3, 6);
                alunos[2] = new Aluno("Banana", "579855", 26, 6);

                System.out.println("Alunos registrados: \n");
                for (Aluno aluno : alunos) {
                        System.out.println("""
                                        nome: %s,
                                        maior nota: %f
                                        media: %f
                                        """.formatted(aluno.getNome(),
                                                        aluno.getIdade(),
                                                        aluno.getMedia()));
                }
        }
}
```

Quando não utilizar métodos de acesso

```
public class Aluno {
...
public String getNome() { return nome; }
public void setNome(String nome) { this.nome = nome;}

public String getMatricula() { return matricula;}
public void setMatricula(String matricula) { this.matricula = matricula;}

public int getIdade() { return idade;}
public void setIdade(int idade) { this.idade = idade;}

public double[] getNotas() { return notas;}
public void setNotas(double[] notas) { this.notas = notas;}

public String getMedia() { return media;}
public void setMedia(double media) { this.matricula = matricula;}
...
}
```

Quando não utilizar métodos de acesso

```
public class Aluno {
...
public String getNome() { return nome; }
public void setNome(String nome) { this.nome = nome;}

public String getMatricula() { return matricula;}
public void setMatricula(String matricula) { this.matricula = matricula;}

public int getIdade() { return idade;}
public void setIdade(int idade) { this.idade = idade;}

public double[] getNotas() { return notas;}
public void setNotas(double[] notas) { this.notas = notas;}

public String getMedia() { return media;}
// public void setMedia(double media) { this.matricula = matricula;}
...
}
```

Quando não utilizar métodos de acesso

```
class Conta {
    private double limite;
    private double saldo;

    public double getSaldo() {
        return saldo;
    }

    public void setSaldo(double saldo) {
        this.saldo = saldo;
    }

    public double getLimite() {
        return limite;
    }

    public void setLimite(double limite) {
        this.limite = limite;
    }
}
```

```
class Conta {
    private double saldo;
    private double limite;

    public Conta(double limite) {
        this.limite = limite;
    }

    public void depositar(double x) {
        this.saldo += x;
    }

    public void sacar(double x) {
        if(this.saldo + this.limite >= x) {
        this.saldo -= x;
        }
        else throw
          new Exception("Fundos insuficientes.");
        }

    public double getSaldo() {
        return this.saldo;
    }
}
```

Modularidade

"Mecanismo para aumentar a flexibilidade e compreensibilidade de um sistema, em conjunto com a redução do seu tempo de desenvolvimento."
- (Parnas, David L. On the Criteria To Be Used in Decomposing Systems into Modules. Communications of the ACM, Vol. 15, No. 12, pp. 1053 - 1058, 1972. Tradução livre do autor.)

Independência funcional e coesão

- Módulo: “grupo de comandos com uma função bem definida e o mais independente possível em relação ao resto do algoritmo”.
- Cada módulo deve cuidar de uma função específica, servindo a um propósito específico.
- É necessária coerência e unidade conceitual.

Independência funcional e coesão

- **Coesão**: Qualidade de uma coisa em que todas as partes estão ligadas umas às outras. Em software, todas as partes estão coerentemente relacionadas.
- Objetivo de um módulo em programação modular: **alta coesão interna**.
  - Facilita a manutenção.
  - Reduz efeitos colaterais e propagação de erros.
  - Dependência deve ser intra-modular: uso de estruturas internas ao módulo.

Independência modular e acoplamento

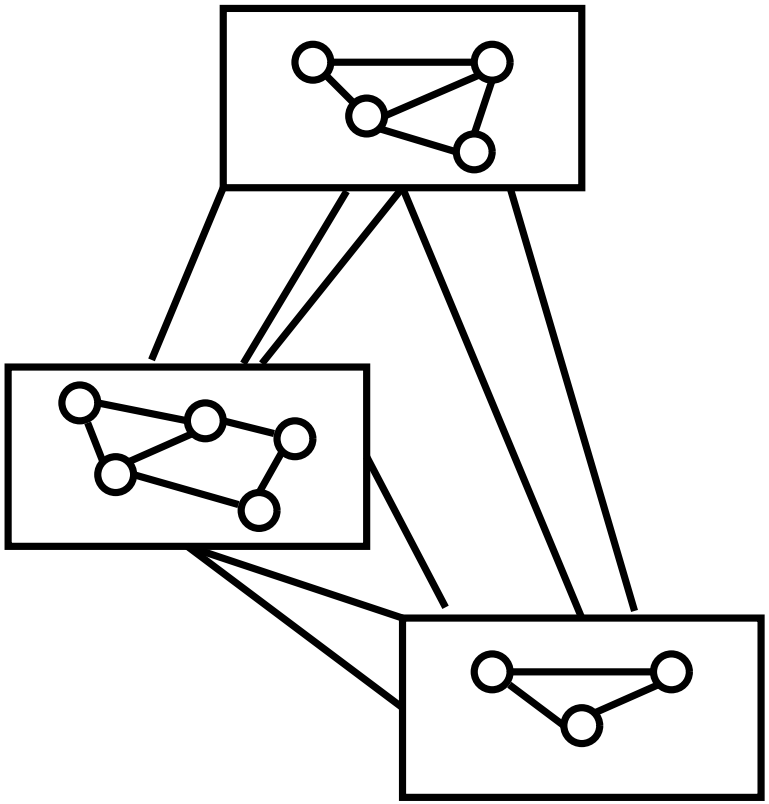
- Módulo: “grupo de comandos com uma função bem definida e o mais independente possível em relação ao resto do algoritmo”.
- A dependência pode ser medida pela quantidade de conexões entre os elementos de software.
- **Acoplamento**:
  - Medida da interconexão entre os elementos de software.
  - Situação ideal: baixo acoplamento.

Indicadores de baixo acoplamento

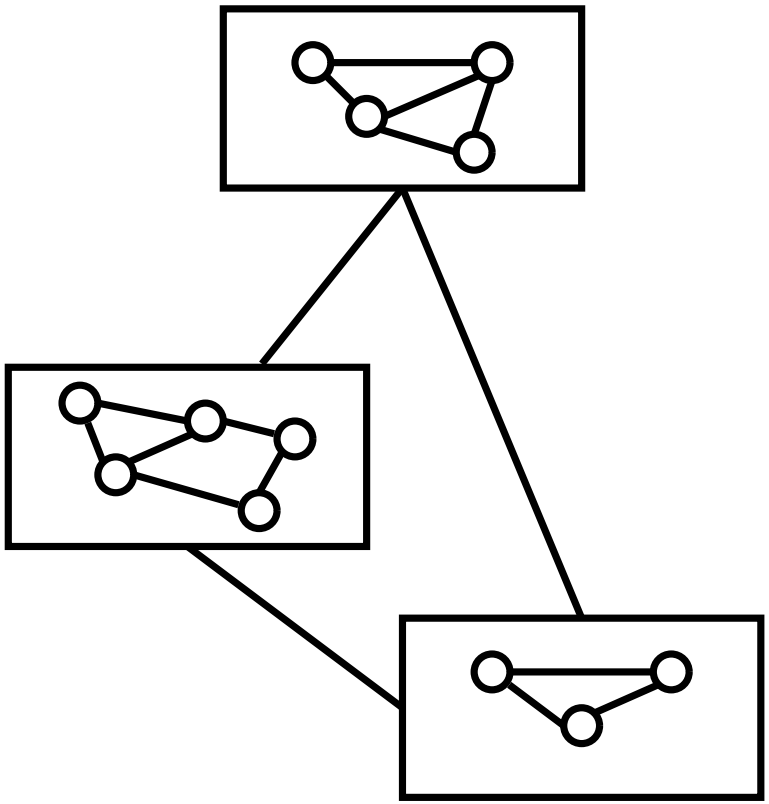
- **Tamanho**: quantidade de parâmetros e métodos públicos.
- **Visibilidade**: uso de parâmetros x uso de variáveis globais.

Coesão e acoplamento

Alto acoplamento



Baixo acoplamento



Bad ending: Código Espaguete

E então, o quê queremos?

Máxima para desenvolvimento de sistemas modulares

Alta Coesão e baixo Acoplamento

Exercício

Classe Hora

Projete uma classe *Hora* para ser utilizada em relógios de diversos sistemas. A hora será representada até o nível de segundos.

- Um objeto Hora **só** pode armazenar estados válidos
- Um objeto Hora **só** pode receber incrementos de horas, minutos ou segundos.
- Um objeto Hora pode ser comparado com outro para verificação de qual valor está mais adiante.

Obrigado!!

Muito obrigado pela atenção! Alguma dúvida? Bora praticar!!!

"*Tenha fé, porque até no lixão nasce flor.*"