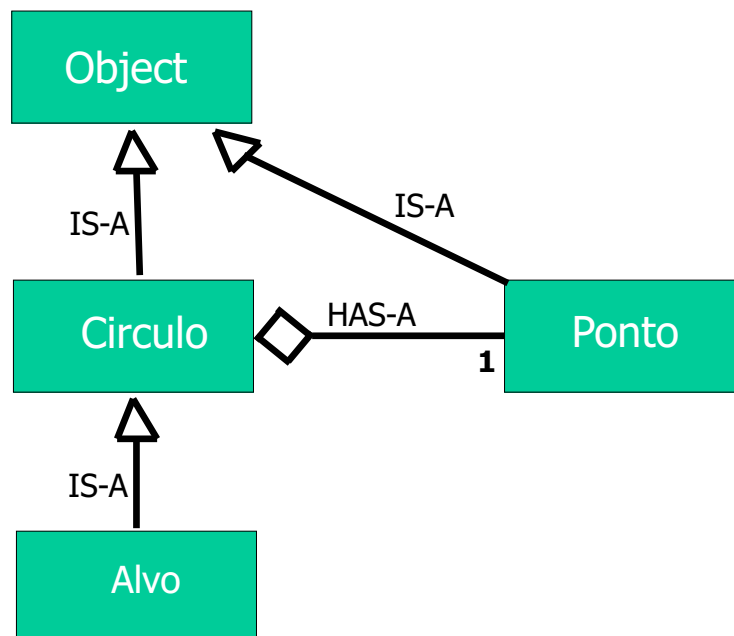


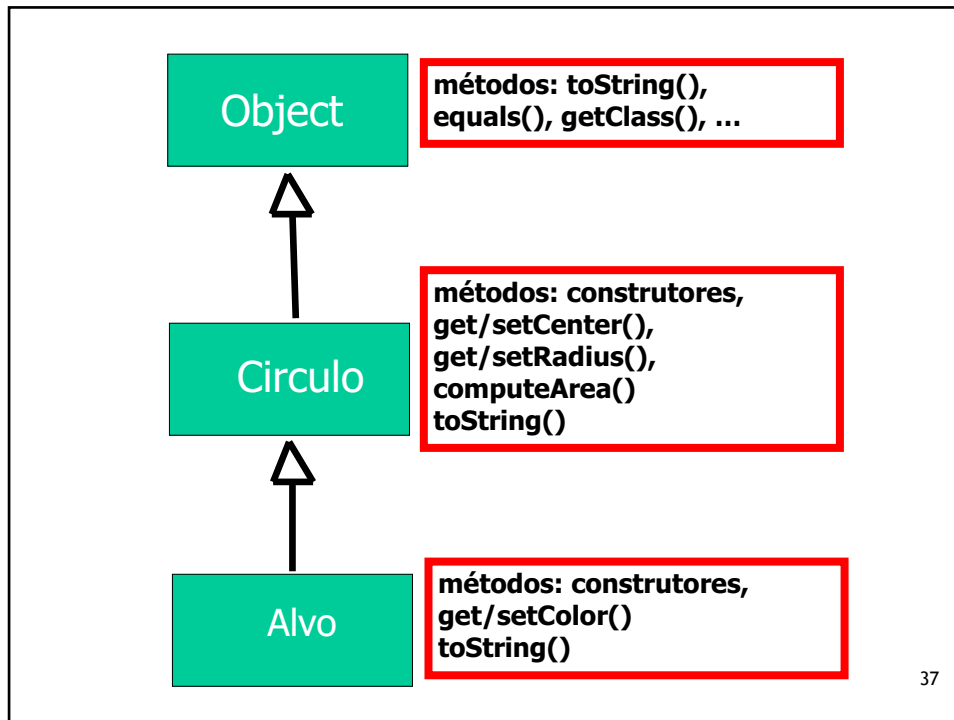
Java Polimorfismo

UA, DETI, Programação III
José Luis Oliveira, Carlos Costa
2016/17

35



36



Upcasting e downcasting

double z = 2.75;
int k = (int) z;
float x = k;
double w = 5;

downcast, $k \leftarrow 2$

upcast automático
 $x \leftarrow 2.0$; $w \leftarrow 5.0$

Alvo fc1 = new Alvo(1.5, 10, 20, Color.red);

Circulo c1;
c1 = fc1;

OK – um Alvo é um Circulo

Alvo fc2;
fc2 = c1;

Erro! – c1 é uma referência para Circulo. Mesmo que aponte para um Alvo precisa de downcast

fc2 = (Alvo) c1;

OK

38

Upcasting e downcasting

```
Circulo c2 = new Circulo(1.5f, 10, 20);
```

```
fc2 = (Alvo) c2;
```

run-time error:
ClassCastException

- O tipo do objeto pode ser testado com o operador instanceof

```
if (c3 instanceof Alvo)  
    fc2 = (Alvo) c3;
```

OK

39

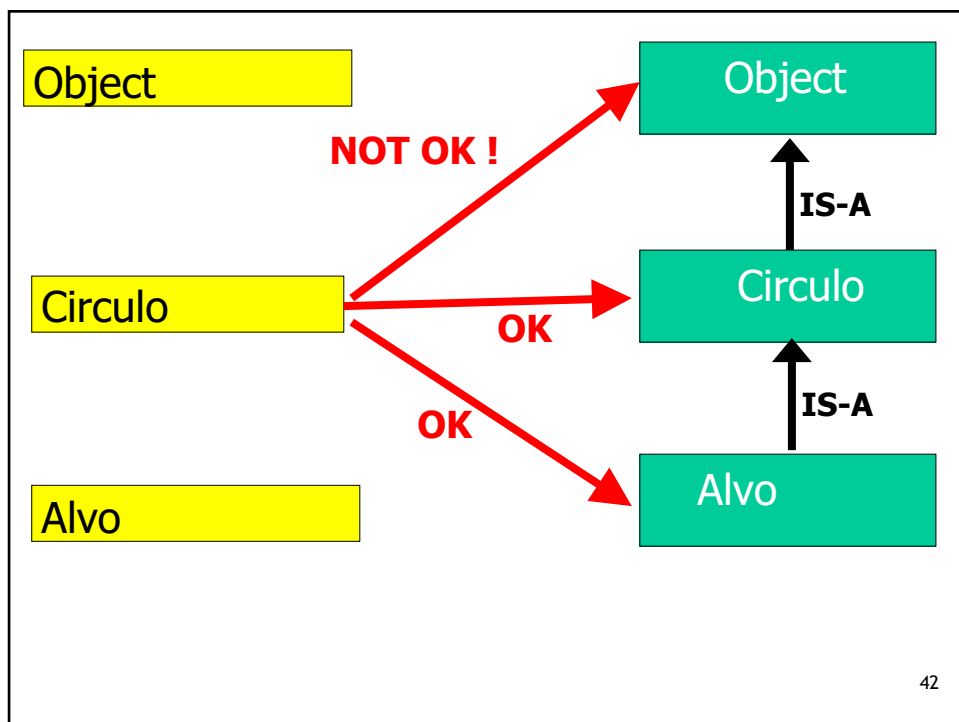
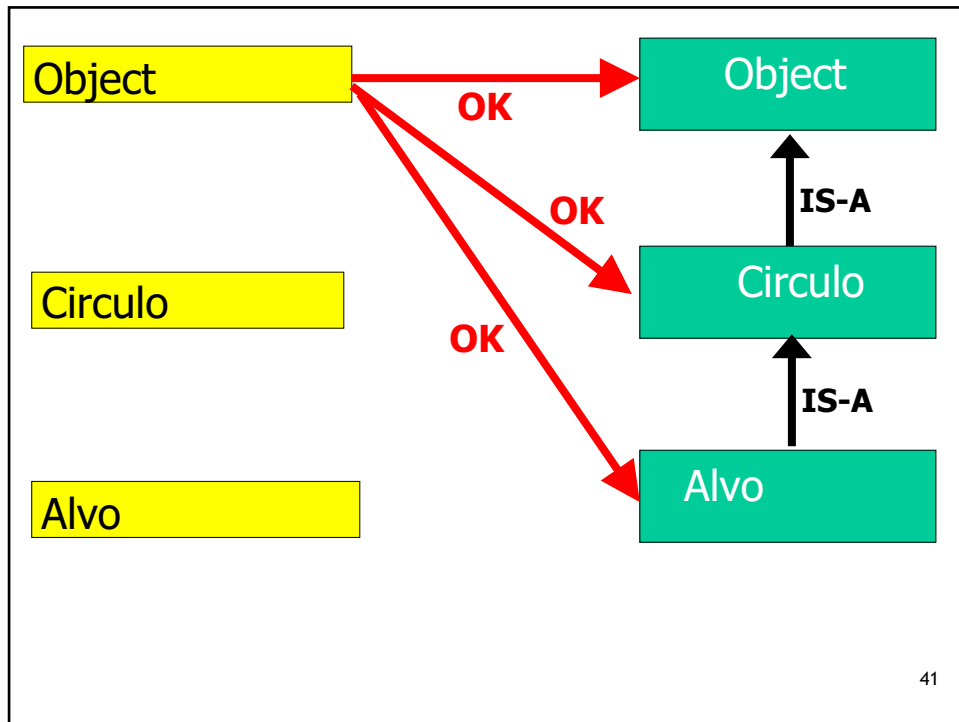
Polimorfismo

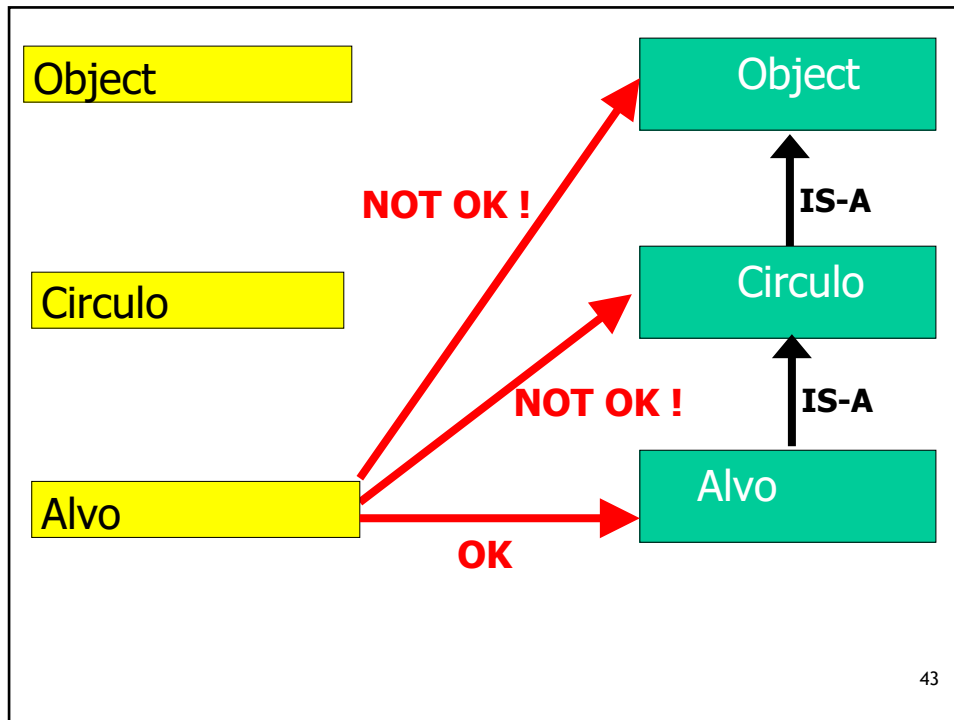
- Ideia base:
 - o tipo declarado na referência não precisa de ser exatamente o mesmo tipo do objeto para o qual aponta - pode ser de qualquer tipo derivado

```
Circulo c1 = new Alvo(...);  
Object obj = new Circulo(...);
```

- Referência polimórfica
 - T ref1 = new S();
 - // OK desde que todo o S seja um T

40





43

Polimorfismo

- Polimorfismo é, conjuntamente com a Herança e o Encapsulamento, uma das características fundamentais da POO.
 - Formas diferentes com interfaces semelhantes.
- Outras designações:
 - Ligação dinâmica (Dynamic binding), late binding ou run-time binding
- Esta característica permite-nos tirar mais partido da herança.
 - Podemos, por exemplo, desenvolver um método X() com parâmetro CBase com a garantia que aceita qualquer argumento derivado de CBase.
 - O método X() só é resolvido em execução.
- Todos os métodos (à excepção dos *final*) são *late binding*.
 - O atributo *final* associado a uma função, impede que ela seja redefinida e simultaneamente dá uma indicação ao compilador para ligação estática (*early binding*) - que é o único modo de ligação em linguagens com o C.

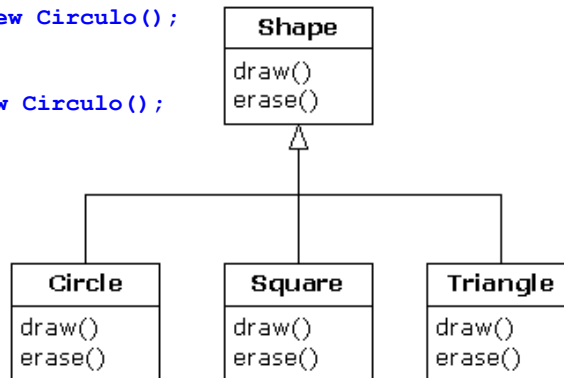
44

Exemplo 1

```
Shape s = new Shape();  
s.draw();
```

```
Circulo c = new Circulo();  
c.draw();
```

```
Shape s2 = new Circulo();  
s2.draw();
```



45

Exemplo 2

```
class Shape {
    void draw() {}
}

class Circle extends Shape {
    void draw() { System.out.println("Circle.draw()");
}

class Square extends Shape {
    void draw() { System.out.println("Square.draw()");
}

public class Shapes {
    public static Shape randShape() {
        switch((int)(Math.random() * 2)) {
            default:
                case 0: return new Circle();
                case 1: return new Square();
        }
    }

    public static void main(String[] args) {
        Shape[] s = new Shape[9];
        for(int i = 0; i < s.length; i++)
            s[i] = randShape(); // Fill up the array with shapes:
        for(int i = 0; i < s.length; i++)
            s[i].draw(); // Make polymorphic method calls:
    }
}
```

The diagram illustrates overlapping code snippets for drawing shapes. The visible code includes:

```

Circulo.draw()
Circulo.draw()
Circulo.draw()
Circulo.draw()
Square.draw()
Circulo.draw()
Square.draw()
Square.draw()
Square.draw()
Square.draw()
Square.draw()
Circulo.draw()
Circulo.draw()
Circulo.draw()
Square.draw()
Square.draw()
Square.draw()
Square.draw()

```

46

Generalização

- A generalização consiste em melhorar as classes de um problema de modo a torná-las mais gerais.

Formas de generalização:

- Tornar a classe o mais abrangente possível de forma a cobrir o maior leque de entidades.

```
class ZooAnimal;
```

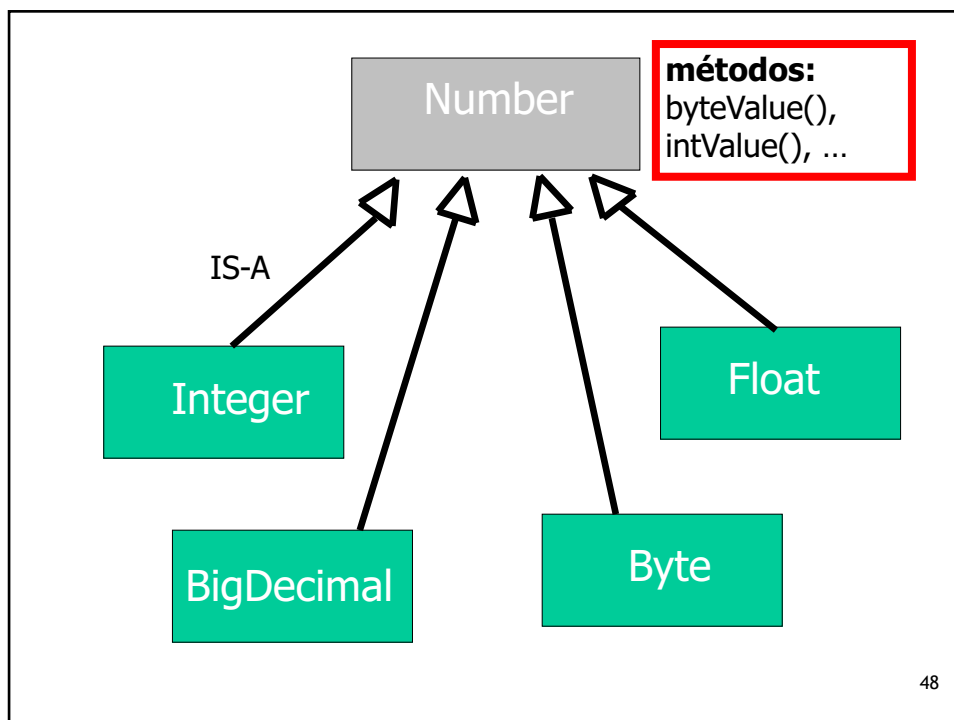
- Abstrair implementações diferentes para operações semelhantes em classes abstractas num nível superior.

```
ZooAnimal.draw();
```

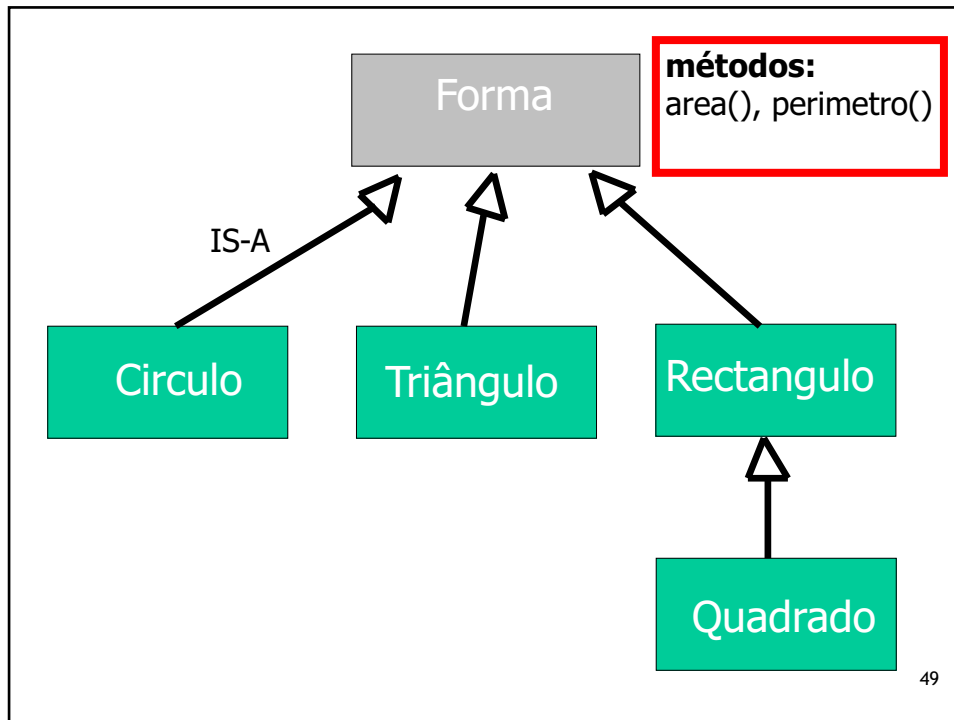
- Reunir comportamentos e características e fazê-los subir o mais possível na hierarquia de classes.

```
ZooAnimal.peso;
```

47

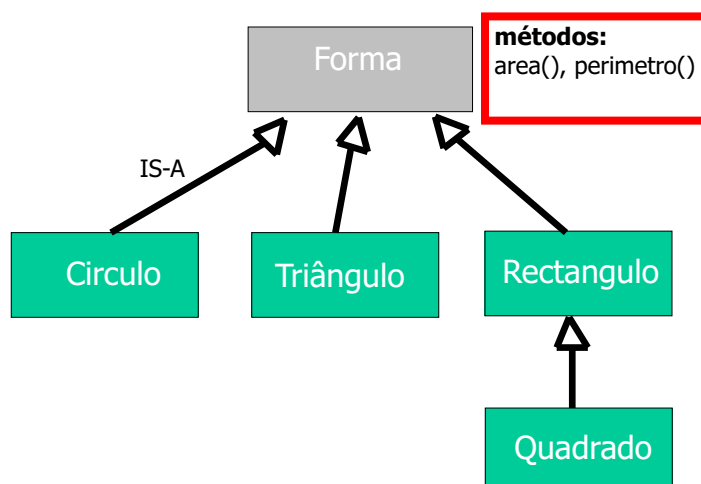


48



Questões?

- Como é que implementamos os métodos de Forma?



Classes abstractas

- Uma classe é abstracta se contiver pelo menos um método abstracto.

- Um método abstracto é um método cujo corpo não é definido.

```
public abstract class Forma
{
    // pode definir constantes
    public static final double DOUBLE_PI = 2*Math.PI;

    // pode declarar métodos abstractos
    public abstract double area();
    public abstract double perimetro();

    // pode incluir métodos não abstractos
    public String aka() { return "euclidean"; }
}
```

- Uma classe abstracta não é instanciável.

```
Forma f;           // OK. Podemos criar uma referência para Forma
f = new Forma();   // Erro! Não podemos criar Formas
```

51

Classes abstractas

- Num processo de herança a classe só deixa de ser abstracta quando implementar todos os métodos abstractos.

```
public class Circulo extends Forma {

    protected double r;

    public double area() {
        return Math.PI*r*r;
    }

    public double perimetro() {
        return DOUBLE_PI*r;
    }
}

Forma f;
f = new Circulo(); // OK! Podemos criar Circulos
```

52

Classes abstractas e Polimorfismo

```

abstract class Figura {
    abstract void doWork();
    protected int cNum;
}

class Circulo extends Figura {
    Circulo(int i) { cNum = i; }
    void doWork() { System.out.println("Circulo"); }
}

class Alvo extends Circulo {
    Alvo(int i) { super(i); }
    void doWork() { System.out.println("Alvo"); }
}

class Quadrado extends Figura {
    void doWork() { System.out.println("Quadrado"); }
}

public class ArrayOfObjects {
    public static void main(String[] args) {
        Figura[] anArray = new Figura[10];
        for (int i = 0; i < anArray.length; i++) {
            switch ((int) (Math.random() * 3)) {
                case 0 : anArray[i] = new Circulo(i); break;
                case 1 : anArray[i] = new Alvo(i); break;
                case 2 : anArray[i] = new Quadrado(); break;
            }
        }
        // invoca o método doWork sobre todas as Figura da tabela
        // -- Polimorfismo
        for (int i = 0; i < anArray.length; i++) {
            System.out.print("Figura(" + i + ") --> ");
            anArray[i].doWork();
        }
    }
}

```

```

Figura(0) --> Quadrado
Figura(1) --> Circulo
Figura(2) --> Quadrado
Figura(3) --> Circulo
Figura(4) --> Quadrado
Figura(5) --> Alvo
Figura(6) --> Circulo
Figura(7) --> Quadrado
Figura(8) --> Circulo
Figura(9) --> Quadrado

```

```

Figura(0) --> Circulo
Figura(1) --> Quadrado
Figura(2) --> Alvo
Figura(3) --> Quadrado
Figura(4) --> Alvo
Figura(5) --> Quadrado
Figura(6) --> Quadrado
Figura(7) --> Quadrado
Figura(8) --> Circulo
Figura(9) --> Quadrado

```

53