

# Upcasting e Downcasting

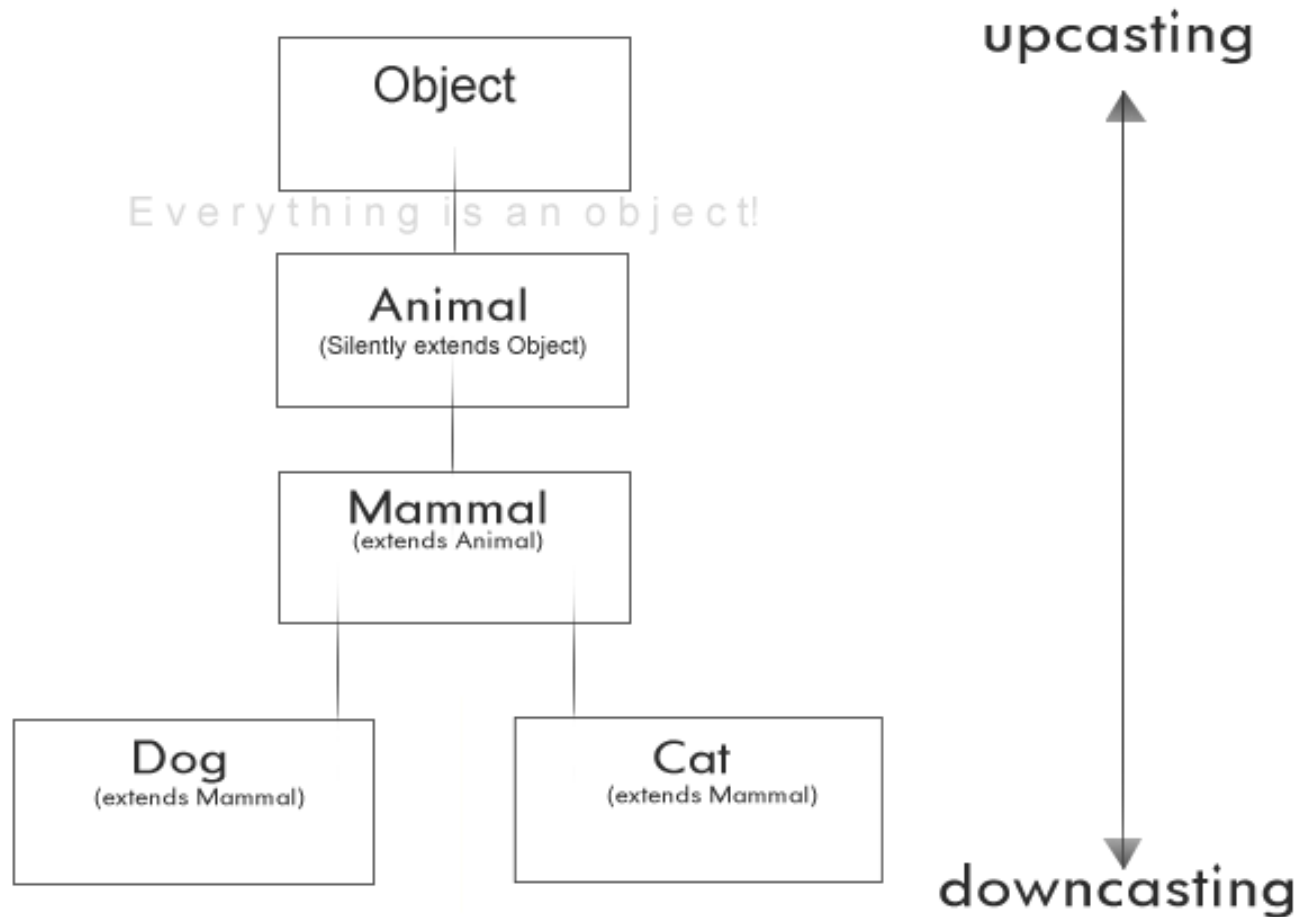
---

## Orientação a Objetos - DCC025

Prof. Edmar Welington Oliveira  
oliveira.edmar@ufjf.edu.br

Universidade Federal de Juiz de Fora - UFJF  
Departamento de Ciência da Computação - DCC

# Herança



# Herança

```
class Animal {  
    int health = 100;  
}  
  
class Mammal extends Animal { }  
  
class Cat extends Mammal { }  
  
class Dog extends Mammal { }  
  
public class Test {  
    public static void main(String[] args) {  
        Cat c = new Cat();  
        System.out.println(c.health);  
        Dog d = new Dog();  
        System.out.println(d.health);  
    }  
}
```

# Exemplo

É preciso compreender que por casting não estamos alterando o objeto em si, mas apenas rotulando-o de forma diferente

```
Cat c = new Cat();  
System.out.println(c);  
Mammal m = c; // upcasting  
System.out.println(m);
```

```
/*  
This printed:  
Cat@a90653  
Cat@a90653  
*/
```

Upcasting é feito automaticamente em Java

O que isso quer dizer?

Exemplo: Se criarmos um gato e realizarmos um upcast para Animal, o objeto gato não deixará de ser um gato. Ainda é um gato, mas é **apenas tratado como qualquer outro animal**. As suas propriedades de gato estarão escondidas até que seja realizado um downcast para gato de novo.

# Exemplo

```
Cat c = new Cat();  
System.out.println(c);  
Mammal m = c; // upcasting  
System.out.println(m);  
  
/*  
This printed:  
    Cat@a90653  
    Cat@a90653  
*/
```

Como pode ser visto, o gato é o mesmo após realizarmos o upcasting. Ele não foi alterado para Mamífero. Ele apenas foi rotulado de forma diferente

Porque isso é permitido?  
Simples: gato É UM mamífero

# Exemplo

## ■ Execute

### ■ Crie uma classe Gato

- Crie um atributo privado `miado(String)`
- Crie o construtor sem argumentos da classe, fazendo `miado = "Miau"`
- Crie métodos `get` e `set`

### ■ Crie uma classe Cachorro

- Crie um atributo privado `latido(String)`
- Crie um construtor sem argumentos da classe, fazendo `latido = "Au"`
- Crie métodos `get` e `set`

# Classes

```
3 public class Gato extends Mamifero{
4
5     protected String miado;
6
7     public Gato(){
8         this.miado = "Miau";
9     }
10
11     public String getMiado() {
12         return miado;
13     }
14
15     public void setMiado(String miado) {
16         this.miado = miado;
17     }
18
19
20 }
```

```
3 public class Cachorro extends Mamifero{
4
5     protected String latido;
6
7     public Cachorro(){
8         this.latido = "AuAu";
9     }
10
11     public String getLatido() {
12         return latido;
13     }
14
15     public void setLatido(String latido) {
16         this.latido = latido;
17     }
18
19 }
```

# Exemplo

## ■ Execute

### ■ Crie uma classe Principal

- Crie o método main()
- Instancie um objeto de Gato
- Faça o objeto ser atribuído a uma variável Mamífero

```
Gato gato = new gato();
```

```
Mamifero m = c;
```

- Tente, com esta variável "m", acessar métodos de Gato. Conseguiu?
- Altere os atributos de todas as classes para protegido.
- Tente acessar os atributos de Gato. Conseguiu?
- Se não, explique porque.



# Classes

```
3 public class Principal {  
4  
5     public static void main(String args[]) {  
6  
7         Gato gato1 = new Gato();  
8         Mamifero m = gato1;  
9  
10        m.  
11    }  
12 }  
13
```

- ◆ idade : int - Mamifero
- equals(Object obj) : boolean - Object
- getClass() : Class<?> - Object
- getIdade() : int - Mamifero
- hashCode() : int - Object
- notify() : void - Object
- notifyAll() : void - Object
- setIdade(int idade) : void - Mamifero
- toString() : String - Object
- wait() : void - Object
- wait(long timeout) : void - Object
- wait(long timeout, int nanos) : void - Object

Press 'Ctrl+Space' to show Template Proposals

# Problema

Não podemos realizar casting entre Cachorro e Gato, mesmo os dois sendo mamíferos



# Upcasting

- Upcasting
  - Como comentado, o upcasting é feito automaticamente
  - Embora não haja necessidade, o programador pode fazê-lo manualmente

```
Mammal m = (Mammal)new Cat();
```

```
Mammal m = new Cat();
```

# Voltando ao Exemplo

## ■ Exemplo

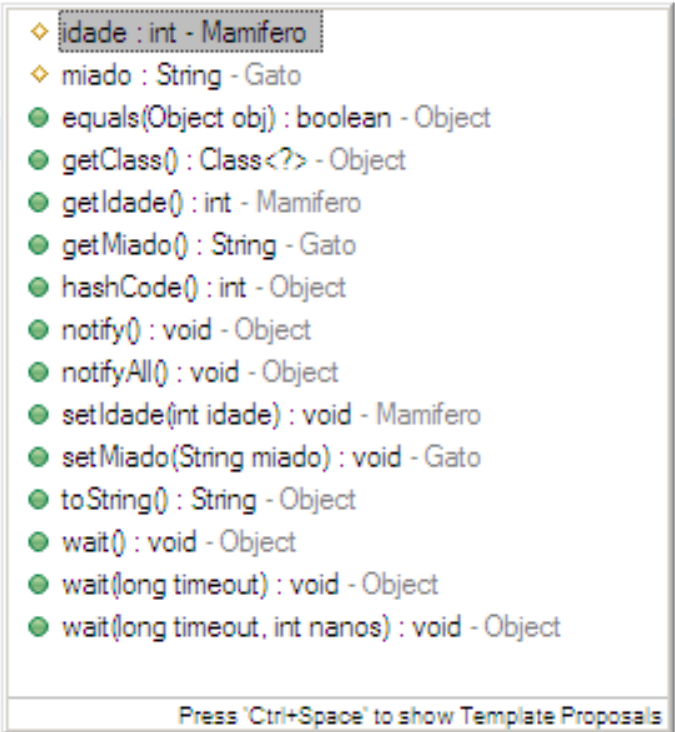
- Conforme foi visto, não se pode acessar nada de Gato usando "m"
- É preciso, então, realizar um downcasting para Gato
- Downcasting é sempre realizado manualmente pelo programador
  - Porque deve ser realizado manualmente?

```
Cat c1 = new Cat();  
Animal a = c1;           //automatic upcasting to Animal  
Cat c2 = (Cat) a;        //manual downcasting back to a Cat
```

Exemplo de upcasting e downcasting

# Downcasting

```
3 public class Principal {
4
5     public static void main(String args[]){
6
7         Gato gato1 = new Gato();
8         Mamifero m = gato1;
9
10        Gato gato2 = (Gato)m;
11        gato2.
12    }
13 }
14
```



The image shows an IntelliJ IDEA code editor with a Java file named Principal.java. The code demonstrates downcasting from Mamifero to Gato. On line 11, the text 'gato2.' is entered, triggering an autocomplete popup. The popup lists various methods and fields available to the Gato object. The first two items, 'idade : int - Mamifero' and 'miado : String - Gato', are highlighted with a yellow background. The rest of the list includes methods like equals, getClass, getIdade, getMiado, hashCode, notify, notifyAll, setIdade, setMiado, toString, wait, and wait with timeout parameters. At the bottom of the popup, it says 'Press 'Ctrl+Space' to show Template Proposals'.

- ◆ idade : int - Mamifero
- ◆ miado : String - Gato
- equals(Object obj) : boolean - Object
- getClass() : Class<?> - Object
- getIdade() : int - Mamifero
- getMiado() : String - Gato
- hashCode() : int - Object
- notify() : void - Object
- notifyAll() : void - Object
- setIdade(int idade) : void - Mamifero
- setMiado(String miado) : void - Gato
- toString() : String - Object
- wait() : void - Object
- wait(long timeout) : void - Object
- wait(long timeout, int nanos) : void - Object

Press 'Ctrl+Space' to show Template Proposals

# Downcasting

- Respondendo a pergunta

- Porque, ao contrário de upcasting, downcasting deve ser manual?

Pode-se perceber que **upcasting nunca vai falhar**, uma vez que estamos inserindo em uma variável de superclasse um objeto de subclasse. O conceito de herança evita erros neste tipo de construção, já que um objeto de subclasse É UM objeto de superclasse.

No exemplo: um gato É UM mamífero

# Downcasting

- Continuando

- Agora imagine que tenhamos um conjunto de animais (não sabemos quais). Imagine, ainda, que queremos fazer um downcasting desses animais para gatos. Neste caso, pode acontecer de alguns desses animais serem, na verdade, cachorros. Logo, estaríamos tentando fazer downcasting entre gatos e cachorro - já vimos que isso não funciona.
- Um exceção do tipo `ClassCastException` é gerada.

# Downcasting

```
Cat c1 = new Cat();  
Animal a = c1;           //upcasting to Animal  
if(a instanceof Cat){ // testing if the Animal is a Cat  
    System.out.println("It's a Cat! Now i can safely downcast it to a Cat, wi  
    Cat c2 = (Cat)a;  
}
```

Uma forma interessante para evitarmos exceções consiste em verificar se um objeto é instância de uma certa classe. Usamos a expressão **instanceof** para isto.

Se estivéssemos tratando de um conjunto de animais, bastaria usar uma estrutura de Loop (um for, por exemplo) para realizarmos um teste no conjunto de animais, buscando verificar quais deles são instância da classe Gato.



# Problema

```
3 public class Principal {  
4  
5     public static void main(String args[]) {  
6  
7         Gato gato1 = new Gato();  
8         Mamifero m1 = gato1;  
9  
10        Mamifero m = new Mamifero();  
11        Gato gato = (Gato)m;  
12    }  
13 }  
14
```

Isso não gera problema

Isso gera problema

Problems @ Javadoc Declaration Console X Progress

<terminated> Principal (6) [Java Application] C:\Program Files\Java\jdk1.6.0\_21\jre\bin\javaw.exe (17/06/2011 16:27:07)

Exception in thread "main" java.lang.ClassCastException: updo.Mamifero cannot be cast to updo.Gato  
at updo.Principal.main(Principal.java:11)

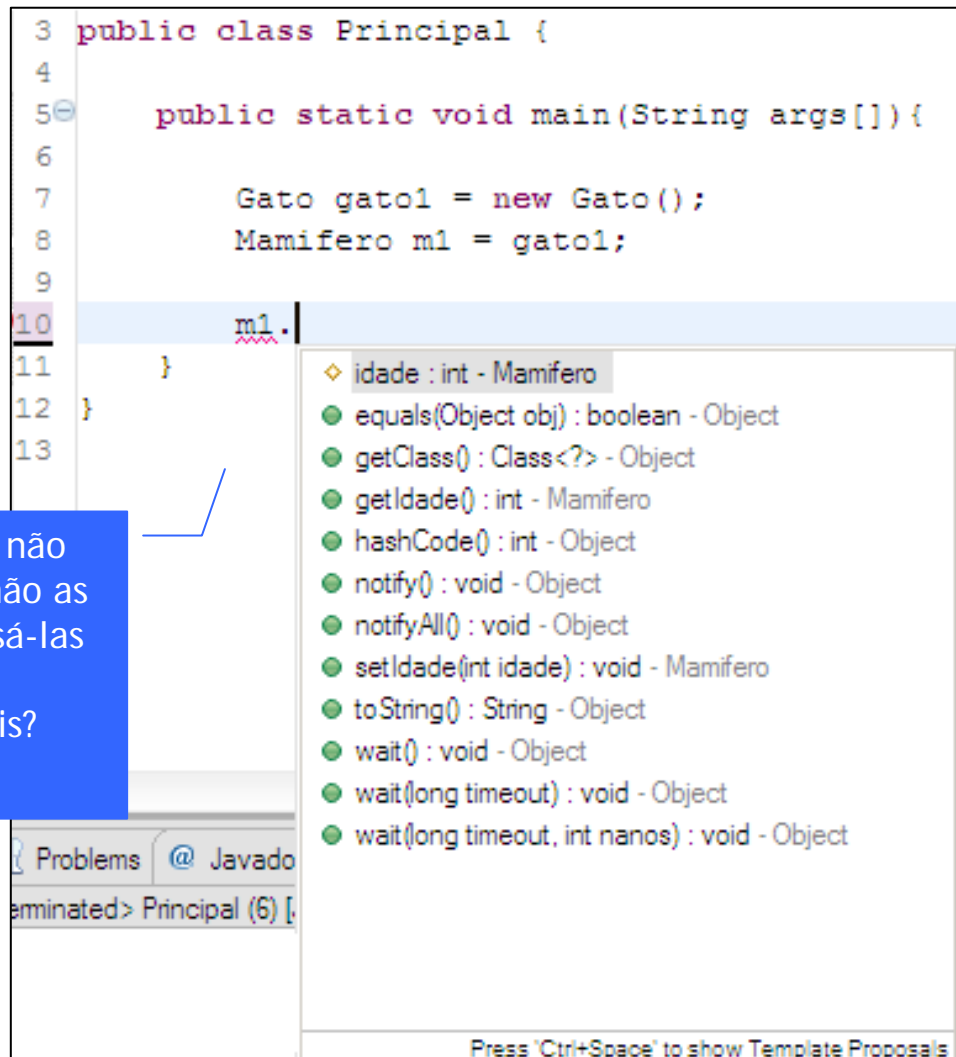
# Upcasting

- Fato

- Se realizarmos um upcast de um objeto, ele perderá todas as suas propriedades particulares. Ex: Na classe Gato, havia um atributo (String) miado. Na classe Mamifero, havia um atributo (int) idade.
- Se realizarmos um upcast de gato para animal, não veremos mais o atributo miado de gato. Veja isso no código abaixo

# Upcasting

```
3 public class Principal {
4
5     public static void main(String args[]){
6
7         Gato gato1 = new Gato();
8         Mamifero m1 = gato1;
9
10        m1.
11    }
12 }
13
```



- ◆ idade : int - Mamifero
- equals(Object obj) : boolean - Object
- getClass() : Class<?> - Object
- getIdade() : int - Mamifero
- hashCode() : int - Object
- notify() : void - Object
- notifyAll() : void - Object
- setIdade(int idade) : void - Mamifero
- toString() : String - Object
- wait() : void - Object
- wait(long timeout) : void - Object
- wait(long timeout, int nanos) : void - Object

Problems @ Javado  
terminated> Principal (6) [

Press 'Ctrl+Space' to show Template Proposals

Observe que, após realizar o upcasting, m1 não pode mais ver as propriedades de gato. Ele não as perde, mas apenas se torna incapaz de acessá-las

Quando eles ficarão novamente disponíveis?  
Quando realizarmos um downcast

# Upcasting e Downcasting

---

## Orientação a Objetos - DCC025

Prof. Edmar Welington Oliveira  
[oliveira.edmar@ufjf.edu.br](mailto:oliveira.edmar@ufjf.edu.br)

Universidade Federal de Juiz de Fora - UFJF  
Departamento de Ciência da Computação - DCC