

PROGRAMAÇÃO ORIENTADA PELOS OBJECTOS

Extensibilidade e a classe Object

2

A classe Object

Já reparou bem na lista de membros que o Eclipse costuma apresentar a seguir ao **this**.?

3

```
1 /**
4 package animals;
5
6 /**
7  * Classe que representa os cães.
8  * @author Miguel Goulão
9  *
10 */
11 public class DogClass extends AbstractAnimal {
12     private static final String SOUND = "Béu!Béu!";
13
14     /**
15      * Cria um cão de nome <code>name</code>.
16      * @param name - o nome do cão a criar.
17      */
18     public DogClass(String name) {
19         super(name, DogClass.SOUND);
20         this.
21     }
22 }
23
```

Member List:

- clone() : Object - Object
- equals(Object obj) : boolean - AbstractAnimal
- finalize() : void - Object
- getClass() : Class<?> - Object
- getName() : String - AbstractAnimal
- hashCode() : int - AbstractAnimal
- notify() : void - Object
- notifyAll() : void - Object
- speak() : String - AbstractAnimal
- toString() : String - Object
- wait() : void - Object
- wait(long timeout) : void - Object
- wait(long timeout, int nanos) : void - Object
- SOUND : String - DogClass

Protected Method:

```
protected Object clone()
throws CloneNotSupportedException
```

Creates and returns a copy of this object. The precise meaning of "copy" may depend on the class of the object. The general intent is that, for any object x, the expression:

```
x.clone() != x
```

- De onde surgiram estes membros?
- Alguns são específicos da classe DogClass
- Outros são herdados de AbstractAnimal
- Outros, da classe Object

A classe Object

4

- É a super-classe de **TODAS** as classes em Java
 - Todas as classes, existentes e a criar, são sub-classes de **Object**
 - **Object** não herda de nenhuma classe
 - Quando não se declara que uma classe é sub-classe de outra, automaticamente, ela fica sub-classe directa de **Object**
 - As nossas classes herdam membros da classe **Object**

Membros herdados de Object

5

- **public** boolean equals(Object obj)
 - Compara dois objectos
- protected Object clone() throws CloneNotSupportedException
 - Cria e devolve uma cópia do objecto
- protected void finalize() throws Throwable
 - Operação invocada pelo colector de lixo (*garbage collector*) sobre um objecto, quando máquina virtual determina que já não existem mais referências para o objecto
- public final Class getClass()
 - Devolve a classe concreta de um objecto, ou seja, a sua classe em tempo de execução
- public int hashCode()
 - Devolve o valor do hash code de um objecto
- **public** String toString()
 - Retorna uma **String** representando o objecto

Membros herdados de Object

6

- Os métodos `notify`, `notifyAll`, e `wait` são usados na sincronização de `threads` em programas concorrentes
 - `public final void notify()`
 - `public final void notifyAll()`
 - `public final void wait()`
 - `public final void wait(long timeout)`
 - `public final void wait(long timeout, int nanos)`

7

Identidade dos objectos e o método equals

Identidade versus igualdade

8

- A **identidade** é uma propriedade fundamental da Programação Orientada por Objectos
 - Permite aos objectos referenciarem-se uns aos outros
 - Permite a um objecto ser referenciado por múltiplos outros objectos
 - Não depende do tipo estático da referência ou variável
 - Testa-se comparando duas referências com `==`

```
void doSomething(Animal a, GermanSheppherd b) {  
    if (a == b) System.out.println("São o mesmo objecto!");  
}
```

```
void doSomething(CatClass a, CatClass b) {  
    if (a == b)
```

A identidade não depende do tipo da referência que usamos para ter acesso ao objecto

Identidade versus igualdade

9

- A **igualdade** não é o mesmo que identidade
 - Difere sempre que admitimos que dois objectos distintos possam ser considerados iguais em certas circunstâncias
 - Usamos o método **equals** para determinar que circunstâncias são essas
 - Frequentemente, o critério para a igualdade é a igualdade dos dados
 - Porém, há casos em que os dados determinam a própria identidade do objecto
 - Exemplo: **String** – a máquina virtual impede que haja duas strings com o mesmo valor: substitui todas as ocorrências pela mesmo objecto

○ método `equals`: propriedades

10

- O método `equals` implementa uma relação de equivalência de referências não nulas a objectos:
 - **Reflexivo**: para cada referência não nula ao objecto `x`, `x.equals(x)` deve retornar `true`
 - **Simétrico**: para referências não nulas `x` e `y`, `x.equals(y)` retorna `true` se e só se `y.equals(x)` retorna `true`
 - **Transitivo**: para referências não nulas `x`, `y` e `z`, se `x.equals(z)` retorna `true` e `y.equals(z)` retorna `true`, então `x.equals(y)` também tem de retornar `true`
 - **Consistente**: para quaisquer referências não-nulas `x` e `y`, múltiplas invocações de `x.equals(y)` retornam consistentemente `true`, ou consistentemente `false`, desde que nada se altere nos valores referenciados `x` e `y`
 - Para qualquer referência não nula `x`, `x.equals(null)` retorna sempre `false`

O método equals na classe Object

11

- Na classe **Object**, o método **equals** retorna a forma de equivalência mais discriminatória possível:
 - Para quaisquer referências não nulas **x** e **y**, o método **equals** retorna **true** se e só se **x** e **y** forem referências para o mesmo objecto
 - Note que, nesse caso, **x==y** também é avaliado como **true**
- Sintaxe:

```
public boolean equals(Object obj)
```
- Parâmetros:
 - **obj** – a referência do objecto que vamos comparar a **this**
- Retorno:
 - **true**, se o objecto referenciado por **this** for o mesmo que o objecto referenciado por **obj**, ou **false**, caso contrário

Implementação do método equals (classe AbstractAnimal)

12

```
public abstract class AbstractAnimal implements Animal {  
    ...  
    public boolean equals(Object obj) {  
        AbstractAnimal other = (AbstractAnimal) obj;  
        return (!name.equals(other.name)  
            && (!sound.equals(other.sound)));  
    }  
}
```

○ Funciona?

○ Depende...

- E se tivermos uma vaca e um papagaio, ambos chamados “Mimosa” e dizendo “Muuuuuh!”?
- As variáveis de instância são objectos e o argumento é um objecto
 - Se nenhum destes for **null**, tudo bem
 - Mas, e se algum for **null**?

O método equals (classe AbstractAnimal)

13

```
public boolean equals(Object obj) {  
    if (this == obj)  
        return true;  
    if (obj == null)  
        return false;  
    if (!(obj instanceof AbstractAnimal))  
        return false;  
    AbstractAnimal other = (AbstractAnimal) obj;  
    if (name == null) {  
        if (other.name != null)  
            return false;  
    } else if (!name.equals(other.name))  
        return false;  
    if (sound == null) {  
        if (other.sound != null)  
            return false;  
    } else if (!sound.equals(other.sound))  
        return false;  
    return true;  
}
```

- Método redefinido de Object, na classe AbstractAnimal
- Se obj referencia a mesma memória que this, são o mesmo objecto
- Um objecto nunca é igual a null
- Se obj não for do mesmo tipo (ou de um subtipo deste tipo), não são iguais
- As variáveis de instância têm de ser iguais
- Os vários testes a null evitam que o programa possa terminar com erro, caso alguma das variáveis esteja a null
- Vale a pena redefinir esta operação nas sub-classes?
 - Claro ...

O método equals

(classe AbstractAnimal)

14

```
public boolean equals(Object obj) {  
    if (this == obj)  
        return true;  
    if (obj == null)  
        return false;  
    if (!(obj instanceof AbstractAnimal))  
        return false;  
    AbstractAnimal other = (AbstractAnimal) obj;  
    if (name == null) {  
        if (other.name != null)  
            return false;  
    } else if (!name.equals(other.name))  
        return false;  
    if (sound == null) {  
        if (other.sound != null)  
            return false;  
    } else if (!sound.equals(other.sound))  
        return false;  
    return true;  
}
```

15

○ método toString

○ método `toString`

16

- O método `toString` da classe `Object` devolve uma `String` representando o objecto
- A representação como `String` de um objecto depende completamente do objecto, pelo que tipicamente as classes devem redefinir o método `toString`, em vez de usar a definição de `Object`
- Podemos usar a operação `toString()` juntamente com o `System.out.println()`, para escrever na consola, de modo prático, informação sobre o objecto

Método toString

(classe AbstractAnimal)

17

○ Exemplo:

```
public class AbstractAnimalClass implements Animal {  
    ...  
    public String toString() {  
        return this.getName() + ":" + this.speak();  
    }  
}  
  
public class Main() {  
    public static void main(String[] args) {  
        Animal garfield = new CatClass("Garfield");  
        System.out.println(garfield);  
    }  
}
```

○ Retorno:

Garfield:Miau!