

# Computação II – Orientação a Objetos

---

Fabio Mascarenhas - 2016.1

<http://www.dcc.ufrj.br/~fabiom/java>

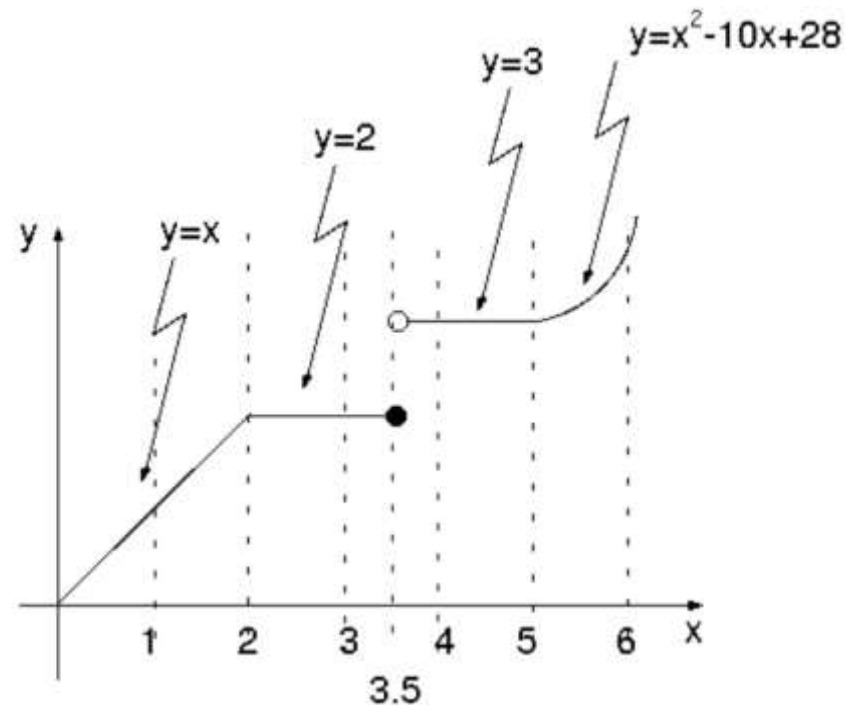
# Recursão estrutural

---

- Decoradores e compósitos são exemplos de *recursão estrutural*
- Recursão estrutural aparece sempre que uma ou mais partes de um objeto são similares ao todo
- Chamamos métodos que operam sobre essas partes de *métodos recursivos*
- Os métodos `getValor` das classes `Escala`, `Derivada`, `Soma` e `Composta` são exemplos de métodos recursivos

# Classes anônimas

- Algumas vezes queremos apenas uma única instância de uma classe que implementa alguma interface simples
- Por exemplo, queremos representar a função abaixo:



*função (double x) {  
if (x < 2) {  
return x;  
} else if (x < 3.5) {  
return 2;  
} else {  
return x<sup>2</sup> - 10x + 28;  
}  
}*

# Classes anônimas, cont.

---

- Podemos criar classes para representar o conceito de “função por partes”, e aí instanciar uma composição de objetos das classes que temos
- Ou podemos criar uma classe só para representar essa função, mas aí temos que criar um arquivo .java para ela, e dar um nome para essa classe...
- Ou podemos criar uma *classe anônima*!

```
new Funcao() {  
    public double getValor(double x) { ... }  
    public String getFormula() { ... }  
}
```

↑  
limites  
da classe  
anônima

# Classes anônimas, cont.

---

- Podemos usar uma classe anônima em qualquer lugar que podemos usar uma expressão
- Uma classe anônima pode ter campos, e outros métodos, mas não poderemos acessá-los de fora da classe, mesmo que sejam públicos
- Dentro de uma classe anônima, podemos usar campos e métodos visíveis naquele ponto do código, e usar variáveis locais visíveis que sejam declaradas como final
- Uma variável `final` não pode mudar seu valor depois de ser inicializada
- Para usar o `this` do ponto onde ela foi criada usamos `NomeDaClasse.this`

# Métodos default (Java 8)

---

- Normalmente uma interface deixa a implementação de seus métodos totalmente a cargo das classes que a implementam, mas ela pode ter uma *implementação default*
- Ela é anotada com a palavra-chave `default` antes da declaração do método

```
default Funcao derivada() {  
    return new Derivada(this);  
}
```

- Dentro de um método `default` só se tem acesso (seja direto ou via `this`) aos outros métodos declarados naquela interface