



UPskill – JAVA + .NET

Programação Orientada a Objetos em Java – Agregação e Composição

Adaptado de Donald W. Smith (TechNeTrain)

Relações entre Classes (1)

- Uma classe **depende** de outra se usa objetos dessa classe
 - relação “conhece”
- Visualização de relações: diagramas de classe
- **UML**: Unified Modeling Language
 - Notação para análise e projeto orientado ao objeto

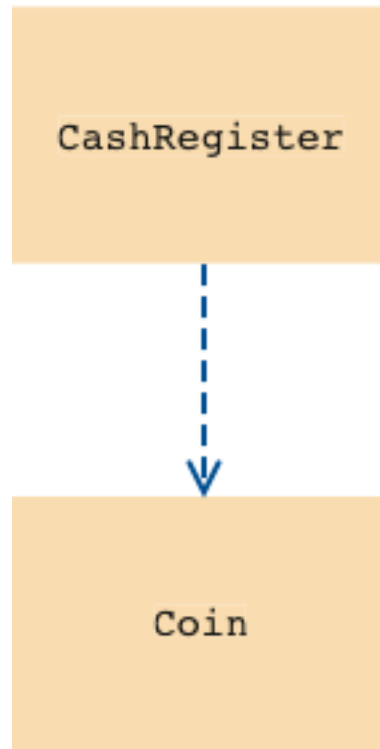
Relações entre Classes (2)

- CashRegister depende de Coin para determinar o valor do pagamento

```
public class Coin
{
    public Coin(double aValue, String aName) { . . . }
    public double getValue() { . . . }
    . . .
}

public class CashRegister
{
    public void enterPayment(int coinCount, Coin coinType)
    { . . . }
    . . .
}
```

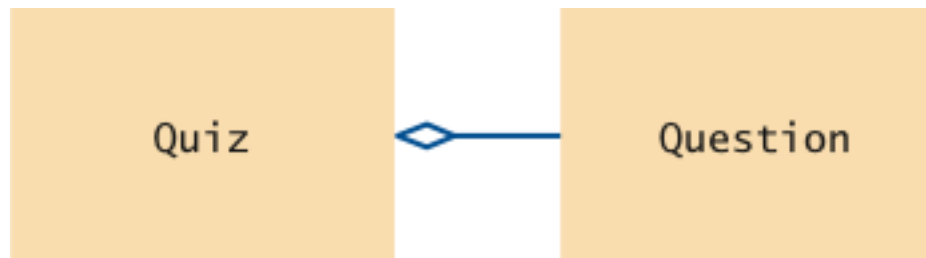
Relação de Dependência



- Na programação orientada ao objeto, um objeto está relacionado com outro para usar funcionalidades e serviços fornecidos por esse objeto
- Esta relação entre dois objetos é conhecida como **associação**
- **Composição** e **Agregação** são formas de **associação** entre dois objetos

Agregação (1)

- Uma classe **agrega** outra se os seus objetos contêm objetos de outra classe
 - relação “has-a”
 - Exemplo: um teste (*quiz*) é formado por questões
 - A classe Quiz agrega a classe Question



Agregação (2)

- A identificação de relações de agregação ajuda na implementação de classes
- Exemplo: uma vez que um teste pode ter qualquer número de questões, usar um ArrayList para os colecionar:

```
public class Quiz
{
    private ArrayList<Question> questions;
    . . .
}
```

Agregação (3)

- Os objetos agregados podem existir sem serem parte de um objeto principal
- Exemplos:
 - Uma questão pode existir sem pertencer a um teste
 - Aluno numa Escola, quando a Escola encerra, o Aluno continua a existir e pode ingressar noutra Escola

Agregação (4)

- Uma vez que *Organization* tem *Person* como *employees*, a relação entre eles é a Agregação

```
public class Organization {  
    private List employees;  
}
```

```
public class Person {  
    private String name;  
}
```

Composição (1)

- Referimo-nos à associação entre dois objetos como composição, quando uma classe possui outra classe e a existência desta outra classe não faz sentido, quando o seu “dono” é destruído
- Relação “part-of”
- Exemplo: a classe Secretaria é uma *composition* de várias partes incluindo pernas, tampo e gaveta
 - Quando o objeto Secretaria é destruído, a existência de todas as partes da secretaria deixam de ter utilidade

Composição (2)



- Outro exemplo de Composição é Car e as suas partes (motor, rodas, ...) – as partes individuais de um carro não funcionam isoladamente quando o carro é destruído

```
public class Car {  
    //final will make sure engine is initialized  
    private final Engine engine;  
  
    public Car(){  
        engine = new Engine();  
    }  
}  
  
class Engine {  
    private String type;  
}
```

- São formas de associação



Agregação e Composição

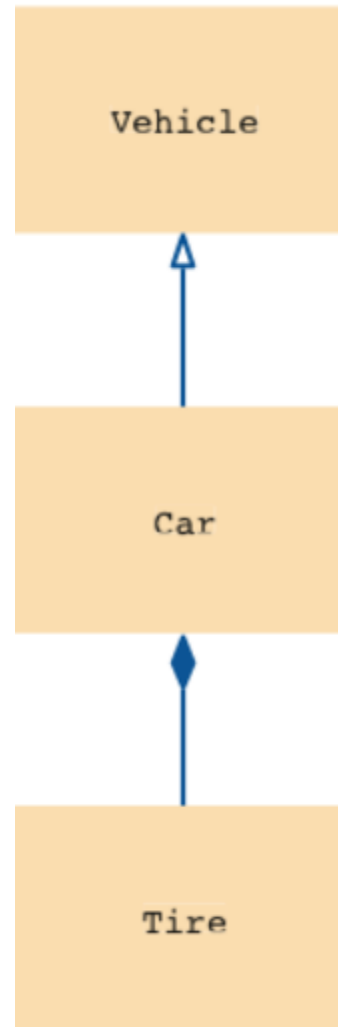
	Agregação	Composição
Geral	Objetos podem existir de forma independente	Objeto que faz parte de outro não pode existir de forma autónoma
Relacionamento	has-a	part-of
Associação	fraca	forte
Representação UML		
Função	A eliminação de um objeto não implica a eliminação do outro	A eliminação de um objeto implica a eliminação de outros que o constituem

Herança (1)

- **Herança** é uma relação entre uma classe mais genérica (**superclass**) e uma classe mais especializada (**subclass**)
 - Relação “is-a”
- Exemplo: qualquer carro *is a* veículo; qualquer carro tem pneus
 - A classe Car é uma subclasse da classe Vehicle; a classe Car agrega a classe Tire

Herança (2)

```
public class Car extends Vehicle
{
    private Tire[] tires;
    . . .
}
```



Composição ou Herança?

- Identifique os componentes do objeto
 - As partes devem ser agregadas ao objeto por composição “*has-a*”
- Tente encontrar no objeto uma semelhança de identidade com classes existentes
 - Deve ser usada herança se verificar uma relação do tipo “*is-a*”
 - Tipicamente deve ser usada herança quando estiver a construir uma família de tipos (relacionados entre si)

UML: Relações entre Objetos

