Inheritance vs. Delegation

a cura di **Angelo Furfaro**
da “Effective Java”, Bloch

Dipartimento di Ingegneria Informatica Elettronica Modellistica e Sistemistica
Università della Calabria, 87036 Rende(CS) - Italy
Email: a.furfaro@dimes.unical.it
Web: http://angelo.furfaro.dimes.unical.it

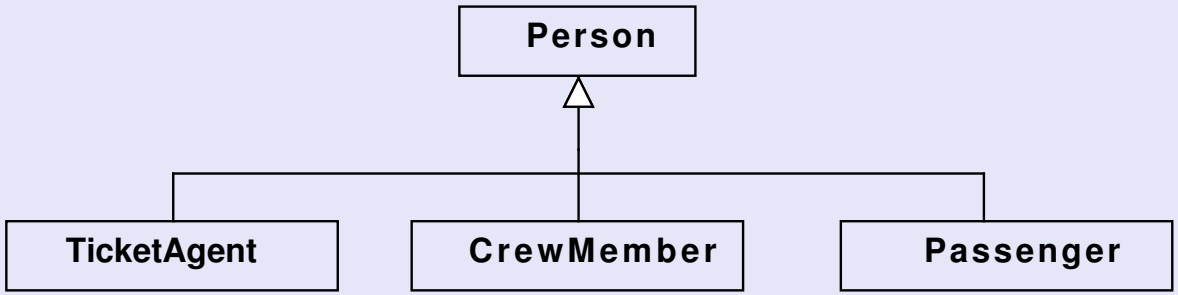
Inheritance or Delegation (by Composition) ?

- All'atto pratico, l'inheritance può risultare meno elastica della composizione ai fini della modificabiltà e riusabilità del software

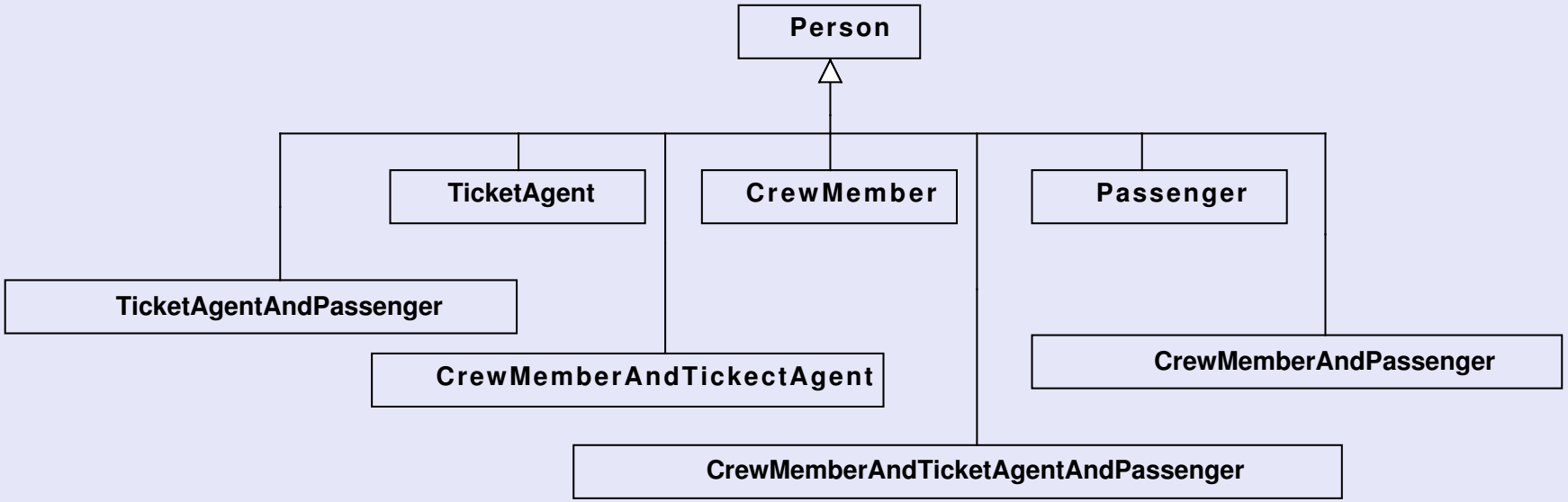
  preferire possibilmente la composizione all'inheritance
- Come osservazione generale: quando una classe di oggetti C2 è correlata ad un'altra classe C1 più per una questione di *ruolo*, allora può essere conveniente realizzare C2 mediante delegazione anziché inheritance da C1.
- Con la delegazione diventa possibile cambiare agevolmente il ruolo o giocare in tempi diversi ruoli diversi per uno stesso oggetto
- Usare la delegazione tra C2 e C1 anziché l'ereditarietà equivale a sostituire la relazione C2 **is-a** C1 tipica dell'ereditarietà con la relazione C2 **has-a** C1 tipica della delegazione.
- Nel primo caso (in Java si scrive C2 **extends** C1) un oggetto di classe C2 (classe derivata) *è* a tutti gli effetti un oggetto di classe C1 (super classe).
- Nel secondo caso, un oggetto di classe C2 *ha* dentro di sè un attributo di classe C1 (delegazione)

Ruoli ed ereditarietà

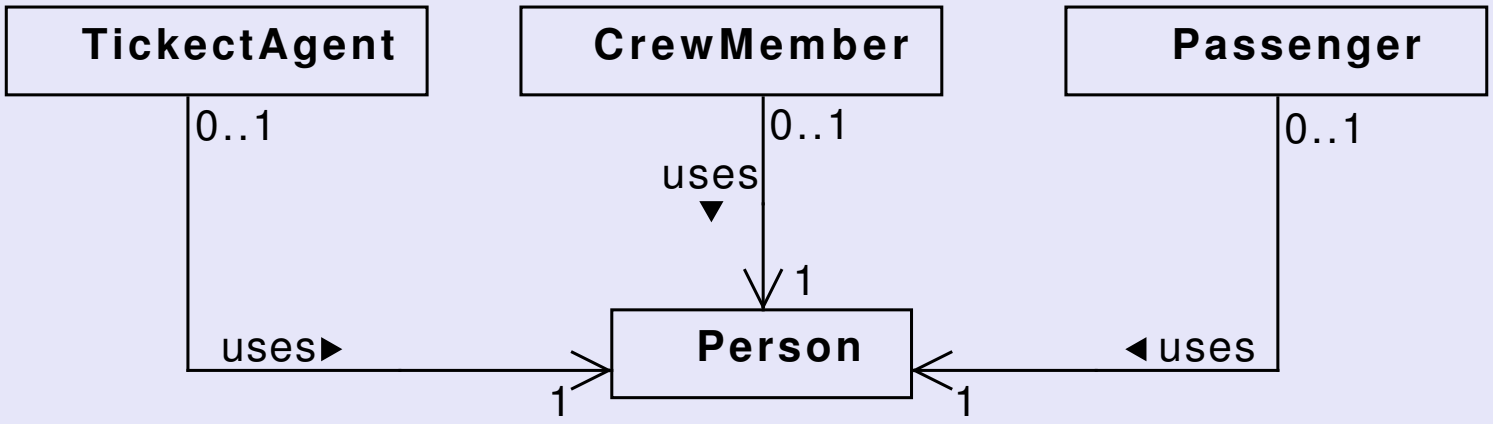
Ruoli



Ruoli Multipli



Ruoli ed delegation



Implementation Inheritance

Effective Java, Item 16 [pag. 81]

- L'implementation inheritance (una classe cioè che estenda un'altra classe) è in generale problematica in quanto viola l'incapsulazione
- Una sottoclasse dipende dai dettagli implementativi della super classe. Se questi cambiano, allora anche la sottoclasse è costretta a cambiare pur non essendo stato modificato il suo codice
- Super classe e sotto classe, pertanto, sono costrette ad evolvere in tandem

Esempio

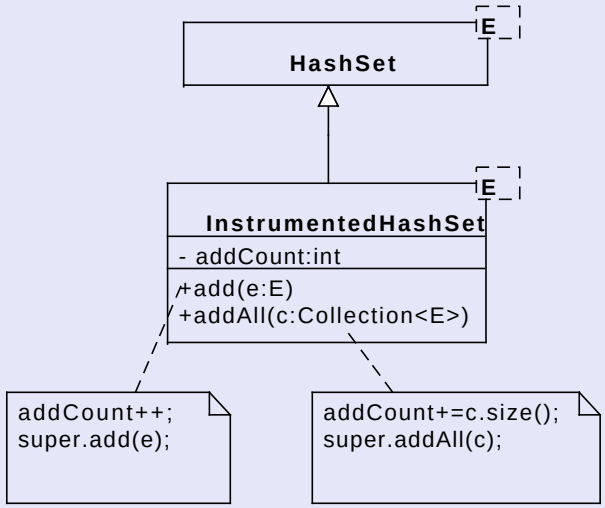
- Al fine di tenere traccia del numero di invocazioni del metodo `add()` della classe `HashSet` si introduce una classe erede `InstrumentedHashSet`
- Si introduce una variabile di istanza `addCount` ed un relativo metodo accessore
- Si sovrascrivono i metodi `add()` e `addAll()` per contare il numero di elementi inseriti
- Il codice cliente **non** funziona come ci si attende poiché l'implementazione di `addAll()` nella superclasse invoca a sua volta `add()`

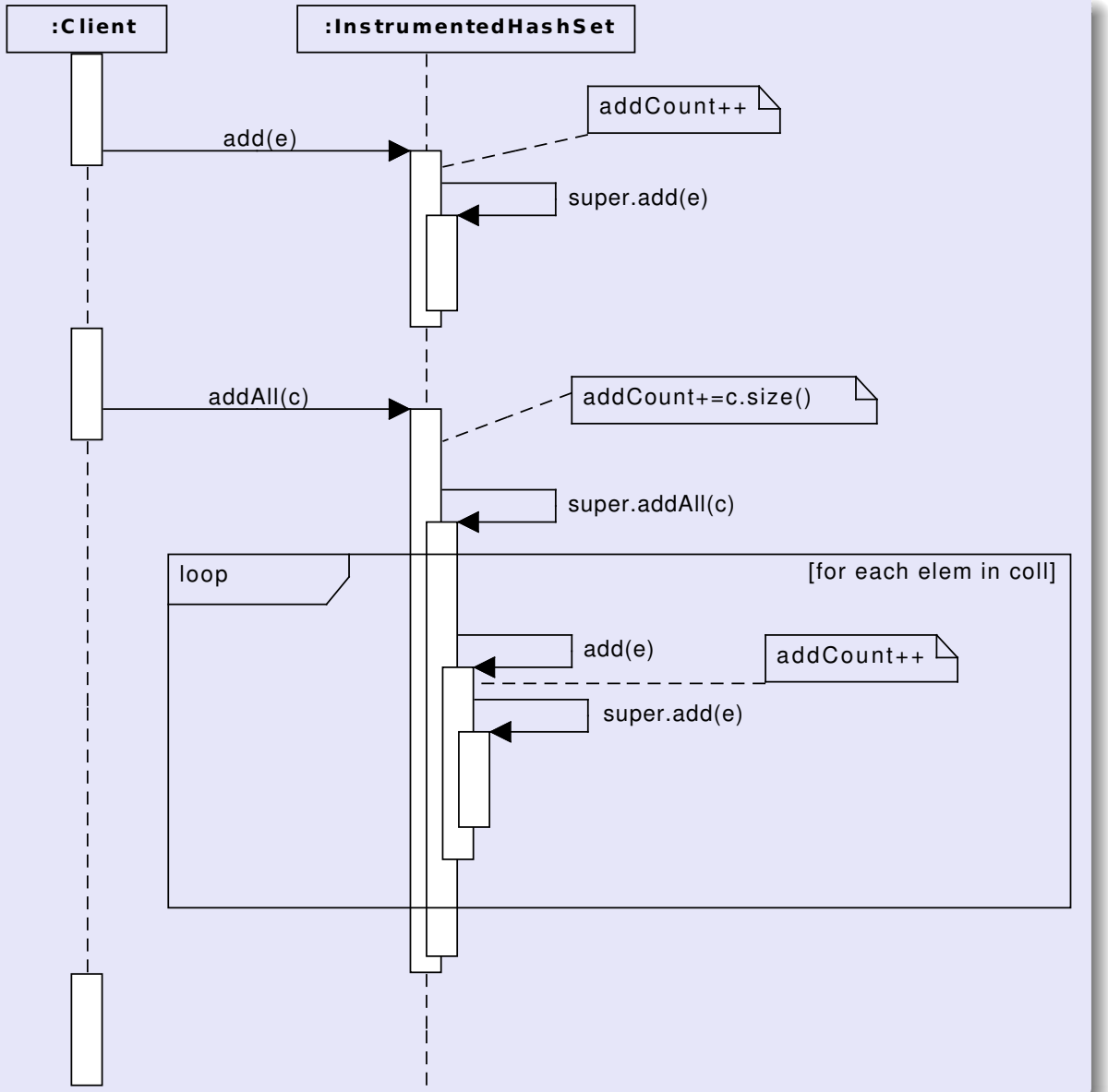
Esempio

```
// Broken — Inappropriate use of inheritance!
public class InstrumentedHashSet<E> extends HashSet<E> {
  //The number of attempted element insertions
  private int addCount = 0;
  public InstrumentedHashSet() {}
  public InstrumentedHashSet( int initCap, float loadFactor ) {
    super(initCap, loadFactor);
  }
  @Override public boolean add( E e ) {
    addCount++;
    return super.add(e);
  }//add
 @Override public boolean addAll( Collection<? extends E> c ) {
    addCount += c.size();
    return super.addAll(c);
  }//addAll
  public int getAddCount() {
    return addCount;
  }//getAddCount
}//InstrumentedHashSet
```

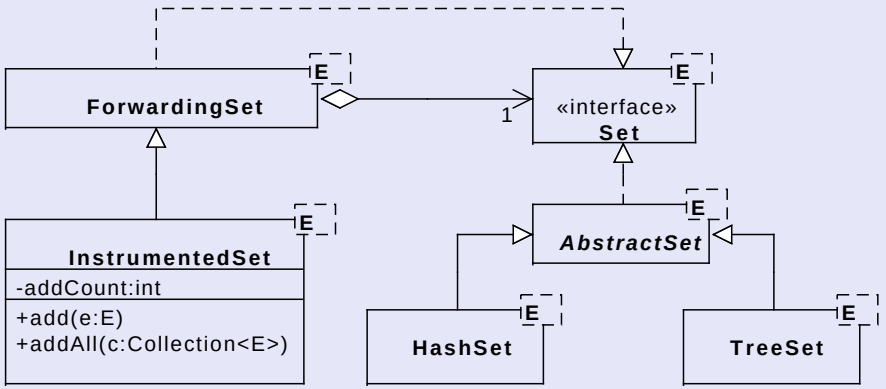
```
InstrumentedHashSet<String> s = new InstrumentedHashSet<String>();
s.addAll( Arrays.asList("Snap", "Crackle", "Pop") );
System.out.println("Numero inserimenti = "+s.getAddCount() ); //stampa 6!
```

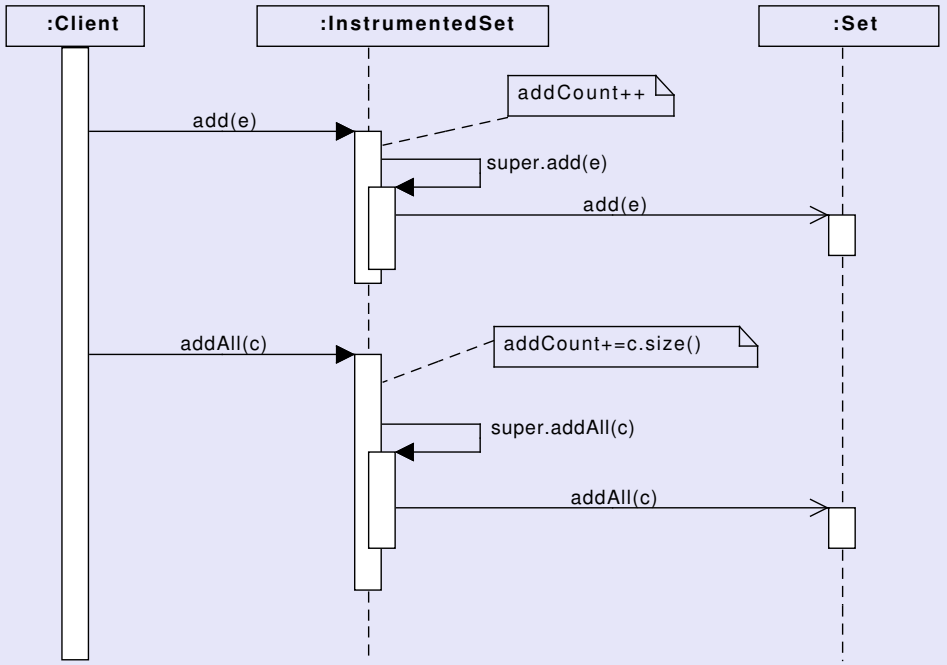
Esempio: Cattivo uso dell'ereditarietà





Esempio: soluzione basata su Decorator






```
public class ForwardingSet<E> implements Set<E>{
  private final Set<E> s;
  public ForwardingSet(Set<E> s){this.s=s;}

  public void metodoXXX(params){
    s.metodoXXX(params);
  }
  public ZZZ metodoYYY(params){
    return s.metodoYYY(params);
  }
}
```

Esempio con decorator

```
public class InstrumentedSet<E> extends ForwardingSet<E> {
   //The number of attempted element insertions
   private int addCount = 0;
   public InstrumentedSet(Set<E> s) {
         super(s);
   }
 @Override public boolean add( E e ) {
      addCount++;
      return super.add(e);
   }//add
 @Override public boolean addAll( Collection<? extends E> c ) {
      addCount += c.size();
      return super.addAll(c);
   }//addAll
   public int getAddCount() {
      return addCount;
   }//getAddCount
}//InstrumentedSet
```

```
InstrumentedSet<String> s = new InstrumentedSet<String>(new HashSet<String>());
s.addAll( Arrays.asList("Snap", "Crackle", "Pop") );
System.out.println("Numero inserimenti = "+s.getAddCount() );
```