# Bacharelado em Ciência e Tecnologia

## *BC 1501 – Programação Orientada a Objetos*

## Aula 07

UFABC

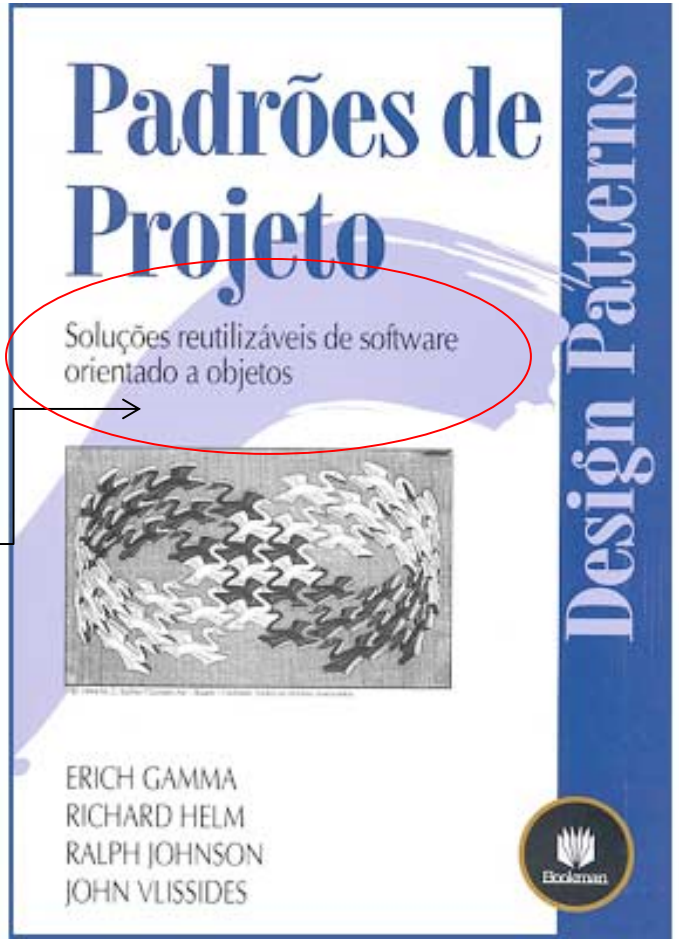
Universidade Federal do ABC

# Nesta Aula

- Introdução aos padrões de projeto de software

# Introdução

- O que são **padrões de projeto** de software?
  - Padrões de Projeto de software podem ser definidos simplesmente como "***soluções reutilizáveis de software orientado a objetos***" ou "*soluções para problemas recorrentes*"



# Introdução

O conceito de padrão de projeto foi criado na década de 70 pelo arquiteto Christopher Alexander, e estabeleceu que um padrão deve ser descrito em cinco partes:

# Introdução

Cinco partes para descrever um Padrão de Software:

1. **Nome**: uma descrição da solução, mais do que do problema ou do contexto.
2. **Exemplo**: uma ou mais figuras, diagramas ou descrições que ilustrem um protótipo de aplicação.
3. **Contexto**: a descrição das situações sob as quais o padrão se aplica.
4. **Problema**: uma descrição das forças e restrições envolvidos e como elas interagem.
5. **Solução**: relacionamentos estáticos e regras dinâmicas descrevendo como construir artefatos de acordo com o padrão, frequentemente citando variações e formas de ajustar a solução segundo as circunstâncias. Inclui referências a outras soluções e o relacionamento com outros padrões de nível mais baixo ou mais alto.

# Introdução

- Quantos padrões de projeto existem?

  Muitos padrões de projetos foram propostos por diversos pesquisadores e profissionais da área de POO.

- Entretanto, os padrões mais usados são os **23 padrões** descritos no livro da "Gangue dos Quatro" (GOF)

# Introdução

- Os 23 padrões GOF são divididos em 3 categorias:
  - Padrões de criação (5)
    - Criação de objetos
  - Padrões estruturais (7)
    - Associações entre classes e objetos
  - Padrões comportamentais (11)
    - Interações e divisões de responsabilidades entre classes e objetos

# Introdução

## Padrões de criação:

1. *Abstract Factory*
2. *Builder*
3. *Factory Method*
4. *Prototype*
5. *Singleton*

Ref: https://pt.wikipedia.org/wiki/Padrão_de_projeto_de_software

# Introdução

## Padrões estruturais

1. *Adapter*
2. *Bridge*
3. *Composite*
4. *Decorator*
5. *Façade* (ou *Facade*)
6. *Flyweight*
7. *Proxy*

Ref: https://pt.wikipedia.org/wiki/Padrão_de_projeto_de_software

# Introdução

**Padrões comportamentais**

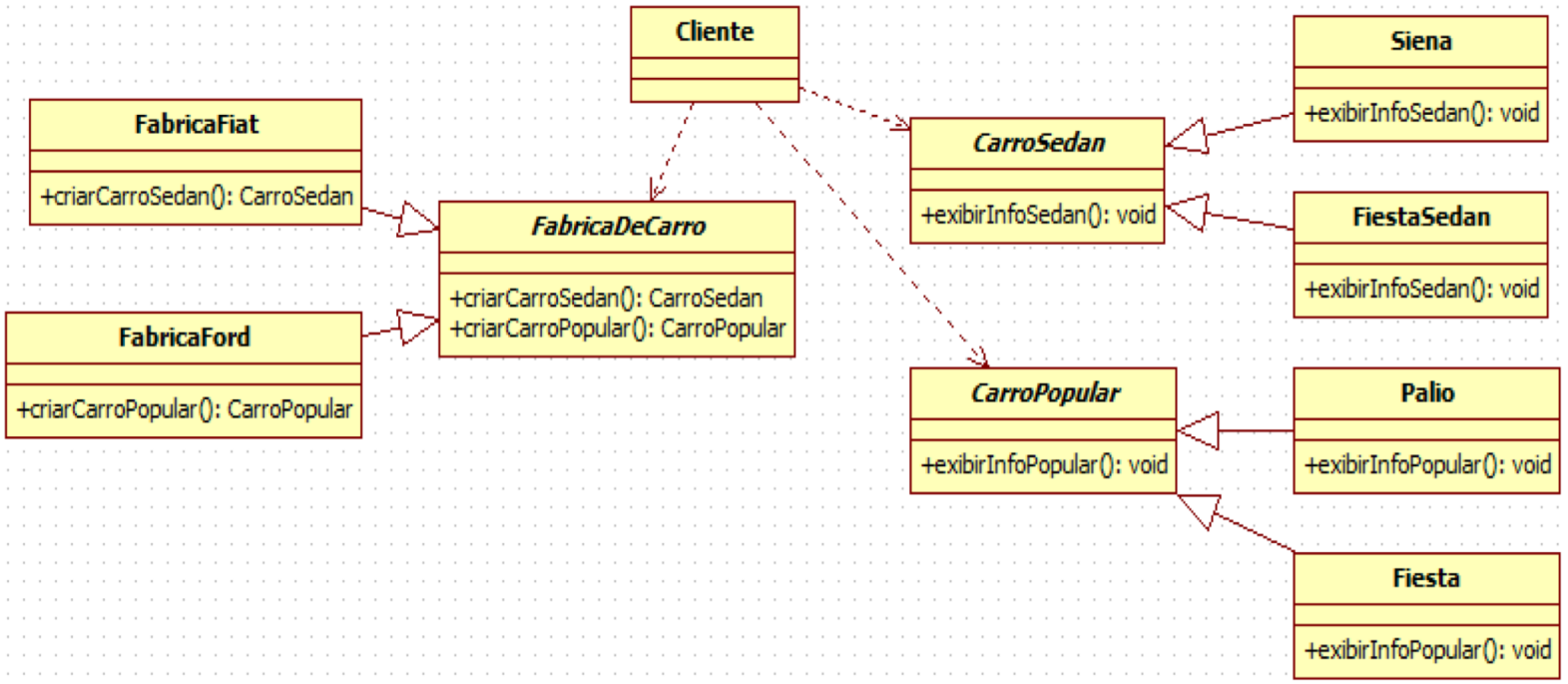
1. *Chain of Responsibility*
2. *Command*
3. *Interpreter*
4. *Iterator*
5. *Mediator*
6. *Memento*
7. *Observer*
8. *State*
9. *Strategy*
10. *Template Method*
11. *Visitor*

Ref: https://pt.wikipedia.org/wiki/Padrão_de_projeto_de_software

# Padrão Abstract Factory

Categoria: Criação

"Fornecer uma interface para criação de famílias de objetos relacionados ou dependentes sem especificar suas classes concretas."




Ref: https://brizeno.wordpress.com/category/padroes-de-projeto/abstract-factory/ - https://youtu.be/k--87KQalBs

# Padrão Singleton

Categoria: Criação

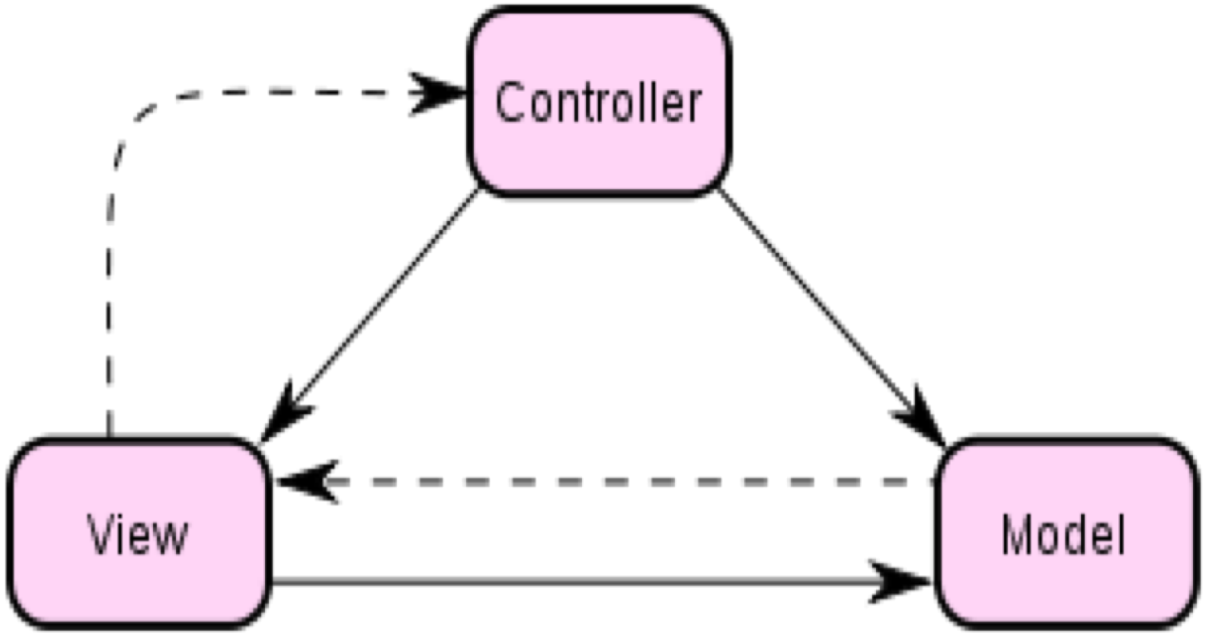
Utilizado quando precisamos garantir que uma classe tenha uma única instância e fornecer um ponto global de acesso à mesma.

- Exemplo:
  - Quando utilizamos objetos DAO, queremos apenas um único objeto para cada classe DAO

***DAO*** *(acrônimo de* ***Data Access Object****), é um padrão para persistência de dados que permite separar regras de negócio das regras de acesso a banco de dados. Numa aplicação que utilize a arquitetura MVC, todas as funcionalidades de bancos de dados, tais como obter as conexões, mapear objetos Java para tipos de dados SQL ou executar comandos SQL, devem ser feitas por classes de DAO. [fonte: http://pt.wikipedia.org/wiki/Data_Access_Object]*

# Padrão Singleton

***Model-view-controller** (**MVC**) é um padrão de arquitetura de software que visa separar a lógica de negócio da lógica de apresentação, permitindo o desenvolvimento, teste e manutenção isolado de ambos [fonte: http://pt.wikipedia.org/wiki/MVC]:*



# Padrão Singleton

1 – Garantir que a classe tenha apenas uma instância. Um exemplo:

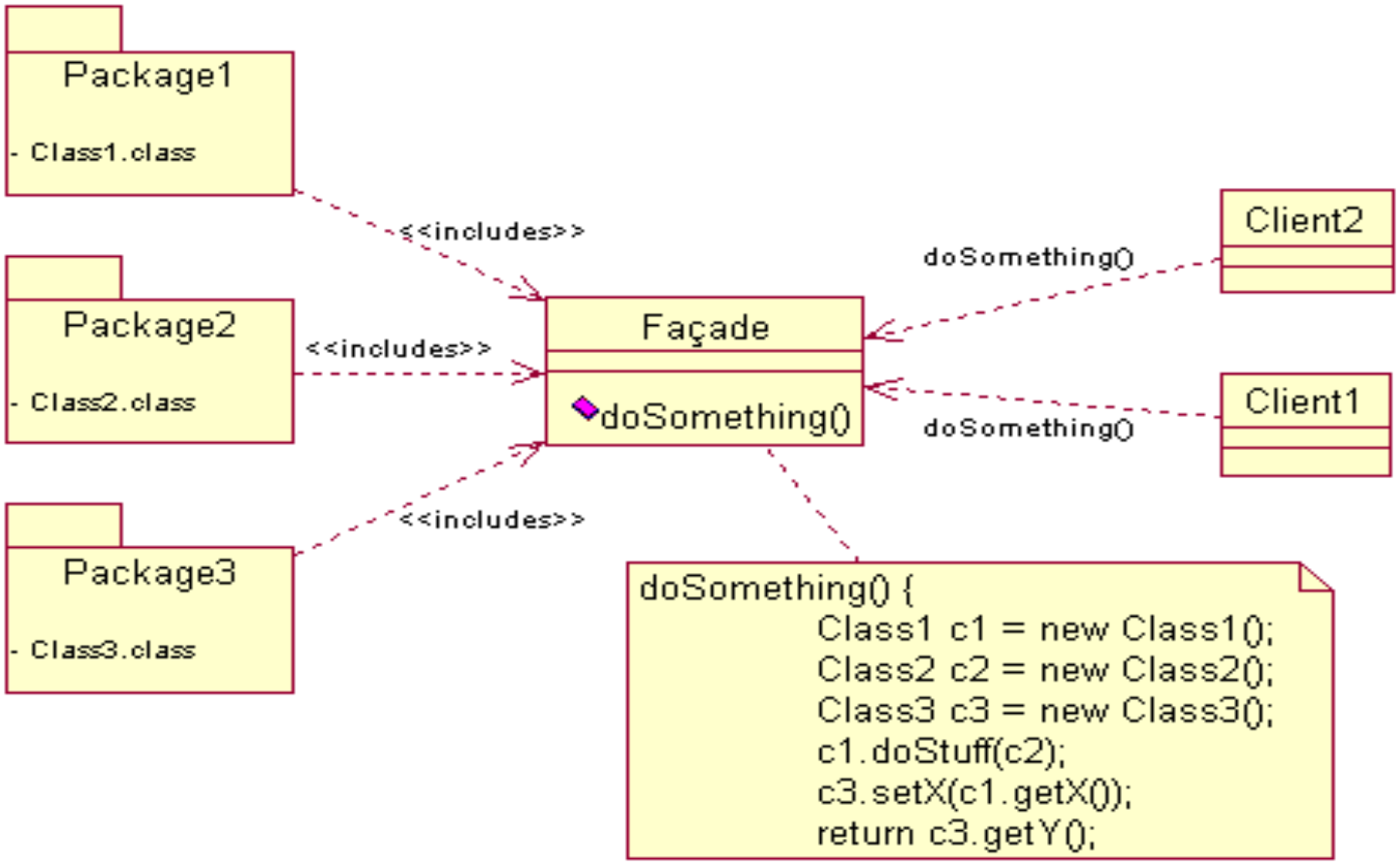
```
public class CidadeDAO {

    private EntityManagerFactory emf;
    private EntityManager em;
    private static boolean haveone = false;

    public CidadeDAO(EntityManagerFactory emf) throws RuntimeException {
        if (haveone)
            throw new RuntimeException("Only one instance allowed");

        this.emf = emf;
        haveone = true;
    }
```

# Padrão Façade

- Categoria: Estrutural

- Provê uma interface única para um conjunto de interfaces em um subsistema.

# Padrão Façade



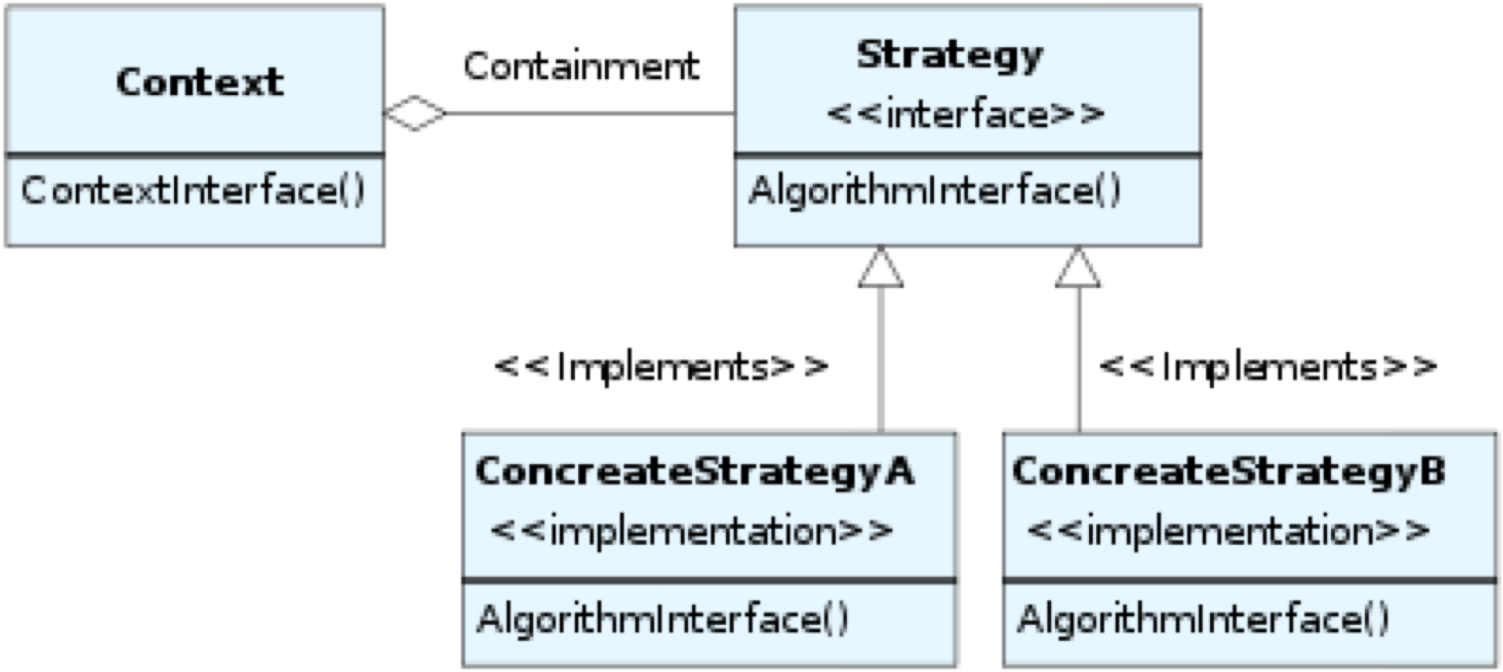
# Padrão Strategy

- Categoria: Comportamental

- Para representar uma operação a ser realizada sobre os elementos de uma estrutura de objetos.
- definir novas operações sem alterar as classes dos elementos sobre os quais opera.

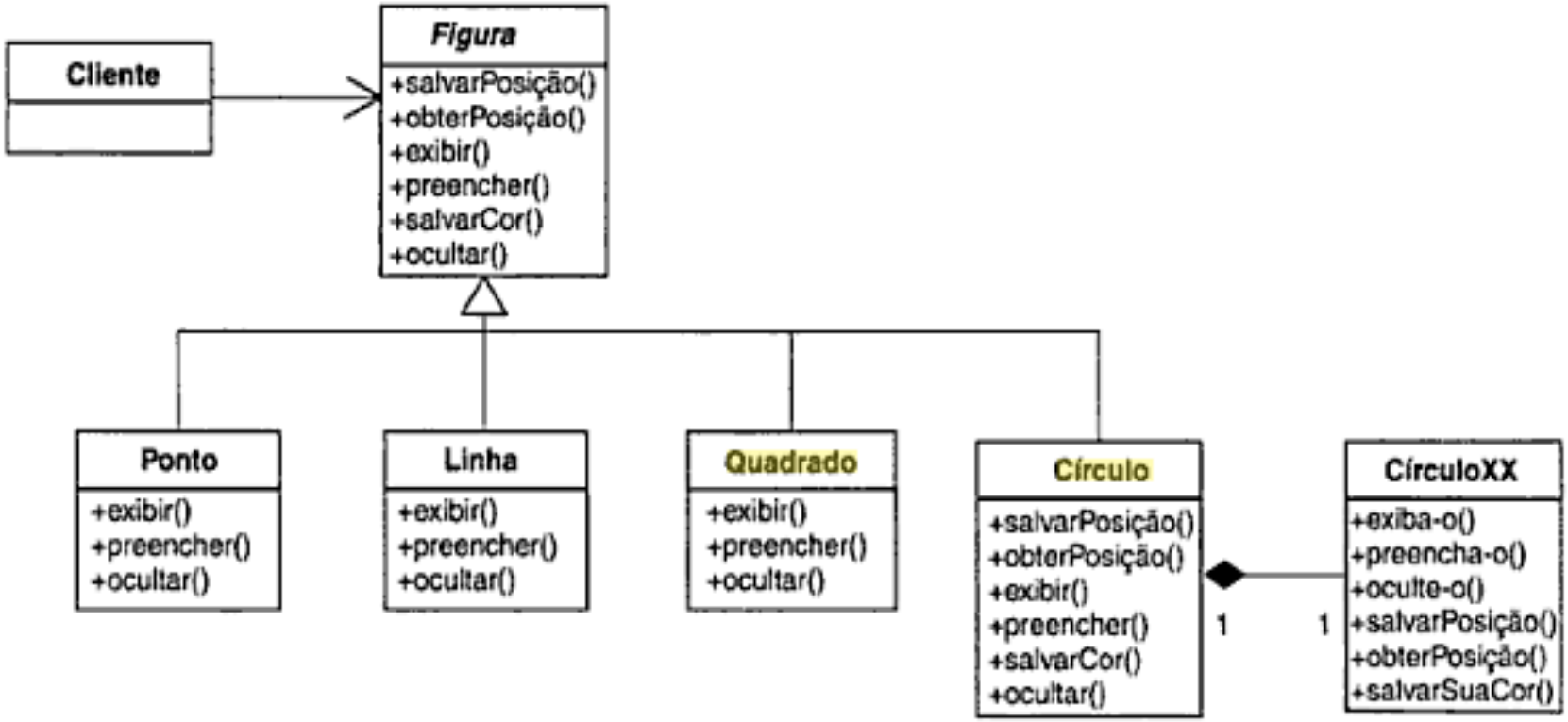
- Exemplo: Definir uma família de algoritmos e encapsular cada algoritmo como uma classe, permitindo assim que elas possam ter trocados entre si. Este padrão permite que o algoritmo possa variar independentemente dos clientes que o utilizam.

# Padrão Strategy

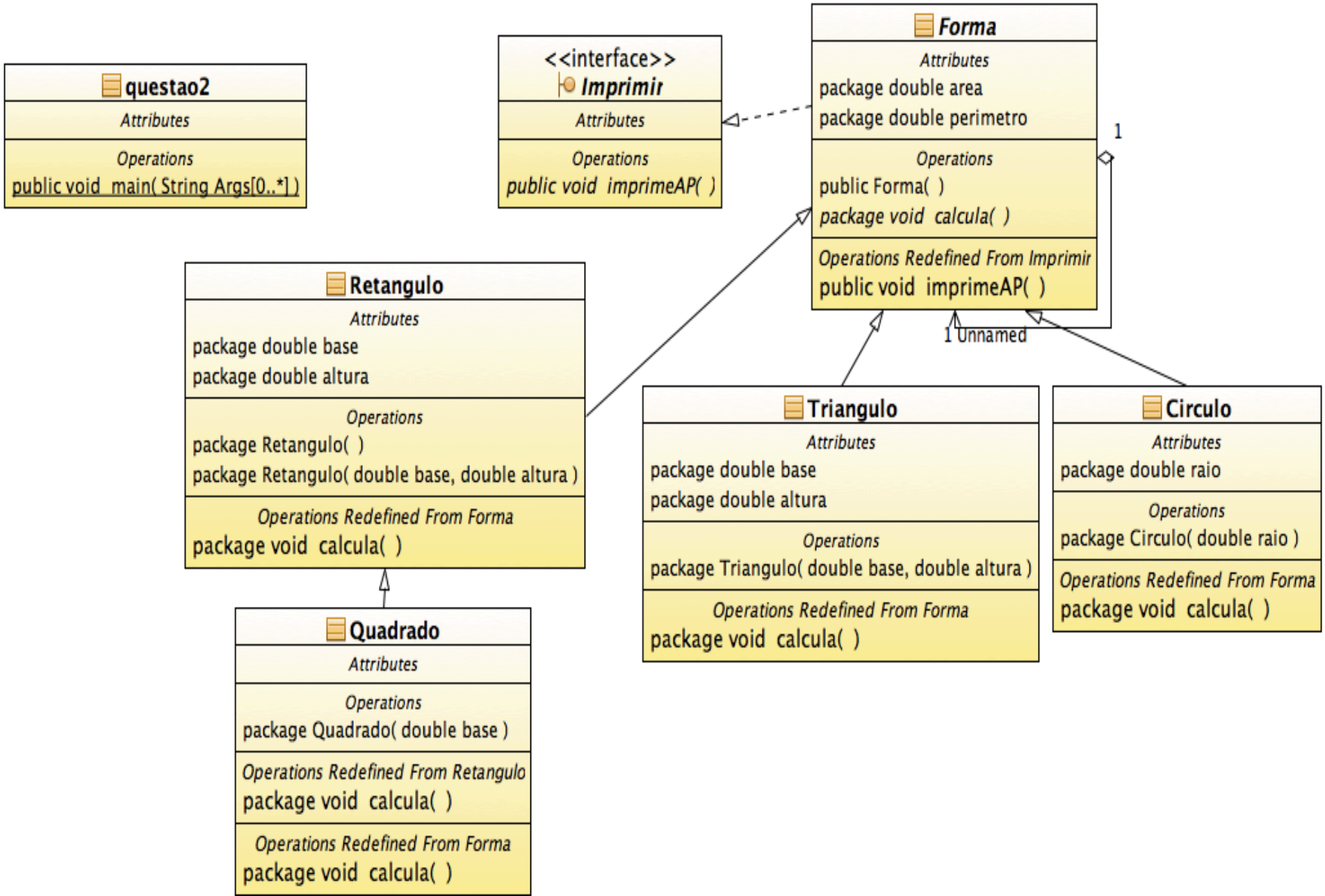



Ver exemplo: http://en.wikipedia.org/wiki/Strategy_pattern

# Exemplo interessante:



# Exemplo


questao2
Attributes
Operations
public void main( String Args[0..*] )
<<interface>>
Imprimir
Attributes
Operations
public void imprimeAP( )
Forma
Attributes
package double area
package double perimetro
Operations
public Forma( )
package void calcula( )
Operations Redefined From Imprimir
public void imprimeAP( )
1
1 Unnamed
Retangulo
Attributes
package double base
package double altura
Operations
package Retangulo( )
package Retangulo( double base, double altura )
Operations Redefined From Forma
package void calcula( )
Triangulo
Attributes
package double base
package double altura
Operations
package Triangulo( double base, double altura )
Operations Redefined From Forma
package void calcula( )
Circulo
Attributes
package double raio
Operations
package Circulo( double raio )
Operations Redefined From Forma
package void calcula( )
Quadrado
Attributes
Operations
package Quadrado( double base )
Operations Redefined From Retangulo
package void calcula( )
Operations Redefined From Forma
package void calcula( )

# Exemplo

```
interface Imprimir {
    public void imprimeAP();
}

// exemplo de classe abstrata
abstract class Forma implements Imprimir {

    double area, perimetro;
    static ArrayList<Forma> formas;

    public Forma() {
        formas = new ArrayList<Forma>();
    }

    abstract void calcula();

    @Override
    public void imprimeAP() {
        System.out.println("área e perímetro do " + getClass() + ":" + area + "," + perimetro);
    }
}
```

# Exemplo

```
class Circulo extends Forma {

	double raio;

	Circulo(double raio) {
		this.raio = raio;
		calcula();
	}

	@Override
	void calcula() {
		perimetro = 2 * Math.PI * raio;
		area = Math.PI * raio * raio;
	}
}

// será considera triângulo equilátero
class Triangulo extends Forma {

	double base, altura;

	Triangulo(double base, double altura) {
		this.base = base;
		this.altura = altura;
		calcula();
	}

	@Override
	void calcula() {
		perimetro = 3 * base;
		area = base * altura / 2;
	}
}
```

# Exemplo

```
class Retangulo extends Forma {
      double base, altura;
      Retangulo() {
            base = altura = 0;
      }

      Retangulo(double base, double altura) {
            this.base = base;
            this.altura = altura;
            calcula();
      }
      // exemplo de sobreescrita no método calcula

      @Override
      void calcula() {
            perimetro = 2 * (base + altura);
            area = base * altura;
      }
}

class Quadrado extends Retangulo {
      // exemplo de sobrecarga no construtor
      Quadrado(double base) {
            super.base = base;
            calcula();
      }

      @Override
      void calcula() {
            perimetro = 4 * base;
            area = base * base;
      }
}
```

# Exemplo

```
public class questao2 {

    public static void main(String Args[]) {

        Circulo c = new Circulo(3);
        Triangulo t = new Triangulo(3, 2);
        Quadrado q = new Quadrado(2);
        Retangulo r = new Retangulo(2, 3);

        Forma.formas.add(c); // exemplo de polimorfismo em Forma.formas
        Forma.formas.add(t);
        Forma.formas.add(q);
        Forma.formas.add(r);
        for (Forma aux : Forma.formas) {
            aux.imprimeAP();
        }

    }
}
```