SCC0504 - Programação Orientada a Objetos

Pacotes, Cópias e Clones, Associação

Prof.: Leonardo Tórtoro Pereira

leonardop@usp.br

Pacotes

Pacotes [1]

- Um pacote (*package*) é um conjunto de nomes (*namespace*) que organiza um conjunto de classes e interfaces relacionadas
- São semelhantes a pastas no seu computador
- Como programas em Java podem usar centenas a milhares de classes, faz sentido organizá-las em pacotes de acordo com a funcionalidade

Pacotes [2]

- Por quê criar pacotes?
  - Determinar facilmente que tipos estão relacionados
  - Saber onde encontrar tipos que realizem funções relacionadas a um tema
  - Os nomes dos tipos não irão conflitar com nomes em outros pacotes
  - Tipos podem ter acesso irrestrito a outros dentro do mesmo pacote

Pacotes [2]

- Obs:
  - A palavra tipo pode ser usada para se referir genericamente a classes, interfaces, enumerações e anotações*
  - *Anotações são uma forma de metadados que fornecem dados sobre um programa que não é o programa em si

Pacotes [3]

- Para criar um pacote, escolha um nome e coloque "*package NomePacote*" no topo de cada arquivo que contém tipos daquele pacote (deve ser a primeira linha)
- Por convenção, nomes de pacotes devem ter todas as letras minúsculas

Pacotes [4]

- Para usar membros *public* de um pacote existem 3 jeitos
- Usar o nome completo
  - *mypackage.MyClass obj = new mypackage.MyClass();*
- Importar apenas o membro
  - *import mypackage.MyClass;*
- importar todos os membros do pacote
  - *import mypackage.*;*
  - *MyClass obj = new MyClass();*

Pacotes [4]

- Não existe uma hierarquia de pacotes!
  - *java.awt.color* e *java.awt.font* não são parte do pacote *java.awt*. *java.awt* é só um prefixo para agrupar pacotes!
- Importar *java.awt.** importa todos os types no pacote *java.awt*, mas **não importa** os tipos do pacote *java.awt.color*, por exemplo

Pacotes [4]

- Se houver ambiguidade entre pacotes (ex: 2 classes com mesmo nome em pacotes diferentes) é preciso especificar de qual pacote é a classe usada
  - *graphics.Rectangle rect;*

Pacotes [4]

- É possível importar os tipos estáticos de um pacote
  - *import static java.lang.Math.PI*
  - *double r = cos(PI * theta);*

Pacotes [5]

- Java tem vários pacotes prontos. Alguns deles:
  - java.lang (importado automaticamente)
    - Tipos primitivos, operações matemáticas, etc.
  - java.io
    - Operações de entrada e saída
  - java.util
    - Estruturas de dados (listas, dicionários, datas...)

Pacotes [5]

- Java tem vários pacotes prontos. Alguns deles:
  - java.applet
    - Criação de Applets
  - java.awt
    - Componentes para interfaces gráficas
  - java.net
    - Operações de rede

Modificadores de Acesso

	default	private	protected	public
Same Class	Yes	Yes	Yes	Yes
Same package subclass	Yes	No	Yes	Yes
Same package non-subclass	Yes	No	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

Fonte: https://www.geeksforgeeks.org/access-modifiers-java/

Construtor de Cópia

Construtor de Cópia[6]

➔ Função de uma classe que inicializa um objeto usando outro objeto da mesma classe

Construtor de Cópia [7]

```
class Product {
    private int value;
    private String name;
    // A normal parametrized constructor
    public Product(int value, String name) {
        this.value = value;
        this.name = name;
    }
    // copy constructor
    Product(Product p) {
        System.out.println("Copy constructor called");
        value = p.value;
        name = p.name;
    }
}
```

Construtor de Cópia [7]

```
public class MainProduct {
    public static void main(String[] args) {
        Product p1 = new Product(250, "Persona 5 Royal");
        // Construtor de Copia
        Product p2 = new Product(p1);
        // Apenas passa referência!
        Product p3 = p2;
        System.out.println(p2);
    }
}
```

Clone

Clone [8, 9]

- Criar uma cópia exata do objeto
- Cria uma nova instância com os campos inicializados de acordo com os valores passados
- É preciso criar um método público clone
  - O método precisa chamar *super.clone()* para obter a referência do objeto clonado
  - Precisa implementar a interface *java.lang.Cloneable* para não lançar uma exceção

Clone [8, 9]

- Cópia pode ser rasa (*shallow*) ou profunda (*deep*)
  - A rasa copia apenas a referência
  - A profunda faz uma cópia de cada campo de objeto referenciado, criando um novo objeto com os mesmos parâmetros

Cópia Rasa (*shallow*) [8, 9]

```
class Test {
    int x;
}
class Test2 implements Cloneable {
    int a;
    Test c = new Test();
    public Object clone() throws CloneNotSupportedException
    {
        return super.clone();
    }
}
```

Cópia Rasa (*shallow*) [8, 9]

```
public class Main
{
    public static void main(String args[]) throws CloneNotSupportedException
    {
       Test2 t1 = new Test2();
       t1.a = 10;
       t1.c.x = 30;
       Test2 t2 = (Test2)t1.clone();
       t2.a = 100;
       t2.c.x = 300;
       System.out.println(t1.a + " " + t1.c.x);
       System.out.println(t2.a + " " + t2.c.x);
    }
}
```

Cópia Profunda (*deep*) [8, 9]

```
class Test {
    int x;
}
class Test2 implements Cloneable {
    int a;
    Test c = new Test();
    public Object clone() throws CloneNotSupportedException
    {
        Test2 t = (Test2)super.clone();
        t.c = new Test();
        return t;
    }

}
```

Cópia Profunda (*deep*) [8, 9]

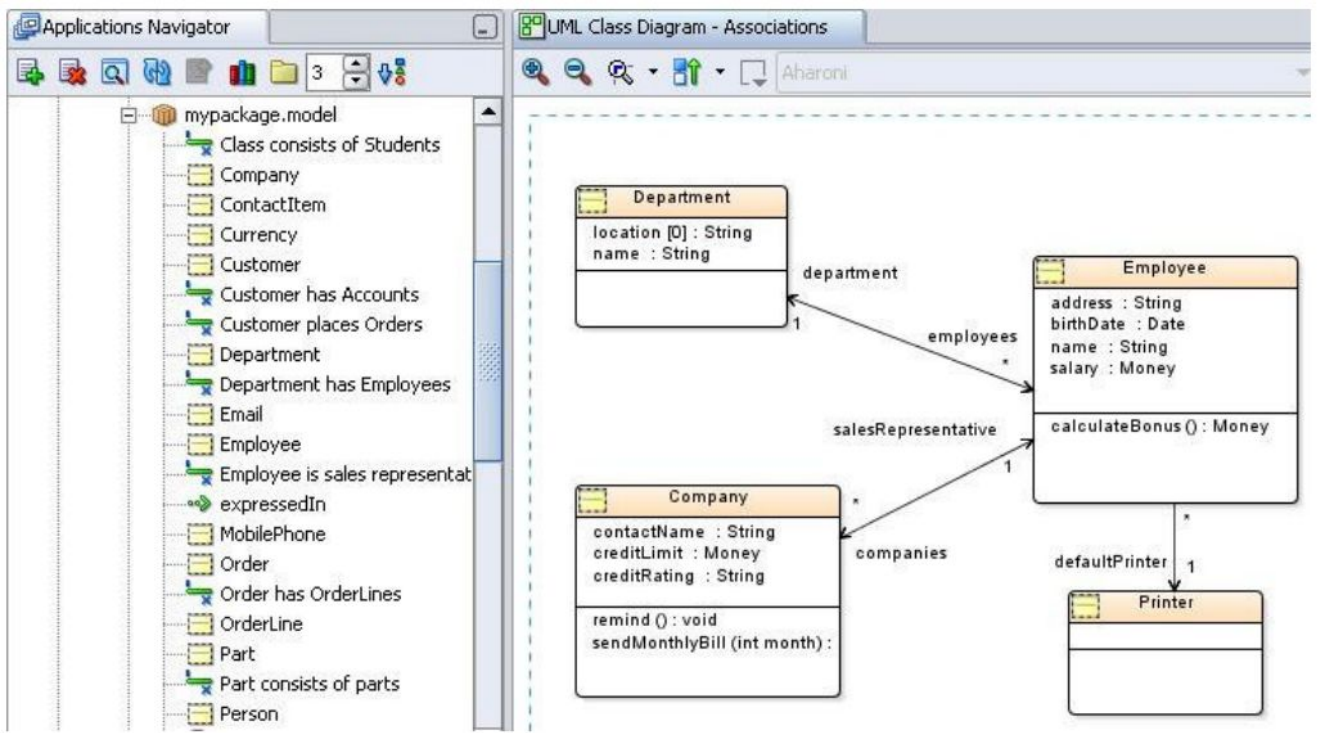
```
public class Main
{
    public static void main(String args[]) throws CloneNotSupportedException
    {
       Test2 t1 = new Test2();
       t1.a = 10;
       t1.c.x = 30;
       Test2 t2 = (Test2)t1.clone();
       t2.a = 100;
       t2.c.x = 300;
       System.out.println(t1.a + " " + t1.c.x);
       System.out.println(t2.a + " " + t2.c.x);
    }
}
```

Associação: Agregação e Composição

Associação

Associação [10]

- Relação estrutural entre classes que especifica que objetos de uma classe são conectados a objetos de outra classe
  - Empregados conectados a um departamento
- Em UML, são linhas sólidas
- A cada fim da linha, a **multiplicidade** da associação é indicada
  - Quantos de uma classe estão associados à outra



Classes and associations in the navigator

Fonte: [10]

Associação [10]

- Multiplicidade:
  - 0..1 Opcional (zero ou um)
  - 1 Requerido (um e apenas um)
  - * ou 0..* Zero ou mais
  - 1..* Um ou mais
- Cada fim de uma associação também tem um nome, que representa o **papel** da associação

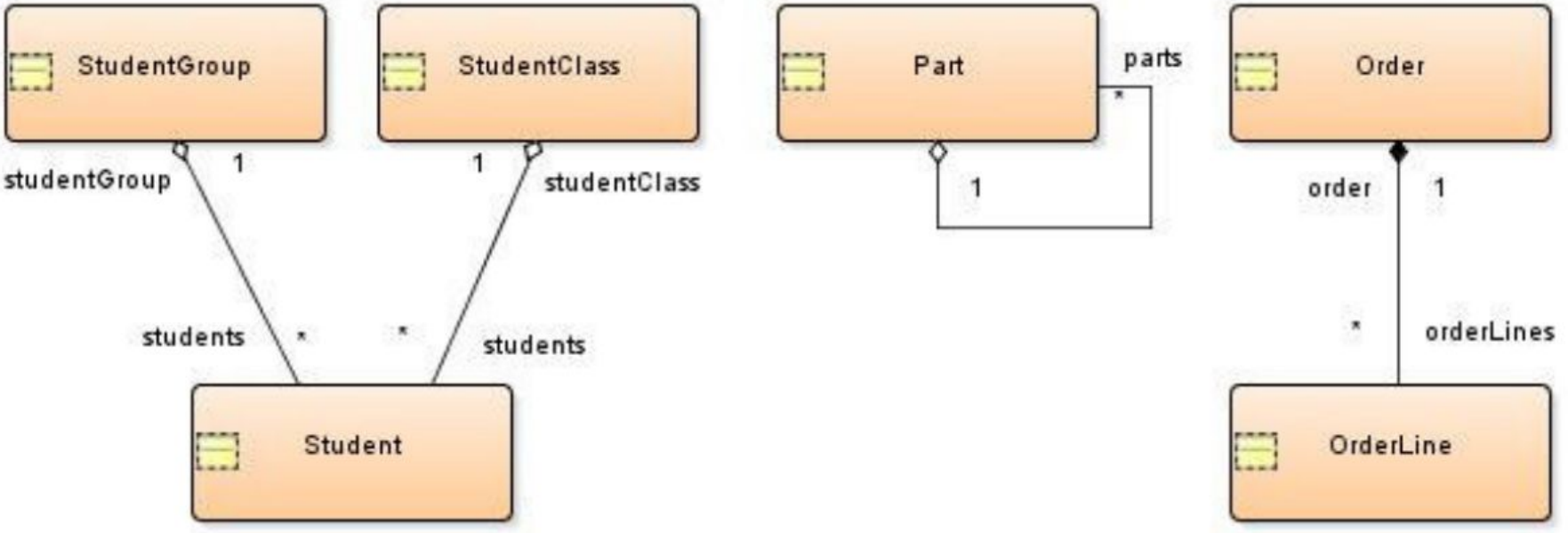
Agregação e Composição

Agregação e Composição [10]

- ➔ Agregação e composição são relacionamentos “inteiros”
- ➔ Na composição, uma parte não pode existir sem o todo, mas na agregação isso é permitido
- ➔ Agregação é chamada de “agregação fraca” e composição de “agregação forte”

Agregação e Composição [10]

- Em UML uma agregação é representada como uma associação com um diamante “aberto” no lado do “todo”
- A composição é representada com um diamante “fechado” no lado do “todo”



Aggregation versus composition

Fonte: [10]

Agregação e Composição [10]

- ➔ Agregação implica que não pode haver relação circular
- ➔ Uma classe pode participar de mais de uma agregação, como um **estudante** que participa de duas agregações: **grupo de estudante** e **sala de estudante**

Agregação e Composição [10]

- O “todo” da composição determina o tempo de vida da “parte”
- No geral, quando deletamos o “todo” fica implícito a deleção das “partes”, a não ser que seja especificado o contrário
- Composição não implica que uma “parte” não possa ser transferida de um “todo” a outro, mas é responsabilidade do “todo” prover a “parte” para outro “todo”
- Uma classe só pode participar de 1 composição

Agregação e Composição [10]

- Um **grupo de estudantes** consiste de **estudantes**
- Um **estudante** não deixa de existir quando um **grupo de estudantes** se encerra.
- Portanto, é uma agregação!

Agregação [11]

- Também pode ser entendido como uma relação “possui um(a)”
- É unidirecional. Um departamento pode ter estudantes, mas não o contrário.
- Ajuda a modelar reuso de código

Agregação e Composição [10]

- ➔ Um **Pedido** consiste de **Linhas de Pedido**
- ➔ Quando o **Pedido** é deletado, as **Linhas do Pedido** também são.
- ➔ Portanto, é uma composição!

Composição [11]

- ➔ Também pode ser entendido como uma relação “parte de algo”
- ➔ Ambas entidades dependem uma da outra

Agregação e Composição [10]

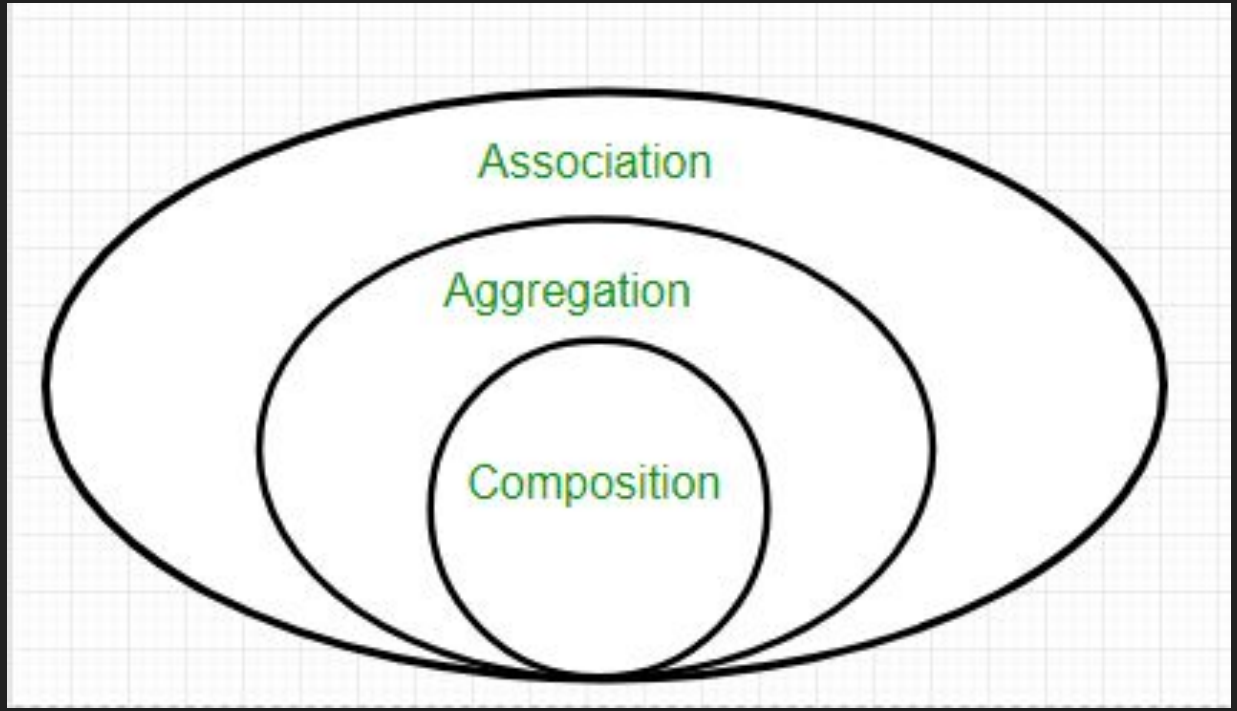
- ➔ Qual usar depende do contexto do Sistema em discussão.
- ➔ Se você quisesse modelar um **Computador** que consiste de **Peças**, entre elas um **HD**
- ➔ Uma organização que vende computadores e peças provavelmente modelaria isso como uma agregação, já que o **HD** pode ser vendido separadamente

Agregação e Composição [10]

- ➔ Mas para uma organização que somente *usa* computadores, pode-se modelar como uma composição uma vez que o fato de que se o **HD** pode ser removido de um computador é totalmente irrelevante para a organização

Agregação e Composição [10]

➔ Alguns autores defendem que agregação só deveria ser usada por modeladores de UML mais experientes



Fonte: [11]

Referências

1. https://docs.oracle.com/javase/tutorial/java/concepts/package.html
2. https://docs.oracle.com/javase/tutorial/java/package/packages.html
3. https://docs.oracle.com/javase/tutorial/java/package/createpkgs.html
4. https://docs.oracle.com/javase/tutorial/java/package/usepkgs.html
5. https://www.geeksforgeeks.org/packages-in-java/
6. https://www.geeksforgeeks.org/copy-constructor-in-cpp/
7. https://www.geeksforgeeks.org/copy-constructor-in-java/
8. https://www.geeksforgeeks.org/clone-method-in-java-2/
9. https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html#clone--

Referências

10. https://www.oracle.com/technetwork/developer-tools/jdev/gettingstartedwithumlclassmodeling-130316.pdf
11. https://www.geeksforgeeks.org/association-composition-aggregation-java/
12. https://www.geeksforgeeks.org/difference-between-association-and-aggregation/