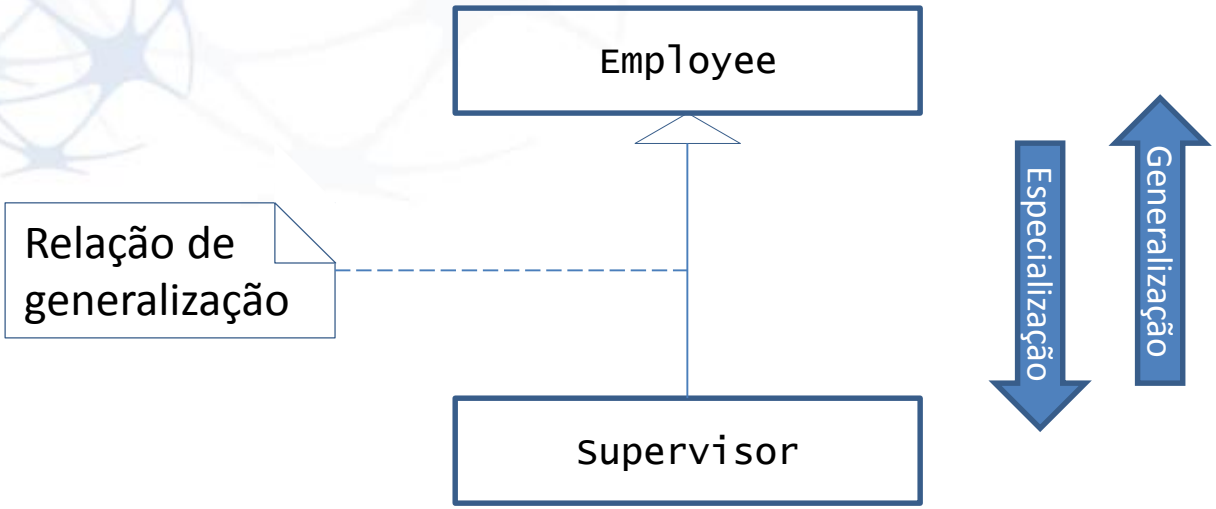
Herança e Polimorfismo

Avisos

- Questionário sobre plágio
- FISTA –
  - Inscrições Workshps abertas,
  - Concurso de programação (1 tablet)
- Enunciado – ASAP
- 1º mini-teste online (deadline Terça)
- Inscrição no forum

Generalização (relação)



- Um `Supervisor` é um `Employee`.
- Um `Employee` pode ser um `Supervisor`.
- Employee é a classe- base / super-class
- Supervisor é a classe derivada (especialização) / sub-class

Herança

```
public class Employee {
    public Employee(final String
  name) {
        …
    }
     public String getName() { … }
}
```

Herança

```
public class Supervisor extends Employee {
   public Supervisor(…
}

public static void main (String[] args) {
   Supervisor sup = new Supervisor(“Joaquina”, …);
   sup.getName();
}
```

Herança

Um Supervisor é um Employee.

```
public class Supervisor extends Employee {
    private String supervisedSection;
    public Supervisor(final String name, String supervisedSection) {
        super(name);
        this.supervisedSection = supervisedSection;
    }
     public String getSupervisedSection() {…}
    @Override
    public String getName() {
        return super.getName() + " (Supervisor);
    }
}
```

Novo método específico da classe Supervisor.

Sobrepõe-se ao método com a mesma assinatura na classe base Employee.

Herança - Polimorfismo

```
Employee employee = new Employee("Felisberto”);


Supervisor supervisor = new
   Supervisor("Guilhermina”, “Secção Congelados”);


Employee supervisor = new Supervisor("Guilhermina”,
   “Secção Congelados”);


Supervisor sup = new Employee("Felisberto”);
```

Herança - Polimorfismo

```
supervisor.getName()
supervisor.getSupervisedSection()
employee.getName()
employee.getSupervisedSection()
```




Uma operação pode ser implementada por diferentes métodos

Sobreposição

- Método de classe derivada pode sobrepor-se a método de classe base
- Sobreposição é especialização
- Regras
  - Mesma assinatura e tipo de devolução compatível
  - Método na classe base não privado e não final
  - Método na classe derivada com acessibilidade igual ou superior

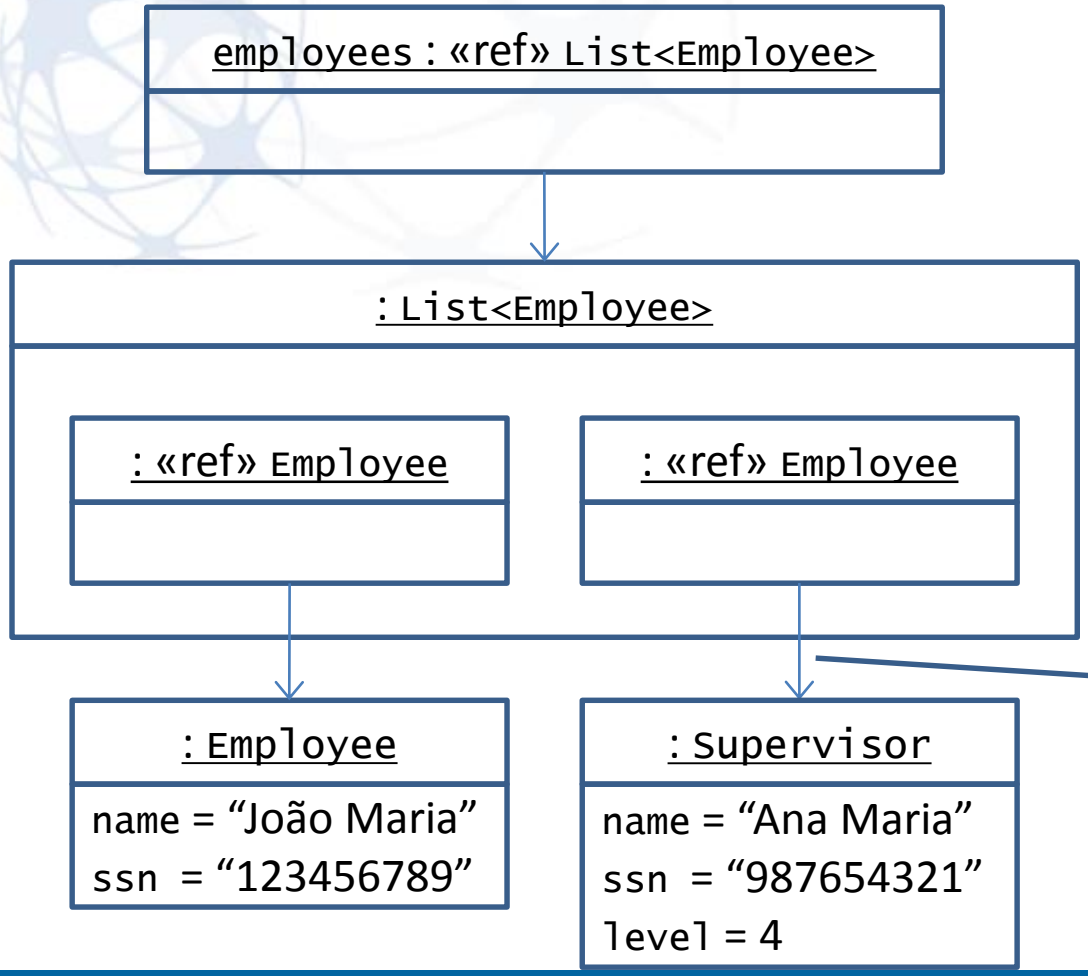
Categorias de acesso
(de novo, agora todas)

- Características ou membros podem ser
  - `private` – acesso apenas por outros membros da mesma classe
  - *package-private* (sem qualificador) – adicionalmente, acesso por membros de classes do mesmo pacote
  - `protected` – adicionalmente, acesso por membros de classes derivadas
  - `public` – acesso universal

Interfaces de uma classe

- Dentro da própria classe tem-se acesso a:
  - Membros da classe e membros não privados de classes base
- Nas classes do mesmo pacote tem-se acesso a:
  - Membros não privados da classe ou suas bases
- Numa classe derivada:
  - Membros protegidos ou públicos da classe ou suas bases
- Noutras classes:
  - Membros públicos da classe ou suas bases

Organização



Possível porque a classe `Supervisor` deriva da classe `Employee`, ou seja, possível porque um `Supervisor` é (sempre também) um `Employee`.

Exemplo

```
List<Employee> employees = …

employees.add(new Employee("João Maria));
employees.add(new Supervisor("Ana Maria",
                                “Mercearia”));
…
```

Qual o método `getName()` executado?

```
for (Employee employee : employees)
    out.println(employee.getName());
```

Resultado

João Maria
Ana Maria (Supervisor)
_

A classe `Object`

```
public class Employee { … }
```

```
public class Employee extends
  Object { … }
```

Se uma classe não derivar explicitamente de outra, derivará implicitamente da classe `Object`, que está no topo da hierarquia de classes do Java.

Acesso à classe base

```
public class Base {
    public String className() {
        return "Base";
    }
}
public class Derived extends Base {
    @Override
    public String className() {
        return “Derived”;
    }
    public void testCalls() {
        Base base = this;
        out.println("Through this:  " + this.className());
        out.println("Through base:  " + base.className());
        out.println("Through super: " + super.className());
    }
}
```

```
Through this:  Derived
Through base:  Derived
Through super: Base
_
```

Métodos finais (`final`)

- Método final não pode ser sobreposto
- Razão para um método ser final:
  - Programador que forneceu o método na classe base entendeu que classes derivadas não deveriam poder especializar / alterar o modo de funcionamento desse método

Análise: conceitos

- Veículo
- Motociclo
- Automóvel
- Honda NX 650
- Audi TT

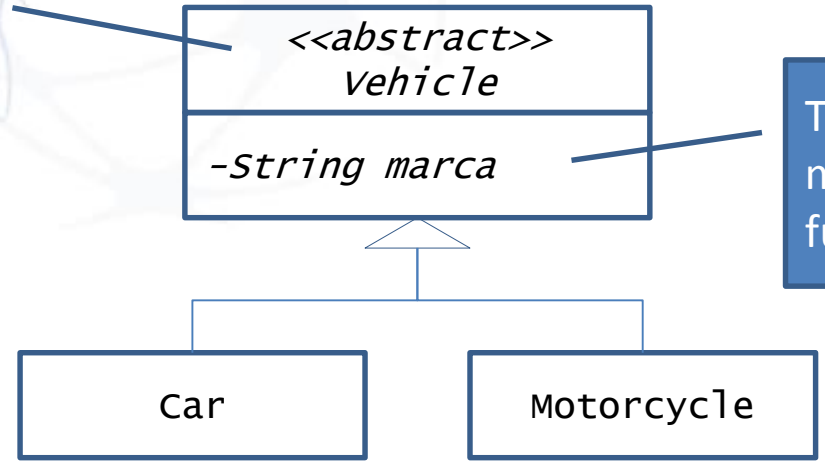
Análise inicial pode resultar num dicionário ou glossário do domínio.

Sub-tipos de veículos

Marcas

Classe abstracta: Análise e desenho

Existem veículos que não têm sub-tipo?



Todos os veículos têm marca? Estes e outros futuros? Barcos? Aviões?

Existem instâncias de Carro?

Onde encaixa o conceito Moto4?
Número de rodas, onde deveria estar?

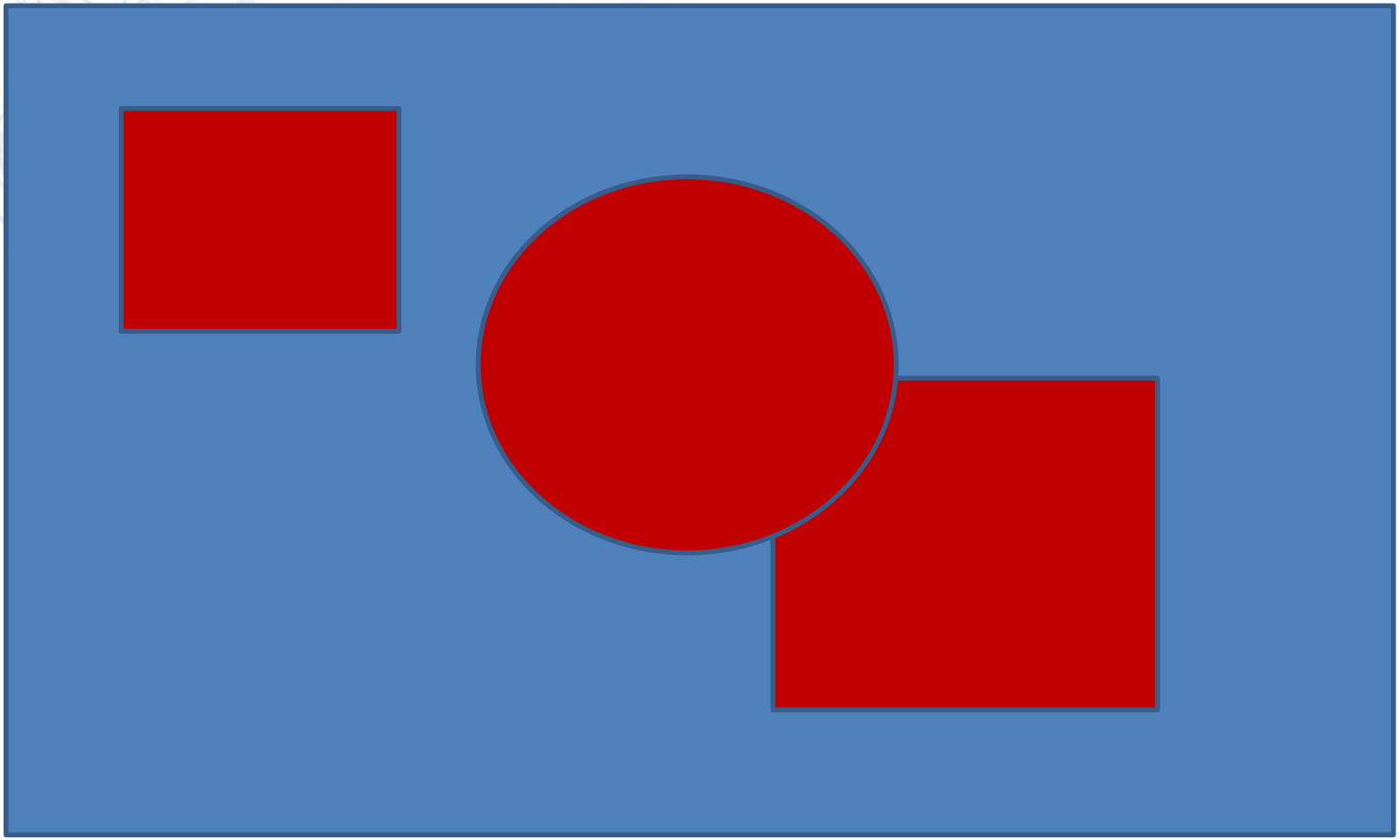
Implementação

```
public abstract class Vehicle {
    private String marca;
}

public class Car extends Vehicle {
    …
}

public class Motorcycle extends Vehicle {
    …
}
```

Classes e métodos abstratos

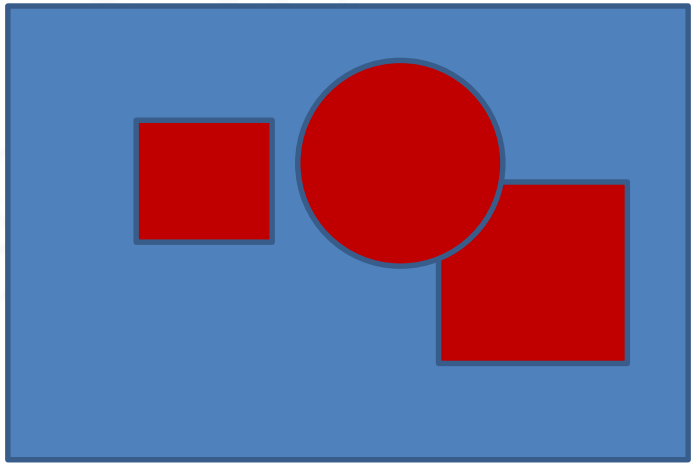
- Operação `abstract` é apenas declaração
- Classe com operação abstracta tem de ser abstracta
- Classe abstracta não pode ser instanciada, i.e., não se podem construir objectos
- Classe derivada de abstracta só pode ser concreta se implementar todas as operações abstractas

Editor de formas



Análise: conceitos

- Figura
- Forma
- Círculo
- Quadrado



Figure

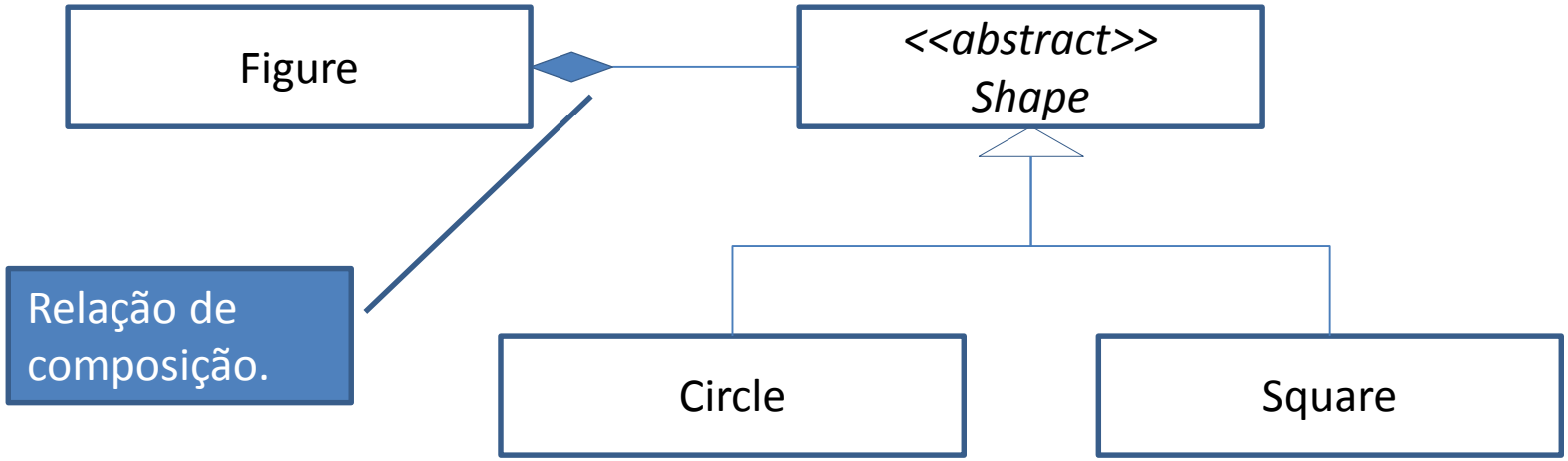
Shape

Circle

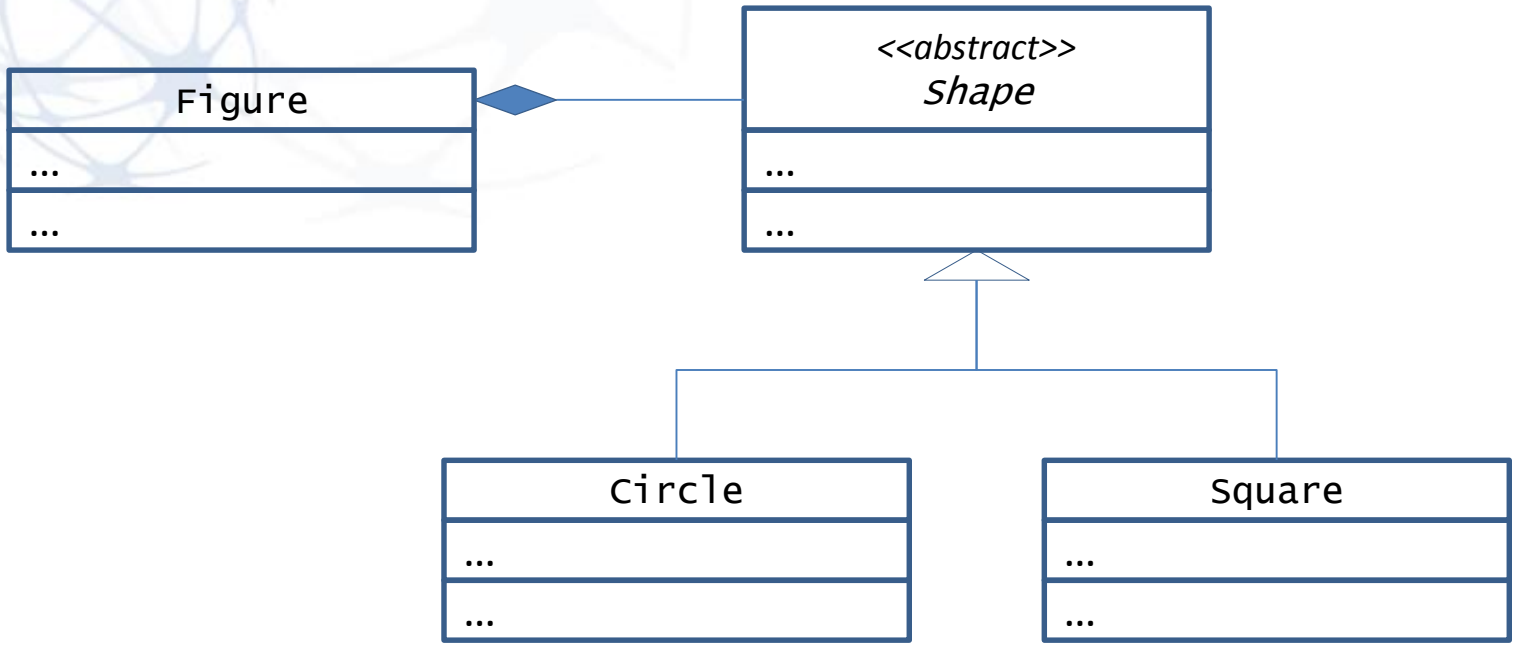
Square

Análise: relações

- Uma Figura é composta de Formas
- Um Círculo é uma Forma
- Um Quadrado é uma Forma



Desenho



Implementação

```
public class Figure {
    private List<Shape> shapes;
    …
}
public abstract class Shape {    }

public class Circle extends Shape { … }

public class Square extends Shape {  … }
```

Implementação: Shape

```
public abstract class Shape {
    private Position position;
    public Shape(final Position position) {
        this.position = position;
    }
    public final Position getPosition() { return position; }
    public abstract double getArea();
    public abstract double getPerimeter();
    public abstract Box getBoundingBox();
    public void moveTo(final Position newPosition) {
        position = newPosition;
    }
}
```

Qual a área de uma "forma"??

Operações abstractas, ou seja, operações sem qualquer implementação disponível até este nível da hierarquia.

Implementação: `Circle`

```
public class Circle extends Shape {
    private double radius;
    public Circle(final Position position,
                  final double radius) {
        super(position);
        this.radius = radius;
    }
    public final double getRadius() {
        return radius;
    }
    …
```

Um `Circle` é uma `Shape` e a classe `Circle` herda a implementação da classe `Shape`.

É necessário apenas um atributo adicional, correspondente a uma das duas propriedades de um círculo (o raio), já que a posição do centro é herdada da classe `Shape`.

Invocação do construtor da classe base…

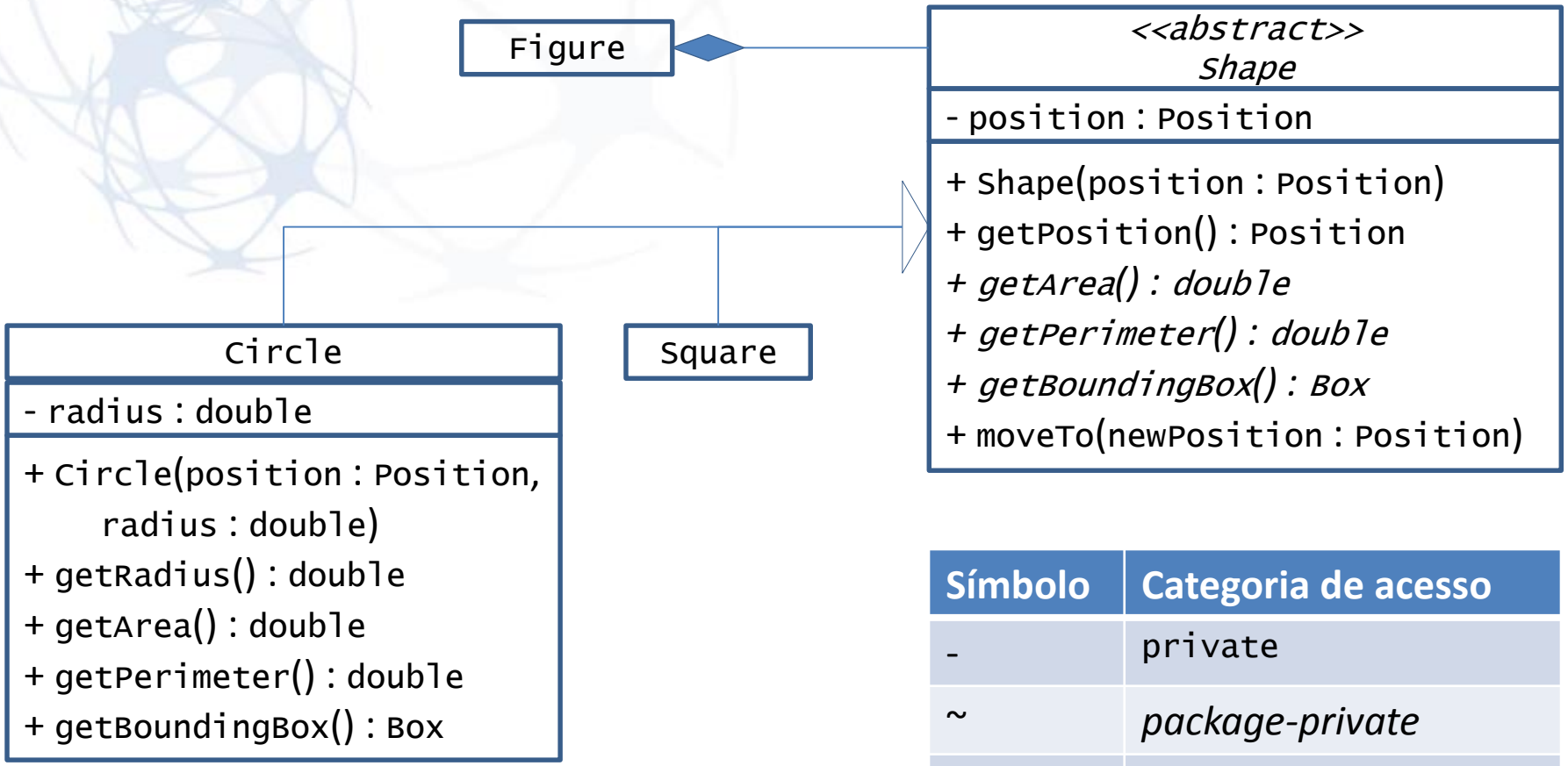
Implementação: `circle`

```
@Override
public double getArea() { return Math.PI * getRadius() * getRadius();}
@Override
public double getPerimeter() {return 2.0 * Math.PI * getRadius();}
@Override
public Box getBoundingBox() {
```

Qual a área de um círculo? Fácil, $\pi \times r^2$.

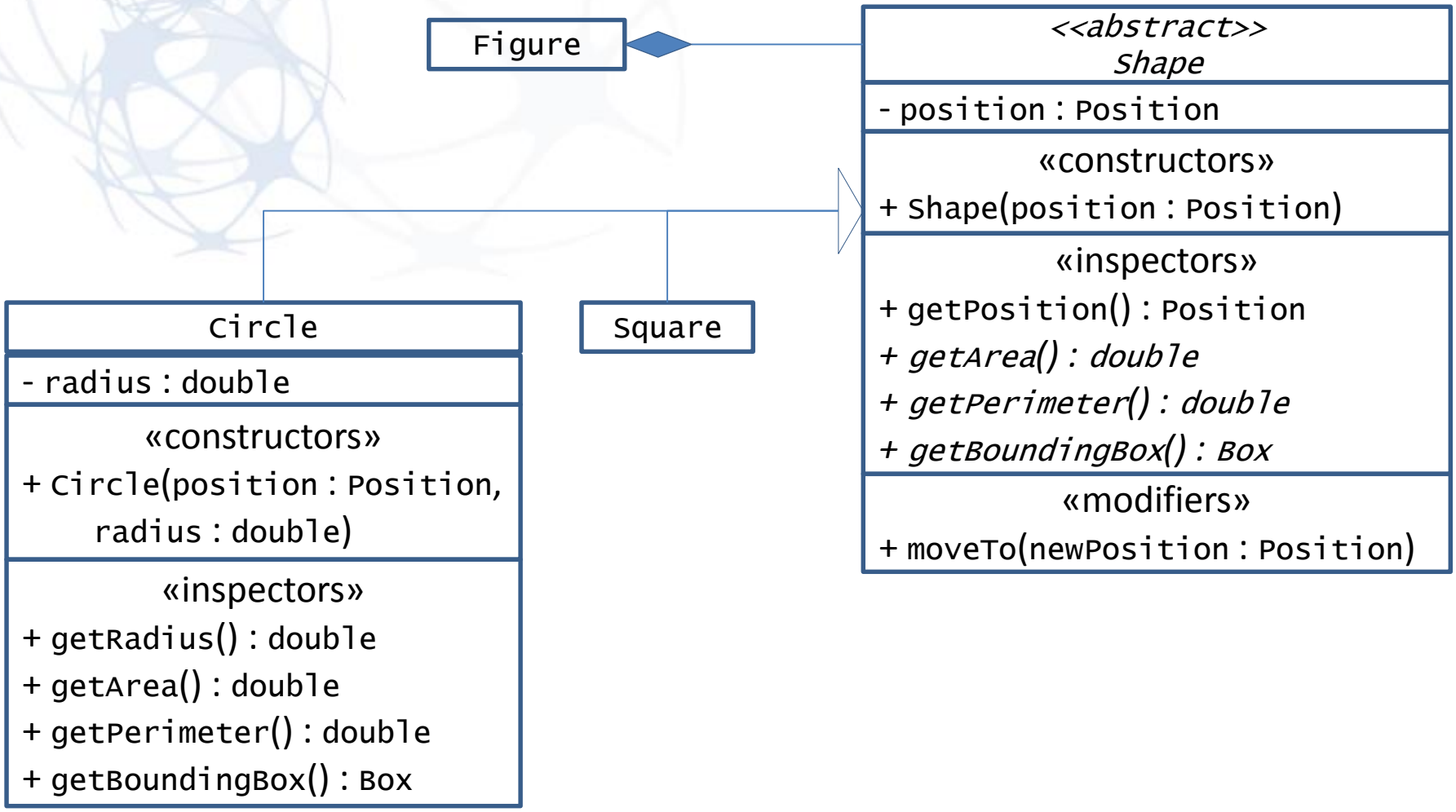
Implementações: métodos concretos para cada uma das operações abstractas da classe `Shape`.

Desenho pormenorizado



Símbolo	Categoria de acesso
-	private
~	*package-private*
#	protected
+	public

Desenho pormenorizado



Mais informação / Referências

- Y. Daniel Liang, *Introduction to Java Programming*, 7.ª edição, Prentice-Hall, 2010.

Sumário

- Herança e Polimorfismo