

Superclasses e  
subclasses

Sobreposição e  
ocultamento

Palavra-chave  
`protected`

Palavra-chave  
`instanceof`

Construtores em  
subclasses

Palavra-chave `final`

Classe `Object`

Referências

# Herança

prof. Fábio Luiz Usberti

## MC322 - Programação Orientada a Objetos

Instituto de Computação - UNICAMP - 2014



Superclasses e subclasses

Sobreposição e ocultamento

Palavra-chave `protected`

Palavra-chave `instanceof`

Construtores em subclasses

Palavra-chave `final`

Classe `Object`

Referências

- 1 Superclasses e subclasses
- 2 Sobreposição e ocultamento
- 3 Palavra-chave `protected`
- 4 Palavra-chave `instanceof`
- 5 Construtores em subclasses
- 6 Palavra-chave `final`
- 7 Classe `Object`
- 8 Referências

## Superclasses e subclasses

## Sobreposição e ocultamento

Palavra-chave  
`protected`Palavra-chave  
`instanceof`Construtores em  
subclassesPalavra-chave `final`Classe `Object`

## Referências

# Superclasses e subclasses

## Definições e motivações

- Em programação orientada a objetos, herança consiste em uma **forma de reuso de código** onde uma classe é criada já contendo entidades (atributos, métodos e classes internas) de outra classe existente.
- Com herança, é possível **poupar tempo de desenvolvimento e depuração** de novas classes fundadas sobre classes já implementadas e testadas.
- Para utilizar herança, ao invés de declarar uma nova classe completa, o programador indica uma classe existente da qual serão herdadas entidades (atributos, métodos e classes internas).
- A classe existente é denominada **superclasse** enquanto a nova classe é denominada **subclasse**.

## Superclasses e subclasses

## Sobreposição e ocultamento

Palavra-chave  
`protected`Palavra-chave  
`instanceof`Construtores em  
subclassesPalavra-chave `final`Classe `Object`

## Referências

# Superclasses e subclasses

## Definições e motivações

- Em Java, um objeto de uma classe também pode ser objeto de outras classes.
- Um objeto de uma subclasse **também é um objeto de uma superclasse**.
- Uma superclasse pode ter muitas subclasses, portanto o conjunto de objetos de uma superclasse é normalmente maior do que o conjunto de objetos de qualquer de suas subclasses.

## Superclasses e subclasses

## Sobreposição e ocultamento

Palavra-chave `protected`Palavra-chave `instanceof`

## Construtores em subclasses

Palavra-chave `final`Classe `Object`

## Referências

# Superclasses e subclasses

## Definição e motivações

- Uma subclasse eventualmente pode se tornar a superclasse de novas subclasses.
- Uma subclasse pode incluir **métodos e atributos próprios**, além daqueles que foram herdados da superclasse. Por esse motivo, uma subclasse consiste em um caso mais específico da superclasse.
- É comum tratar a herança como uma relação de **especialização** de um grupo de objetos, pois a subclasse exibe comportamentos da superclasse mas modifica esses comportamentos de modo que eles operem de forma apropriada para a subclasse.
- Uma **hierarquia de classes** define os relacionamentos de herança entre as classes.

## Superclasses e subclasses

Sobreposição e ocultamento

Palavra-chave `protected`Palavra-chave `instanceof`

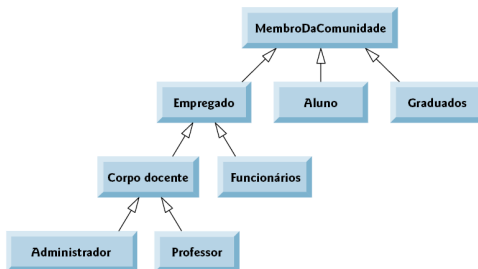
Construtores em subclasses

Palavra-chave `final`Classe `Object`

Referências

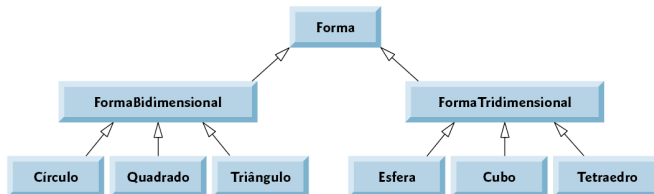
## Superclasses e subclasses

## Hierarquia de heranças em uma universidade



# Superclasses e subclasses

## Hierarquia de heranças em formas geométricas



# Superclasses e subclasses

## Superclasses e subclasses

### Sobreposição e ocultamento

### Palavra-chave `protected`

### Palavra-chave `instanceof`

### Construtores em subclasses

### Palavra-chave `final`

### Classe `Object`

### Referências

## Superclasse e subclasse diretas

- Uma **superclasse direta** `A` consiste em uma classe a partir da qual uma subclasse `B` herda explicitamente. Nesse caso, a classe `B` é **subclasse direta** de `A`.

```
public class B extends A {  
    // B subclasse direta de A; A superclasse direta de B  
}
```

- O relacionamento subclasse/superclasse pode ser definido de forma recursiva.
- Uma classe `C` é subclasse de `A` se uma das seguintes afirmações for verdade:
  - Classe `C` é subclasse direta de `A`
  - Classe `C` é subclasse de `B` que é subclasse direta de `A`.



## Superclasses e subclasses

## Sobreposição e ocultamento

Palavra-chave  
`protected`Palavra-chave  
`instanceof`Construtores em  
subclassesPalavra-chave `final`Classe `Object`

## Referências

# Superclasses e subclasses

## Exemplos de herança:

- Considere as seguintes classes:

```
class Point { int x, y; }  
class ColoredPoint extends Point { int color; }  
class Colored3dPoint extends ColoredPoint { int z; }
```

- A classe `Point` é subclasse direta de `Object`.
- A classe `Object` é superclasse direta de `Point`.
- A classe `ColoredPoint` é subclasse direta de `Point`.
- A classe `Point` é superclasse direta de `ColoredPoint`.
- A classe `Colored3dPoint` é subclasse direta de `ColoredPoint`.
- A classe `ColoredPoint` é superclasse direta de `Colored3dPoint`.

## Superclasses e subclasses

## Sobreposição e ocultamento

Palavra-chave  
`protected`Palavra-chave  
`instanceof`Construtores em  
subclassesPalavra-chave `final`Classe `Object`

## Referências

# Superclasses e subclasses

## Circularidade de Herança

- A seguinte declaração resulta em **erro de compilação** devido à circularidade na herança de classes, ou seja, a classe `Point` é subclasse e superclasse de `Point`.

```
class Point extends Colored3dPoint { int x, y; }  
class ColoredPoint extends Point { int color; }  
final class Colored3dPoint extends ColoredPoint { int z; }
```

# Superclasses e subclasses

## Superclasses e subclasses

### Sobreposição e ocultamento

### Palavra-chave `protected`

### Palavra-chave `instanceof`

### Construtores em subclasses

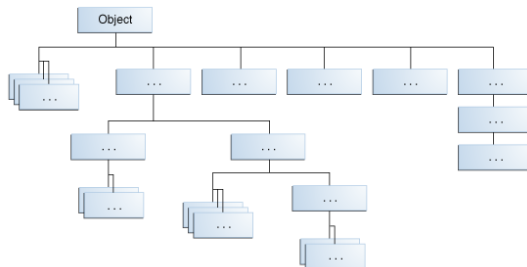
### Palavra-chave `final`

### Classe `Object`

### Referências

## Classe primordial `Object`

- Em Java, a classe `Object` é denominada **classe primordial**, pois não possui superclasse e é superclasse de todas as demais classes.
- Uma hierarquia de classes começa com a classe `Object` (pacote `java.lang`), a partir da qual todas as classes herdam direta ou indiretamente.
- Os métodos da classe `Object` são herdados por todas as classes.



## Superclasses e subclasses

## Sobreposição e ocultamento

Palavra-chave `protected`Palavra-chave `instanceof`

## Construtores em subclasses

Palavra-chave `final`Classe `Object`

## Referências

# Superclasses e subclasses

## Definição e motivações

- É possível distinguir uma relação "tem um" de uma relação "é um" entre objetos.
- A relação **é um** representa herança, pois um objeto de uma subclasse pode ser tratado como um objeto da superclasse. Por exemplo, uma bicicleta elétrica **é uma** bicicleta.
- Por outro lado, a relação **tem um** representa a posse de um objeto por outro objeto. Por exemplo, um carro **tem uma** roda, mas **não é** uma roda.

## Superclasses e subclasses

## Sobreposição e ocultamento

Palavra-chave `protected`Palavra-chave `instanceof`

## Construtores em subclasses

Palavra-chave `final`Classe `Object`

## Referências

# Superclasses e subclasses

## Definição e motivações

Superclasse	Subclasses
Aluno	AlunoDeGraduação, AlunoDePósGraduação
Forma	Círculo, Triângulo, Retângulo, Esfera, Cubo
Financiamento	FinanciamentoDeCarro, FinanciamentoDeCasa
Empregado	CorpoDocente, Funcionário
ContaBancária	ContaCorrente, ContaPoupança

Exemplo de possíveis relações "é um" entre classes.

# Sobreposição e ocultamento

## Sobreposição de métodos de instância

- Uma subclasse pode customizar **métodos de instância** que ela herda da superclasse, processo denominado **sobreposição** (*overriding*).
- Em uma sobreposição de métodos de instância, a subclasse redefine um método da superclasse com uma implementação própria.
- **Atenção:** ao usar uma assinatura incompatível com o método da superclasse, não há mais sobreposição e sim uma sobrecarga.

# Sobreposição e ocultamento

## Ocultamento de métodos de classes

- Se uma subclasse redefine um método estático da superclasse, então esse processo é denominado **ocultamento** (*hiding*).
- A diferença entre sobreposição e ocultamento de métodos possui implicações importantes:
  - O método de instância executado quando há uma sobreposição será o método declarado na classe mais especializada (subclasse) referente ao objeto.
  - O método estático executado quando há um ocultamento depende exclusivamente de onde partiu a chamada, seja ela da superclasse ou da subclasse.

# Exemplo de sobreposição e ocultamento

```
// Animal.java
public class Animal {
    public static void testClassMethod() {
        System.out.println("O método estático na classe Animal");
    }
    public void testInstanceMethod() {
        System.out.println("O método de instância na classe Animal");
    }
}
```

```
// Cat.java
public class Cat extends Animal {
    public static void testClassMethod() {
        System.out.println("O método estático na classe Cat");
    }
    public void testInstanceMethod() {
        System.out.println("O método de instância na classe Cat");
    }

    public static void main(String[] args) {
        Cat myCat = new Cat();
        Animal myAnimal = myCat;
        Animal.testClassMethod();
        myAnimal.testInstanceMethod();
    }
}
```



Superclasses e  
subclasses

Sobreposição e  
ocultamento

Palavra-chave  
`protected`

Palavra-chave  
`instanceof`

Construtores em  
subclasses

Palavra-chave `final`

Classe `Object`

Referências

## Exemplo de sobreposição e ocultamento: saída

O método estático na classe `Animal`  
O método de instância na classe `Cat`

# Sobreposição e ocultamento

## Ocultamento de atributos

- Ocultamento também ocorre com atributos, **estáticos ou de instância**, quando a subclasse declara um atributo com a mesma identificação de um atributo da superclasse.
- Ocultar um atributo é **diferente de sombrear um atributo**, pois o primeiro ocorre em uma relação subclasse/superclasse, enquanto o segundo se verifica entre um atributo e um parâmetro ou variável local da mesma classe.
- Ocultamento de atributos é uma **má prática de programação**, pois diminui a legibilidade do código.

# Palavra-chave `protected`

## Visibilidade para subclasses

- O modificador de acesso `protected` oferece uma visibilidade intermediária entre `public` e `private`.
- As entidades (métodos, atributos e classes internas) de uma superclasse com acesso `protected` podem ser acessados pela **superclasse, subclasses e classes do mesmo pacote** que a superclasse.
- Uma subclasse **mantém os modificadores de acesso** das entidades da superclasse.
- Atributos `protected` tem um acesso mais eficiente nas subclasses por não precisarem de métodos acessores. No entanto, na maior parte dos casos, é recomendável o uso de atributos `private`, promovendo as **boas práticas de encapsulamento** que tornam o código mais fácil de manter, modificar e depurar.

Superclasses e subclasses

Sobreposição e ocultamento

Palavra-chave `protected`

Palavra-chave `instanceof`

Construtores em subclasses

Palavra-chave `final`

Classe `Object`

Referências

## Palavra-chave `protected`

### Desvantagens do acesso `protected` para atributos

- 1 Um objeto da subclasse pode atribuir valores diretamente aos atributos herdados, dispensando a validação dos métodos acessores e potencialmente levando o objeto para um estado inconsistente.
- 2 A alteração de algum atributo `protected` de uma superclasse potencialmente pode requerer a modificação de todas as subclasses associadas (diz-se nesse caso que o código é **frágil**). A localidade dos efeitos de mudanças em códigos consistem em uma boa prática de programação.
- 3 Atributos `protected` de uma classe também são visíveis para outras classes do mesmo pacote, e nem sempre isso é desejável.

Superclasses e  
subclasses

Sobreposição e  
ocultamento

Palavra-chave  
`protected`

Palavra-chave  
`instanceof`

Construtores em  
subclasses

Palavra-chave `final`

Classe `Object`

Referências

## Palavra-chave `protected`

### Redução de visibilidade

- Em Java não é possível reduzir a visibilidade de um método em uma sobreposição. Ou seja, a sobreposição de um método `public` da superclasse por um método da subclasse com um acesso diferente de `public` consiste em um **erro de compilação**.
- A redução da visibilidade de um método sobreposto na subclasse implicaria na quebra da relação **é um** entre a superclasse-subclasse. Por exemplo, objetos da subclasse não poderiam responder às mesmas chamadas de métodos da superclasse.

Superclasses e subclasses

Sobreposição e ocultamento

Palavra-chave `protected`

Palavra-chave `instanceof`

Construtores em subclasses

Palavra-chave `final`

Classe `Object`

Referências

# Palavra-chave `protected`

## Acesso a métodos da superclasse

- Métodos `public` e `protected` de uma superclasse podem ser **acessados diretamente** pelo nome em uma subclasse.
- Quando a subclasse sobrepõe um método de instância da superclasse, o método original da superclasse pode ser acessado pela subclasse com a palavra-chave `super`.

```
super.overridenMethod();
```

Superclasses e  
subclasses

Sobreposição e  
ocultamento

Palavra-chave  
`protected`

Palavra-chave  
`instanceof`

Construtores em  
subclasses

Palavra-chave `final`

Classe `Object`

Referências

# Palavra-chave `instanceof`

## Comparação de tipos

- O operador `instanceof` compara um objeto com um determinado tipo, e retorna `true` se o objeto for uma instânciação do tipo especificado.
- O operador `instanceof` pode ser utilizado para verificar se um objeto é uma instância de uma classe, subclasse ou também se ele implementa uma interface.
- O programa a seguir define uma classe `Parent`, uma interface `MyInterface` e uma classe `Child` que herda de `Parent` e implementa `MyInterface`.

# Exemplo de uso do operador `instanceof`

Superclasses e subclasses

Sobreposição e ocultamento

Palavra-chave `protected`

Palavra-chave `instanceof`

Construtores em subclasses

Palavra-chave `final`

Classe `Object`

Referências

```
class Parent {}  
class Child extends Parent implements MyInterface {}  
interface MyInterface {}  
  
class InstanceofDemo {  
    public static void main(String[] args) {  
  
        Parent obj1 = new Parent();  
        Parent obj2 = new Child();  
        Child obj3 = null;  
  
        System.out.println("obj1 instanceof Parent: " + (obj1 instanceof Parent));  
        System.out.println("obj1 instanceof Child: " + (obj1 instanceof Child));  
        System.out.println("obj1 instanceof MyInterface: " + (obj1 instanceof MyInterface));  
        System.out.println();  
        System.out.println("obj2 instanceof Parent: " + (obj2 instanceof Parent));  
        System.out.println("obj2 instanceof Child: " + (obj2 instanceof Child));  
        System.out.println("obj2 instanceof MyInterface: " + (obj2 instanceof MyInterface));  
        System.out.println();  
        System.out.println("obj3 instanceof Parent: " + (obj3 instanceof Parent));  
        System.out.println("obj3 instanceof Child: " + (obj3 instanceof Child));  
        System.out.println("obj3 instanceof MyInterface: " + (obj3 instanceof MyInterface));  
    }  
}
```



Superclasses e  
subclasses

Sobreposição e  
ocultamento

Palavra-chave  
`protected`

Palavra-chave  
`instanceof`

Construtores em  
subclasses

Palavra-chave `final`

Classe `Object`

Referências

## Exemplo: saída

```
obj1 instanceof Parent: true  
obj1 instanceof Child: false  
obj1 instanceof MyInterface: false
```

```
obj2 instanceof Parent: true  
obj2 instanceof Child: true  
obj2 instanceof MyInterface: true
```

```
obj3 instanceof Parent: false  
obj3 instanceof Child: false  
obj3 instanceof MyInterface: false
```

Superclasses e subclasses

Sobreposição e ocultamento

Palavra-chave `protected`

Palavra-chave `instanceof`

Construtores em subclasses

Palavra-chave `final`

Classe `Object`

Referências

# Construtores em subclasses

## Chamada de construtores da superclasse

- **Construtores não são herdados** pelas subclasses, no entanto, os construtores da superclasse ainda são acessíveis pela subclasse.
- A primeira tarefa de um construtor de uma subclasse é **chamar o construtor da superclasse**, explícita ou implicitamente, para assegurar que as variáveis de instância herdadas serão inicializadas corretamente.
- Se o construtor da subclasse não chama o construtor da superclasse explicitamente, então o compilador gera uma instrução que chama o construtor default ou o construtor sem argumento da superclasse. Se não houver tal construtor na superclasse ocorre um **erro de compilação**.

Superclasses e subclasses

Sobreposição e ocultamento

Palavra-chave `protected`

Palavra-chave `instanceof`

Construtores em subclasses

Palavra-chave `final`

Classe `Object`

Referências

# Construtores em subclasses

## Chamada de construtores da superclasse

- A chamada explícita de um construtor de superclasse deve ser a primeira instrução a ser chamada no construtor da subclasse e sua sintaxe é fornecida a seguir:

```
super(<argumentos>);
```

- Quando uma superclasse contém um construtor sem argumento, é possível chamá-lo explicitamente com o comando `super()` no método construtor da subclasse, porém isso não é necessário.

# Palavra-chave `final`

## Classes e métodos finais

- A palavra-chave `final`, como modificador de um método, indica que esse método **não pode ser sobreposto** em subclasses.
- Um método deve ser declarado final quando sua **implementação não deve mudar**.

```
class ChessAlgorithm {  
    enum ChessPlayer { WHITE, BLACK }  
    final ChessPlayer getFirstPlayer() {  
        return ChessPlayer.WHITE;  
    }  
}
```

- Métodos chamados por construtores são bons candidatos para serem declarados finais, pois a subclasse pode redefinir esse método com resultados indesejados.
- Também é possível declarar uma classe como final. Isso implica que essa **classe não pode gerar especializações**.
- Declarar uma classe final aumenta o **controle sobre a classe**, pois impede que subclasses introduzam comportamentos anômalos.

Superclasses e subclasses

Sobreposição e ocultamento

Palavra-chave `protected`

Palavra-chave `instanceof`

Construtores em subclasses

Palavra-chave `final`

Classe `Object`

Referências

## Exemplo de herança

```
// Employee.java
// classe Employee representa um funcionário
public class Employee {

    private String firstName;
    private String lastName;
    private String cpf;

    // construtor
    public Employee(String first, String last, String argCpf) {
        firstName = first ;
        lastName = last;
        cpf = argCpf;
    } // fim construtor

    // Métodos acessores

    public void setFirstName(String first) {
        firstName = first ;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setLastName(String last) {
        lastName = last;
    }

    public String getLastName() {
        return lastName;
    }

    /* continua na próxima página */
```

Superclasses e subclasses

Sobreposição e ocultamento

Palavra-chave `protected`

Palavra-chave `instanceof`

Construtores em subclasses

Palavra-chave `final`

Classe `Object`

Referências

## Exemplo de herança

```
/* continua da página anterior */

public void setCpf(String argCpf) {
    // TODO: incluir método de validação de CPF
    cpf = argCpf;
}

public String getCpf() {
    return cpf;
}

// método toString retorna uma string representando o objeto Employee
@Override
public String toString() {
    return String.format("%s: %s %s\n%s: %s", "Nome", getFirstName(),
        getLastName(), "CPF", getCpf());
} // fim método toString

public static void main(String args[]) {

    // instancia um objeto Employee
    Employee employee1 = new Employee("Fulano", "Silva", "123.456.789-00");

    System.out.printf("\n%s:\n%s\n", "employee1 - funcionário", employee1.toString());
} // fim main

} // fim classe Employee
```

Superclasses e  
subclasses

Sobreposição e  
ocultamento

Palavra-chave  
`protected`

Palavra-chave  
`instanceof`

Construtores em  
subclasses

Palavra-chave `final`

Classe `Object`

Referências

## Exemplo de herança: saída

```
employee1 - funcionário:  
Nome: Fulano Silva  
CPF: 123.456.789-00
```

Superclasses e subclasses

Sobreposição e ocultamento

Palavra-chave `protected`

Palavra-chave `instanceof`

Construtores em subclasses

Palavra-chave `final`

Classe `Object`

Referências

## Exemplo de herança

```
// CommissionEmployee.java
// classe CommissionEmployee representa um funcionário comissionado
public class CommissionEmployee extends Employee {
    private double grossSales; // total de vendas
    private double commissionRate; // taxa de comissão

    // construtor
    public CommissionEmployee(String first, String last, String cpf,
        double sales, double rate) {
        super(first, last, cpf);
        setGrossSales(sales); // valida e armazena o total de vendas
        setCommissionRate(rate); // valida e armazena a taxa de comissão
    } // fim construtor

    // Métodos acessores

    public void setGrossSales(double sales) {
        // total de vendas deve ser um valor não — negativo
        grossSales = (sales < 0.0) ? 0.0 : sales;
    }

    public double getGrossSales() {
        return grossSales;
    }

    public void setCommissionRate(double rate) {
        // taxa de comissão deve ser um valor no intervalo aberto (0,1)
        commissionRate = (rate > 0.0 && rate < 1.0) ? rate : 0.0;
    }

    public double getCommissionRate() {
        return commissionRate;
    }

    /* continua na próxima página */
```



Superclasses e subclasses

Sobreposição e ocultamento

Palavra-chave `protected`

Palavra-chave `instanceof`

Construtores em subclasses

Palavra-chave `final`

Classe `Object`

Referências

## Exemplo de herança

```
/* continua da página anterior */

// método earnings retorna os vencimentos do funcionário
public double earnings() {
    return getCommissionRate() * getGrossSales();
} // fim método earnings

// método toString retorna uma string representando o objeto
// CommissionEmployee
@Override
public String toString() {
    return String.format("%s \n%s: %.2f\n%s: %.2f", super.toString(),
        "Total de vendas", getGrossSales(), "Taxa de comissão",
        getCommissionRate());
} // fim método toString

public static void main(String args[]) {

    // instancia um objeto CommissionEmployee
    Employee employee2 = new CommissionEmployee("Beltrano", "Souza", "123.456.790-01", 5000, .04);

    System.out.printf("\n%s:\n%s\n", "employee2 - funcionário comissionado", employee2.toString());
} // fim main

} // fim classe CommissionEmployee
```

Superclasses e  
subclasses

Sobreposição e  
ocultamento

Palavra-chave  
`protected`

Palavra-chave  
`instanceof`

Construtores em  
subclasses

Palavra-chave `final`

Classe `Object`

Referências

## Exemplo de herança: saída

```
employee2 - funcionário comissionado:  
Nome: Beltrano Souza  
CPF: 123.456.790-01  
Total de vendas: 5000.00  
Taxa de comissão: 0.04
```

Superclasses e subclasses

Sobreposição e ocultamento

Palavra-chave `protected`

Palavra-chave `instanceof`

Construtores em subclasses

Palavra-chave `final`

Classe `Object`

Referências

## Exemplo de herança

```
// BasePlusCommissionEmployee.java
// classe BasePlusCommissionEmployee representa um funcionário comissionado com um salário base
public class BasePlusCommissionEmployee extends CommissionEmployee {
    private double baseSalary; // salário base semanal

    // construtor
    public BasePlusCommissionEmployee(String first, String last, String cpf,
        double sales, double rate, double salary) {
        super(first, last, cpf, sales, rate);
        setBaseSalary(salary); // valida e armazena o valor do salário base
    } // fim construtor

    // Métodos acessores

    public void setBaseSalary(double salary) {
        // o salário base deve ser um valor não—negativo
        baseSalary = (salary < 0.0) ? 0.0 : salary;
    }

    public double getBaseSalary() {
        return baseSalary;
    }

    /* continua na próxima página */
```

## Exemplo de herança

Superclasses e subclasses

Sobreposição e ocultamento

Palavra-chave `protected`

Palavra-chave `instanceof`

Construtores em subclasses

Palavra-chave `final`

Classe `Object`

Referências

```

/* continua da página anterior */

// método earnings retorna os vencimentos do funcionário
@Override
public double earnings() {
    // vencimentos do funcionário consistem no salário base mais comissão
    return getBaseSalary() + super.earnings();
} // fim método earnings

// método toString retorna uma string representando o objeto
// BasePlusCommissionEmployee
@Override
public String toString() {
    return String.format("%s\n%s: %.2f", super.toString(), "salário base",
        getBaseSalary());
} // fim método toString

public static void main(String args[]) {

    // instancia um objeto BasePlusCommissionEmployee
    Employee employee3 = new BasePlusCommissionEmployee("Ciclano",
        "Santos", "123.456.791-02", 5000, .04, 300);

    System.out.printf("\n%s:\n%s\n",
        "employee3 - funcionário comissionado com salário base",
        employee3.toString());

} // fim main
} // fim classe BasePlusCommissionEmployee

```

Superclasses e  
subclasses

Sobreposição e  
ocultamento

Palavra-chave  
`protected`

Palavra-chave  
`instanceof`

Construtores em  
subclasses

Palavra-chave *final*

Classe `Object`

Referências

## Exemplo de herança: saída

employee3 - funcionário comissionado com salário base:

Nome: Ciclano Santos

CPF: 123.456.791-02

Total de vendas: 5000.00

Taxa de comissão: 0.04

salário base: 300.00

BasePlusCommissionEmployee

Superclasses e subclasses

Sobreposição e ocultamento

Palavra-chave `protected`

Palavra-chave `instanceof`

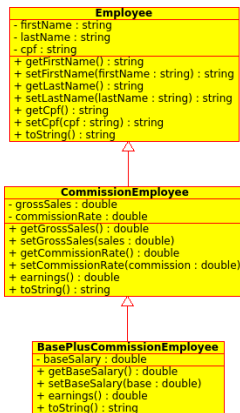
Construtores em subclasses

Palavra-chave `final`

Classe `Object`

Referências

# Representação de Herança em UML



Superclasses e  
subclasses

Sobreposição e  
ocultamento

Palavra-chave  
`protected`

Palavra-chave  
`instanceof`

Construtores em  
subclasses

Palavra-chave `final`

Classe `Object`

Referências

# Classe `Object`

## Raiz da hierarquia de classes

- A classe `Object` encontra-se na **raiz da hierarquia de classes**, portanto é superclasse de todas as classes, exceto dela mesma, dado que a classe `Object` não é subclasse de ninguém.
- Ao criar uma nova classe, se não for especificado uma superclasse, implicitamente a nova classe herdará da classe `Object`, o que equivale a incluir `extends Object` na declaração da nova classe.

# Classe `Object`

## Métodos da classe `Object`

- `clone`: Esse método realiza uma cópia do objeto. A implementação padrão realiza uma **cópia superficial**, ou seja, os valores das variáveis de instância são copiados para o novo objeto, porém, para atributos referenciados, somente a referência é copiada. Normalmente, esse método deve ser sobreposto por uma implementação que faça uma **cópia em profundidade** que cria um novo objeto para cada variável de instância referenciada. Dependendo da classe, a implementação do método `clone` não é trivial.
- `equals`: Esse método compara se dois objetos são iguais. A **implementação padrão compara as referências**, ou seja, se os dois objetos são, na verdade, o mesmo objeto. Normalmente, esse método deve ser sobreposto por um método que realiza a comparação do conteúdo dos objetos.



# Classe `Object`

## Métodos da classe `Object`

- `getClass`: Todo objeto em Java tem acesso à própria classe a que pertence (propriedade conhecida como **reflexão**). Esse método retorna um objeto da classe `Class` que contém informações sobre a classe do objeto original, como nome da classe, atributos e métodos.
- `hashCode`: são valores `int` que representam uma chave do objeto, utilizado em operações de recuperação, armazenamento e comparação de objetos.
- `toString`: Esse método retorna uma representação `String` de um objeto. A implementação padrão desse método retorna o nome do pacote e a identificação da classe do objeto seguido por uma representação hexadecimal do valor retornado pelo método `hashCode()`.

```
// retorno da implementação padrão toString()
getClass().getName() + '@' + Integer.toHexString(hashCode())
```

Superclasses e  
subclasses

Sobreposição e  
ocultamento

Palavra-chave  
`protected`

Palavra-chave  
`instanceof`

Construtores em  
subclasses

Palavra-chave `final`

Classe `Object`

Referências

## Referências

- 1 Java: Como Programar, Paul Deitel & Heivey Deitel; Pearson; 7a. Ed. (no. chamada IMECC – 05.133 D368j)
- 2 Data Structures and Algorithms with Object Oriented Design Patterns in Java, Bruno Preiss;  
(<http://www.brpreiss.com/books/opus6/>)
- 3 The Java Tutorials (Oracle)  
(<http://docs.oracle.com/javase/tutorial/>)
- 4 Guia do Usuário UML, Grady Booch et. al.; Campus(1999)
- 5 Java Pocket Guide - Robert Liguori & Patricia Liguori; O'Reilley, 2008.