

# Programmazione ad Oggetti

---

Collection

Eccezioni

Architettura Applicazioni



**SoftEng**  
http://www.softeng.unito.it

## Contenuti

---

- Collezioni
  - Eccezioni
-

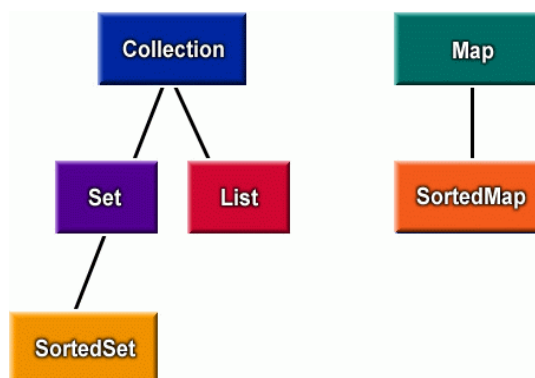
## Collection

---

- Collezione di elementi
    - ◆ reference ad Object
  - Non specificano se
    - ◆ ordinati/non ordinati
    - ◆ con duplicati / senza duplicati
- 

## Interfaces

---



## Attenzione

---

- Queste classi manipolano degli Object
    - ♦ Quando si estrae un elemento è **obbligatorio** fare un cast
  - Si possono inserire elementi di tipo diverso nello stesso contenitore:
    - ♦ Evitare di mischiare elementi non correlati
    - ♦ Inserire solo oggetti di classi che derivano dalla stessa classe di base
- 

## Iterator

---

- Astrae il problema di scandire tutti gli elementi di una collection
  - Mantiene memoria di collection corrente e di elemento corrente su collection
    - ♦ **boolean hasNext()**
    - ♦ **Object next()**
-

## Usò degli iterator

- Dentro un ciclo chiamare il metodo `next()` una sola volta
  - ♦ Ad ogni invocazione si sposta sull'elemento successivo

```
for (Iterator iter = c.iterator();
     iter.hasNext();) {
    Studente element = (Studente) iter.next();
    System.out.println(((Studente)iter.next())
                       .matricola);
}
```

Stampa la matricola  
del secondo elemento

## Collection - contenimento

```
Collection c);
Studente s1 = new
    Studente("10", "Rossi ", "Mari o");
c.add(s1);
Studente s2 = new
    Studente("10", "Rossi ", "Mari o");
if(c.contains(s2)){
    System.out.println("contiene");
}else{
    System.out.println("NON contiene");
}
```

True o False?

## Collection contenimento

---

- **contains()** è vero se la collezione contiene un elemento **uguale** al parametro
  - uguale: risultato del metodo **equals()**
    - ◆ Default in Object true se stesso oggetto
    - ◆ Da ridefinire nelle classi
      - Confronto attributi
- 

## Metodo equals

---

```
public boolean equals(Object obj){  
    Studente other = (Studente)obj;  
    return this.matricola  
        .equals(  
            other.matricola);  
}
```

---

## Ordinamento

---

- Metodo: `Collection.sort()`
  - Utilizza delle liste per ordinare
  - Due approcci
    - ◆ Uso del metodo `compareTo()`
      - `Collections.sort(l);`
    - ◆ Uso di un `Comparator`
      - `Comparator cmp=new Comparator(){ ... }`
      - `Collections.sort(l, cmp);`
- 

## CompareTo

---

- Restituisce
    - ◆ -1 se `this < other`
    - ◆ 0 se `this == other`
    - ◆ 1 se `this > other`
- ```
public int compareTo(Object obj) {  
    Studente other = (Studente)obj;  
    return matricola  
        .compareTo(  
            o.matricola);  
}
```

N.B. Il cast è sempre necessario

## Comparator

```

Comparator cmp=new Comparator(){
    public int compare(Object obj 1,
                       Object obj 2) {
        Studente s1 = (Studente)obj 1;
        Studente s2 = (Studente)obj 2;
        return - s1.compareTo(s2);
    }
};

```

N.B. Il cast è sempre necessario

## Inserimento ordinato

- Per avere una lista ordinata
- Soluzione 1:
  - ♦ Inserimento casuale + Ordinamento
    - List l = new ArrayList();
    - l.add(element)
    - Collections.sort(l);
    - return l;
- Soluzione 2:
  - ♦ Inserimento ordinato
    - TreeSet s = new TreeSet();
    - s.add(element)
    - return s;

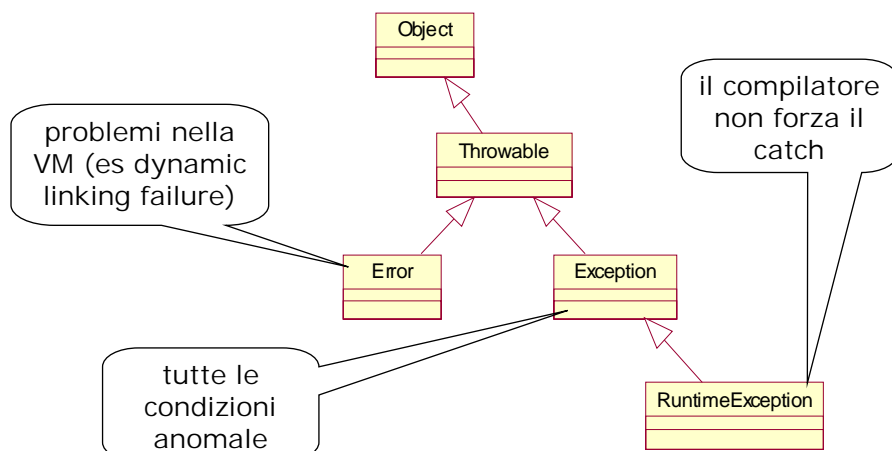
N.B. i "tree" usano **compareTo()** per ordinare gli elementi

## Map

- Si usano le mappe quando data una chiave si vuole ottenere il valore (oggetto) corrispondente
- Inserimento
  - ♦ `studenti.put(s1.matricola, s1);`
- Ricerca
  - ♦ `Studente s = (Studente) studenti.get(matricola);`

## Eccezione

- È una classe derivata da Throwable



## Esempi di eccezioni

---

- Error
    - ◆ OutOfMemoryError
  - Exception
    - ◆ ClassNotFoundException
    - ◆ InstantiationException
    - ◆ NoSuchMethodException
    - ◆ IllegalAccessException
    - ◆ NegativeArraySizeException
    - ◆ EmptyStackException
  - RuntimeException
    - ◆ NullPointerException
- 

## Gestione di eccezioni

---

- Le eccezioni sono oggetti; un tipico errore è quello di dimenticare il "new" quando si lancia l'eccezione:

```
throw new AnException("some message");
```

- Quando un frammento di codice lancia un'eccezione essa può essere gestita in due modi:

1. Intercettare l'eccezione e gestirla:

```
try{ /* lancia eccezioni */  
    }catch(AnException e){ ... }
```

2. Propagare l'eccezione al chiamante:

```
method() throws AnException {  
    /* lancia eccezioni */ }
```

---

## Run-time Exception

---

- Non obbligano a gestire l'eccezione
- Es. NumberFormatException

```
String s = "k123";
int i;
try {
    i = Integer.parseInt(s);
} catch (NumberFormatException nfe) {
    System.out.println("Numero errato");
    i = 0;
}
```

---

## Eccezioni e cicli (1)

---

- Per errori facilmente recuperabili facendo una successiva iterazione il blocco try-catch è contenuto all'interno del ciclo.
- In caso di eccezione l'esecuzione passa al catch e poi prosegue con l'iterazione successiva.

```
while(true){
    try{
        // potential exceptions
    }catch(AnException e){
        // print error message
    }
}
```

---

## Eccezioni e cicli (2)

---

- Per errori che compromettono il ciclo, blocco try-catch contiene il ciclo.
- In caso di eccezione l'esecuzione passa al catch uscendo dal ciclo.

```
try{
    while(true){
        // potential exceptions
    }
}catch(Exception e){
    // print error message
}
```

---

## Test di eccezioni

---

- Occorre distinguere due casi principali:
  - Ci si aspetta che si verifichi una situazione anomala e quindi che il codice generi un'eccezione
    - ◆ In questo caso il test fallisce se **NON** si rileva nessuna eccezione
  - Ci si aspetta un comportamento normale e quindi nessuna eccezione
    - ◆ In questo caso il test fallisce se si rileva una qualsiasi eccezione
-

## Test di eccezione attesa

---

```
try{
    // metodo chiamato con parametri errati
    oggetto.metodo("Parametro");
    fail("L'operazione XXX dovrebbe fallire");
}catch(EccezionePossibile e){
    assertTrue(true); // OK
}
```

```
class LaClasse {
    public void metodo(String p)
        throws EccezionePossibile
    { /*... */ }
}
```

## Test di eccezione NON attesa

---

```
try{
    // metodo chiamato con parametri corretti
    oggetto.metodo("Parametro");
    assertTrue(true); // OK
}catch(EccezionePossibile e){
    fail("Il metodo NON dovrebbe fallire");
}
```

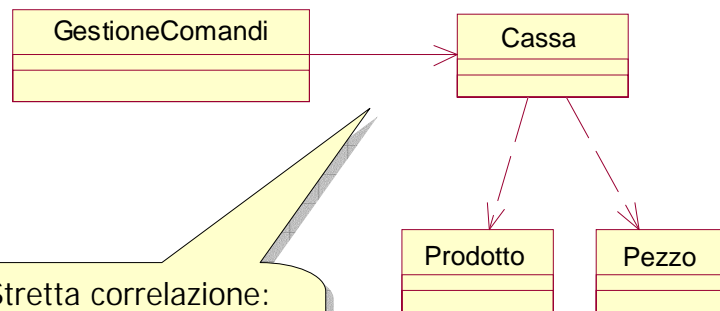
## Esercizio - Cassa

---

- Emulazione di cassa supermercato
- Interfaccia a carattere

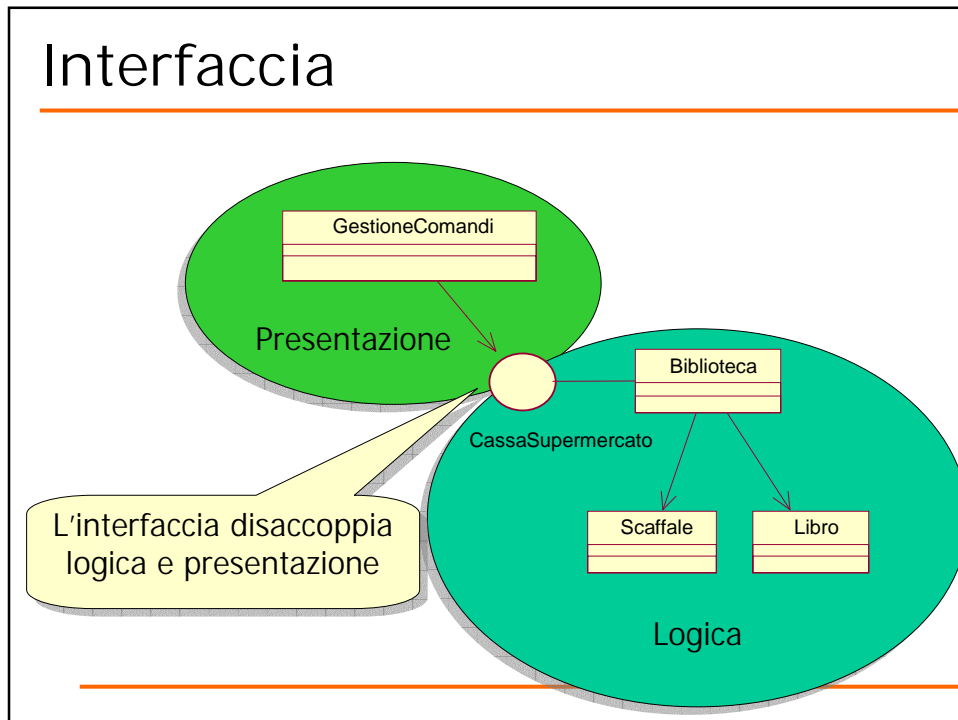
## Architettura Cassa

---

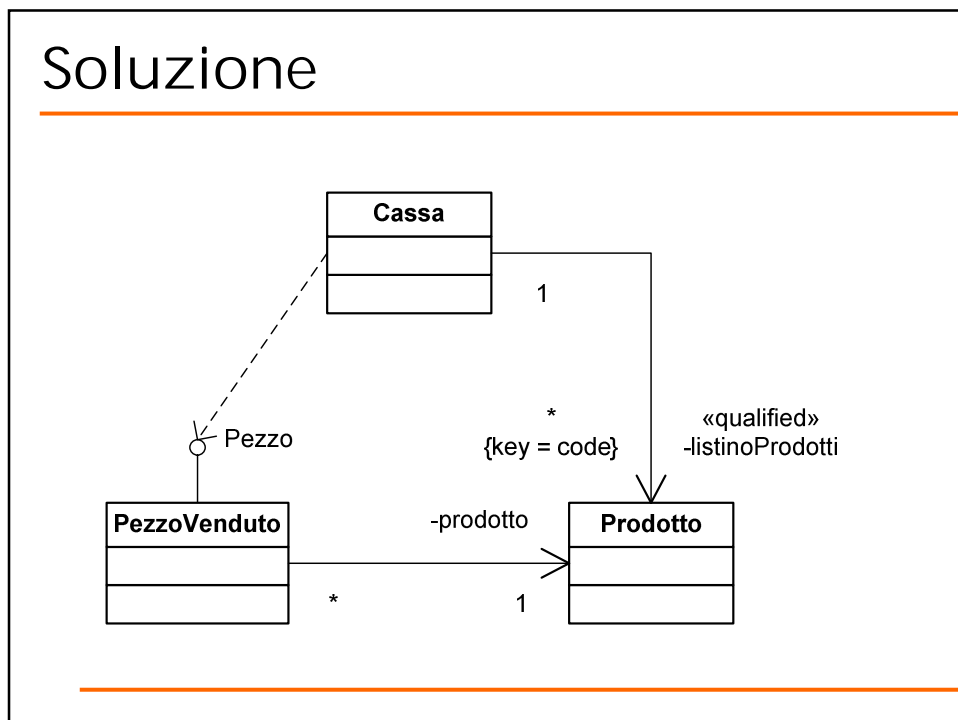


Stretta correlazione:  
ogni modifica in Cassa  
si ripercuote su  
GestioneComandi

## Interfaccia



## Soluzione



## Associazione Qualificata

