Intermezzo: `Object.clone`

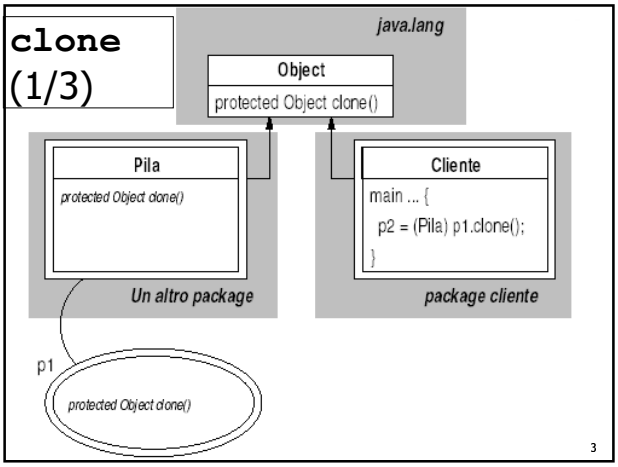
- **`protected Object clone()`**
- Duplica un oggetto
- Come per l'**`equals`**, 2 tipi di copiatura:
  - **`x = y`** copia solo la maniglia (alias)
  - **`x = y.clone()`** crea una nuova copia
- **`clone`** può essere sovrascritto in ogni classe che vogliamo rendere clonabile
- Ma è **`protected`**…

Clonazione: perché l'= non va

- **`p2 = p1`** crea un alias
- Per clonare una **`Pila`** bisogna usare **`clone`**, che
  - è ereditato da **`Object`**
  - però è **`protected`**…

```
class Cliente {
  public ... main ...{
    ...
    Pila p1, p2;
    p1 = new Pila();
    ...
    // ... modifiche a p1...
    p2 = p1;
  }
}
```


`clone` (1/3)



`clone` (2/3)

- Sostituire **`p2 = p1`** con **`p2 = (Pila) p1.clone()`**?
  - (il cast perché **`clone`** restituisce un **`Object`**)
- Non va:
  - **`clone`** della **`Pila p1`** è ereditato da **`Object`**, in cui è definito **`protected`**
  - è visibile solo nel package di **`Object`** o nelle sottoclassi di **`Pila`**
  - **`Cliente`** potrebbe vedere il metodo **`clone`** di **`Object`**, non di **`Pila`**!

`clone` (3/3)

- Soluzione: **`Pila`** ri-implementa **`clone`** e lo rende **`public`**
  - Ri-implementazione standard: uso il **`clone`** della sopraclasse

```
public Object clone()
        throws CloneNotSupportedException {
  return super.clone();
}
```


Osservazioni

- L'unica differenza è la visibilità
- Non si può definire un **`clone`** che restituisce qualcosa di diverso da **`Object`**
- **`clone`** di **`Object`** fa una copia superciale (shallow copy), non profonda (deep).
- Se serve una copia profonda, bisogna modificare (sovrascrivendolo) **`clone`**
- Pila deve anche implementare l'interfaccia marcatore **`Cloneable`**