CONCEITOS BÁSICOS DE ORIENTAÇÃO A OBJETOS

ACH 2003 — COMPUTAÇÃO ORIENTADA A OBJETOS

Daniel Cordeiro

2 de março de 2016

Escola de Artes, Ciências e Humanidades | EACH | USP

VOLTEMOS A JAVA

Quais são os conceitos básicos de OO que vocês já viram?

VOLTEMOS A JAVA

Quais são os conceitos básicos de OO que vocês já viram?

- Objeto
- Classe
- Herança
- Interface
- Pacote

Programação Orientada a Objetos

É um estilo de programação centrado no uso de *objetos* para o projeto e construção de aplicações.

O QUE É UM OBJETO?

> *OOP to me means only messaging, local retention and protection and hiding of state-process, and extreme late-binding of all things. It can be done in Smalltalk and in LISP. There are possibly other systems in which this is possible, but I'm not aware of them.* — *Dr. Alan Kay*

Definição (apesar de não existir consenso)

Objeto é um componente de software que contém propriedades (estado) os métodos (comportamento) necessários para tornar um tipo de dado útil.

OBJETO

- Dados + Métodos
- Os métodos modificam o estado interno do objeto e servem como mecanismo primário para comunicação entre objetos

Encapsulamento (de dados)

Mecanismo de linguagens de programação para restringir o acesso ao estado interno de um objeto, fazendo com que toda interação com ele seja realizada através de seus métodos.

Benefícios

Modularidade: o código fonte de um objeto pode ser escrito e mantido independentemente do código de outros objetos

Encapsulamento: ao interagir com objetos pelos seus métodos, os detalhes de sua implementação interna se mantêm ocultos para o mundo externo

Reutilização: se um objeto já foi definido, você pode usá-lo no seu programa

***Pluggability* e facilidade de depuração:** se um objeto em particular se mostrar problemático, você pode removê-lo e plugar um outro em seu lugar

O QUE É UMA CLASSE?

Classe

é o modelo (ou protótipo) a partir do qual objetos individuais serão criados. Uma instância é um objeto construído a partir de uma classe.

```
class Bicicleta {
    int cadência = 0;
    int velocidade = 0;
    int marcha = 1;

    void mudarCadência(int novoValor) {
         cadência = novoValor;
    }
    void mudarMarcha(int novoValor) {
         marcha = novoValor;
    }
    void acelerar(int incremento) {
         velocidade = velocidade + incremento;
    }
    void frear(int decremento) {
         velocidade = velocidade - decremento;
    }
}
```

O QUE É HERANÇA?

Definição

É a capacidade de linguagens orientadas a objetos de permitir que classes herdem estados e comportamentos comuns a outras classes.

```
class MountainBike extends Bicicleta {
    // novos campos e métodos que definem
    // uma mountain bike
}
```

`MountainBike` é uma subclasse de `Bicicleta`.
`Bicicleta` é a superclasse de `MountainBike`.

O QUE É UMA INTERFACE?

Definição:

Interface é um conjunto de métodos que os objetos expõem para o mundo externo.

```
interface Bicicleta {

    // número de rotações da roda por minuto
    void mudarCadência(int novoValor);

    void mudarMarcha(int novoValor);

    void acelerar(int incremento);
}



class BicicletaBásica implements Bicicleta {
    ...
}
```

O QUE É UM PACOTE?

Definição:

Um pacote é um espaço de nomes que organiza um conjunto de classes e interfaces relacionadas.

```
package bicicleta;

interface Bicicleta { ... }

class BicicletaBásica implements Bicicleta { ... }

class MountainBike extends BicicletaBásica { ... }
```

Fully qualified name de uma classe:

`bicicleta.MountainBike`

```
int cadência = 0;
int velocidade = 0;
int marcha = 1;
```

A linguagem Java define os seguintes tipos de variáveis:

Variáveis de classe : (campos estáticos), definidos com o modificador `static`, que indica para o compilador que existe apenas uma cópia dessa variável, independentemente do número de vezes que a classe foi instanciada

Variáveis de Instância : (campos que não são estáticos) seus valores são únicos para cada instância de uma classe

Variáveis locais: variáveis temporárias que só existem no escopo de um método

Parâmetros: variáveis que armazenam os valores (objetos) passados na chamada a um método

TIPOS PRIMITIVOS DE DADOS

byte	8-bits $[-128; 127]$
short	16-bits $[-32.768; 32.767]$
int	32-bits $[-2^{31}; 2^{31} - 1]$
long	64-bits $[-2^{63}; 2^{63} - 1]$
float	ponto flutuante de precisão simples de 32-bits
double	ponto flutuante de precisão dupla de 64-bits
boolean	`true` ou `false`
char	1 caractere Unicode de 16-bits

Além de suporte especial para cadeias de caracteres. Ex: `"isso é uma String"` (java.lang.String)

OPERADORES

Operador	Precedência
postfix	`expr++ expr--`
unary	`++expr --expr +expr -expr ~ !`
multiplicative	`* / %`
additive	`+ -`
shift	`<< >> >>>`
relational	`< > <= >=` instanceof
equality	`== !=`
bitwise AND	`&`
bitwise exclusive OR	`^`
bitwise inclusive OR	`|`
logical AND	`&&`
logical OR	`||`
ternary	`? :`
assignment	`= += -= *= /= \%= \&= ^= |= <<= >>= >>>=`

Tabela 1: Operadores e suas precedências. Quanto mais no topo da tabela, maior a precedência do operador.

if–then–else

```
void brecar() {
    if (estáEmMovimento) {
        velocidadeAtual--;
    } else {
        System.err.println("A bicicleta já está parada!");
    }
}
```

switch

```
public static int getMonthNumber(String month) {
    int monthNumber = 0;

    if (month == null) {
        return monthNumber;
    }

    switch (month.toLowerCase()) {
        case "january":
            monthNumber = 1;
            break;

        /*          ...          */

        case "december":
            monthNumber = 12;
            break;
        default:
            monthNumber = 0;
            break;
    }
    return monthNumber;
}
```

WHILE E DO–WHILE

```java
class WhileDemo {
    public static void main(
                        String[] args){

        int count = 1;
        while (count < 11) {
            System.out.println(
              "Count is: " + count);
            count++;
        }
    }
}
```

```java
class DoWhileDemo {
    public static void main(
                        String[] args){

        int count = 1;
        do {
            System.out.println(
              "Count is: " + count);
            count++;
        } while (count < 11);
    }
}
```

FOR

```java
class ForDemo {
  public static void main(String[] args){
    for(int i=1; i<11; i++){
      System.out.println(
                 "Count is: " + i);
    }
  }
}
```

```java
class EnhancedForDemo {
  public static void main(
                       String[] args){
    int[] numbers =
          {1,2,3,4,5,6,7,8,9,10};
    for (int item : numbers) {
      System.out.println(
                   "Count is: " + item);
    }
  }
}
```

DECLARAÇÃO DE CLASSES

```
class MinhaClasse extends MinhaSuperClasse implements SuaInterface {

  // campos
  int meuInteiro;
  double meuDouble;

  // construtor
  public MinhaClasse(int i, double d) {
    this.meuInteiro = i;
    this.meuDouble = d;
  }

  // métodos
  public void duplicaInteiro() {
    this.meuInteiro *= 2;
  }
}

// Instanciação de objeto
MinhaClasse mc = new MinhaClasse(17, 3.14159);
```

MÉTODOS QUE DEVOLVEM OBJETOS

Suponha que:

```
public class Número extends Object { ... }
public class NúmeroImaginário extends Número { ... }
```

Pergunta. O método:

```
public Número devolveUmNúmero() {
    ...
}
```

1. Pode devolver um objeto do tipo Object?

MÉTODOS QUE DEVOLVEM OBJETOS

Suponha que:

```
public class Número extends Object { ... }
public class NúmeroImaginário extends Número { ... }
```

Pergunta. O método:

```
public Número devolveUmNúmero() {
    ...
}
```

1. Pode devolver um objeto do tipo Object?
2. Pode devolver um objeto do tipo NúmeroImaginário?

Modificador	Classe	Pacote	Subclasse	Mundo
`public`	Sim	Sim	Sim	Sim
`protected`	Sim	Sim	Sim	Não
(sem modificador)	Sim	Sim	Não	Não
`private`	Sim	Não	Não	Não

Tabela 2: Níveis de acesso

MÉTODOS E VARIÁVEIS DE CLASSE

```
public class Bicicleta {
    private int cadência;   private int marcha;
    private int velocidade;
    private int id;
    private static int númeroDeBicicletas = 0;

    public Bicicleta(int cadênciaInicial, int velocidadeInicial,
                                          int marchaInicial){
        marcha = marchaInicial;
        cadência = cadênciaInicial;
        velocidade = velocidadeInicial;

        // incrementa o número de bicicletas
        // e o atribui como identificador
        id = ++númeroDeBicicletas;
    }

    public int getID() {
        return id;
    }

    public static int getNúmeroDeBicicletas() {
        return númeroDeBicicletas;
    }
}
```

ATRIBUIÇÃO DE VARIÁVEIS ESTÁTICAS

```
class MinhaClasse {
  // variáveis e métodos de classe
  public static float PI = 3.14159;

  public static int ID = geraID();
  private static int geraID() { ... }

  // variáveis e métodos de instância
  protected int x = 10;

  public int y = getY();
  public int getY() { ... }
}
```

Forma de metadados

Provê informação sobre um programa que não faz parte do programa em si; não tem efeito sobre o código que eles anotam.

Usado pelo:

Compilador para detectar erros ou suprimir *warnings*

Instalador programas de instalação podem usar a informação para gerar código, arquivos XML, etc.

Interpretador algumas anotações podem ser examinadas em tempo de execução

```
@Test
void doisMaisDoisSupostamenteSãoQuatro() { ... }

@Autor(
    nome = "Daniel Cordeiro",
    data = "02/03/2016"
)
class MinhaClasse() { ... }

@Autor(nome="Daniel Cordeiro")
@Override
void meuSuperMétodo() { ... }

@interface Autor {
    String nome() default "Desconhecido";
    String data();
```

ANOTAÇÕES NO JAVA 8

Java SE 8[1] permite anotar qualquer uso de um tipo:

- em instanciação de objetos:

```
new @Interned MeuObjeto();
```

- em conversão de tipos:

```
minhaString = (@NonNull String) str;
```

- cláusula `implements`:

```
class ListaImutável<T> implements
    @Readonly List<@Readonly T> { ... }
```

- em declarações de exceções

```
void monitorDeTemperatura() throws
    @Critical TemperaturaException { ... }
```

[1] Ver também a JSR 308

Definem um “contrato” que uma classe deve seguir.

```
public interface GrupoDeInterfaces
       extends Interface1, Interface2, Interface3 {

    // declarações de constantes

    // base de logaritmos neperianos
    double E = 2.718282;

    // assinatura de métodos
    void façaAlgo(int i, double x);

    static public métodoEstático(int i) {
       return Math.sqrt(GrupoDeInterfaces.E);
    }

    // método padrão
    default int duplica(String s) {
        return s+s;
    }
}
```

HERANÇA

- Classes definem a estrutura e comportamento de objetos
- Herança é a característica de linguagens OO que permite que novas classes sejam criadas usando classes pré-existentes como base
- A nova classe *herda* os campos e métodos da classe original
- Dizemos que a nova classe é uma *subclasse* da original; e a classe utilizada como base é chamada de *superclasse*.

Vantagens:

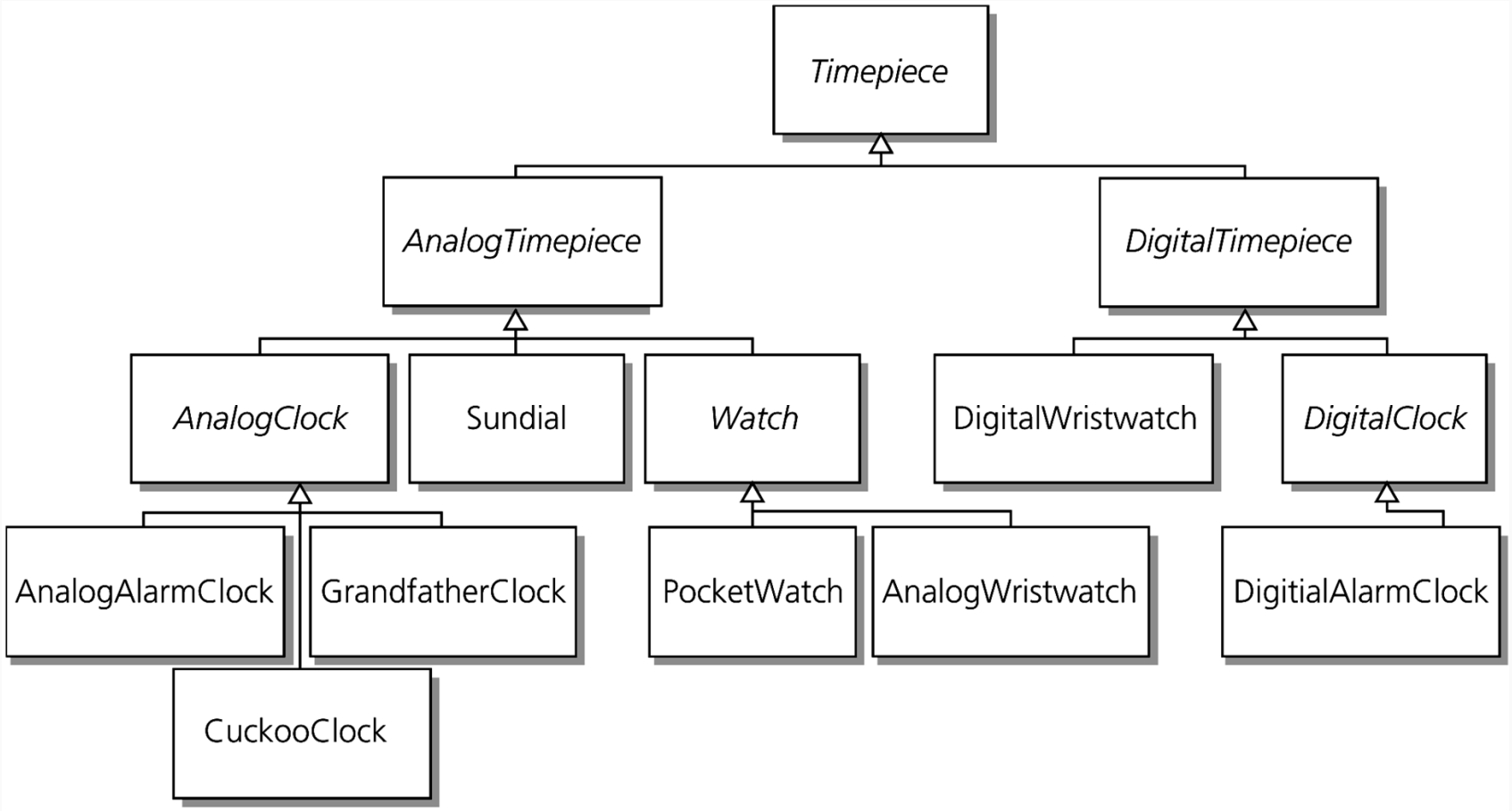
- Melhor modelagem conceitual — hierarquias de especialização são comuns na vida real
- Fatorização — herança permite que propriedades comuns sejam fatorizadas, i.e., definidas apenas uma vez
- Refinamento do projeto e validação — construção de classes com base em outras bem testadas produzirá menos defeitos
- Polimorfismo (mais sobre isso daqui a pouco)

EXEMPLO DE HERANÇA

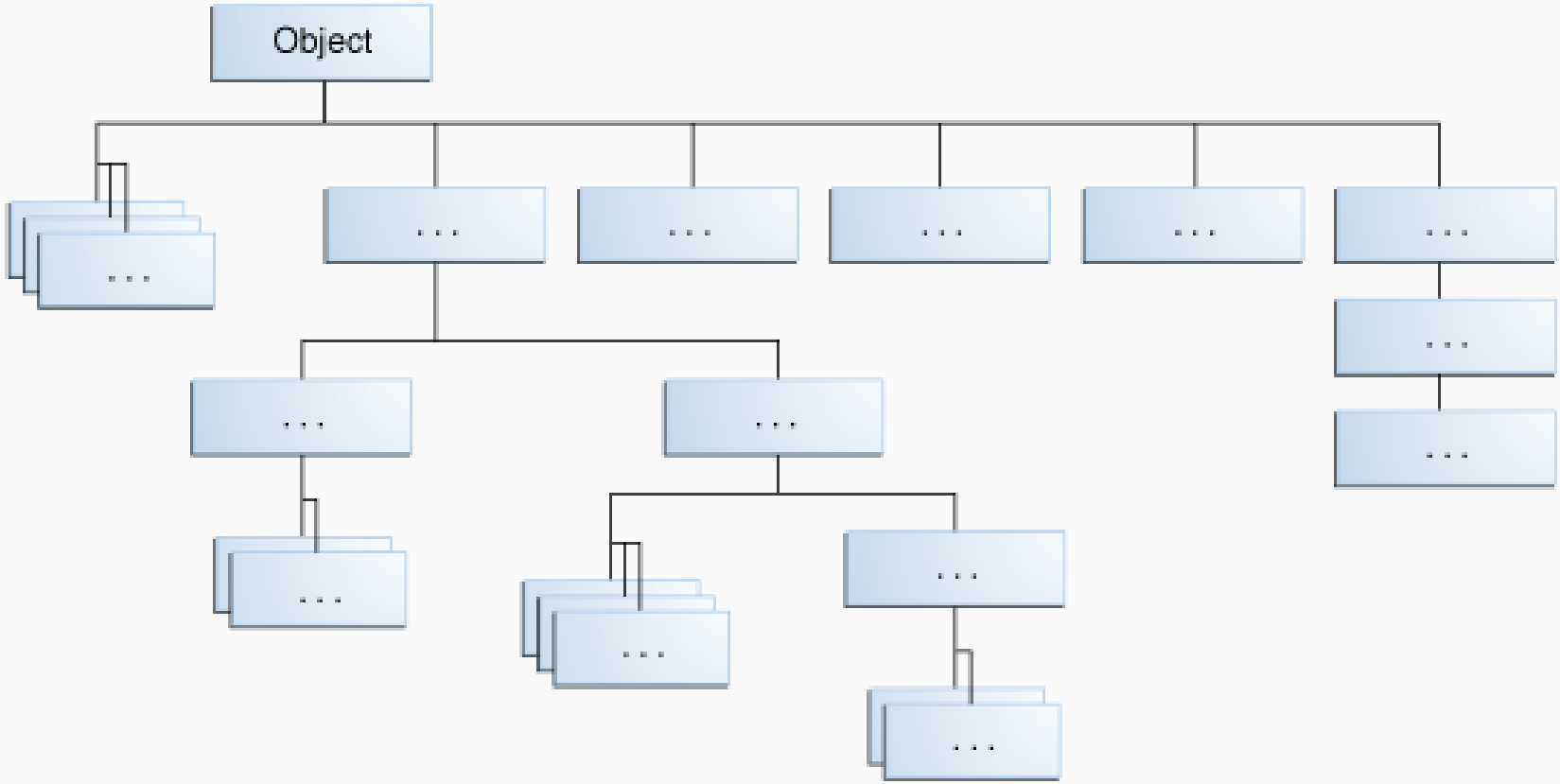
```java
public class Bicycle {

  // a classe Bicycle tem três campos
  public int cadence, gear, speed;

  // the Bicycle class has one constructor
  public Bicycle(int startCadence,
                 int startSpeed,
                 int startGear) {
      gear = startGear;
      cadence = startCadence;
      speed = startSpeed;
  }

  // a classe Bicycle tem quatro métodos
  public void setCadence(int newValue) {
      cadence = newValue;
  }
  public void setGear(int newValue) {
      gear = newValue;
  }
  public void applyBrake(int decrement) {
      speed -= decrement;
  }
  public void speedUp(int increment) {
      speed += increment;
  }

}
```

```java
public class MountainBike extends Bicycle {

  // a subclasse MountainBike adiciona um campo
  public int seatHeight;

  // a subclasse MountainBike tem um construtor
  public MountainBike(int startHeight,
                      int startCadence,
                      int startSpeed,
                      int startGear) {
      super(startCadence, startSpeed, startGear);
      seatHeight = startHeight;
  }

  // a subclasse MountainBike adiciona um método
  public void setHeight(int newValue) {
      seatHeight = newValue;
  }
}
```

EXEMPLO #2


Timepiece
AnalogTimepiece
DigitalTimepiece
AnalogClock
Sundial
Watch
DigitalWristwatch
DigitalClock
AnalogAlarmClock
GrandfatherClock
PocketWatch
AnalogWristwatch
DigitialAlarmClock
CuckooClock

HIERARQUIA DE CLASSES



BIBLIOGRAFIA

- The Java™ Tutorials
  https://docs.oracle.com/javase/tutorial/java/concepts/index.html