UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II
Facoltà di Ingegneria
Corso di Studi in Ingegneria Informatica

tesi di laurea

Design patterns in Java

Anno Accademico 2012/13

relatore
Ch.mo prof. Porfirio Tramontana

candidato
Luciano Amitrano
Matr. 534/2042

Progettare SW a oggetti è difficoltoso

I progettisti devono cercare di far coesistere i parametri di qualità del software al fine di soddisfare le aspettative sia di funzionamento che di struttura interna dello stesso; perciò esistono varie problematiche da considerare:

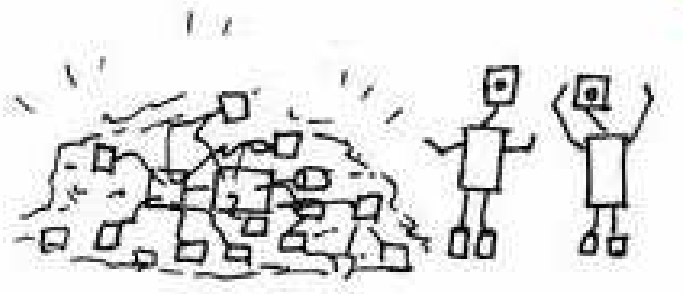
- **La scomposizione di un sistema in oggetti**
- **Stabilire la granularità degli oggetti**
- **Definizione delle interfacce**
- **Definizione dell'implementazione degli oggetti**

ERRORI NEL PROGETTO

Presagio di difetto: "BAD SMELL"

L'ingegneria del software ha inquadrato diversi casi in cui il codice sorgente assume connotati tali da far presagire difetti di programmazione. Pertanto con la termologia "code smell" (o "bad smell") si indica proprio quel codice riconosciuto come problematico non per l'effettiva correttezza di funzionamento ma per la scarsa qualità del software che si viene a determinare.

Comments	Temporary field
Long method	Data class
Long parameter list	Refused bequest
Duplicated code	Lazy class
Conditional complexity	Divergent change
Large class	E molti altri ancora…
Speculative generality	…



Uno tra i più negativi: “Duplicated code”

Violazione del principio DRY (Don't Repeat Yourself) secondo il quale una modifica ad una parte del sistema non deve replicarsi in altri punti dello stesso.

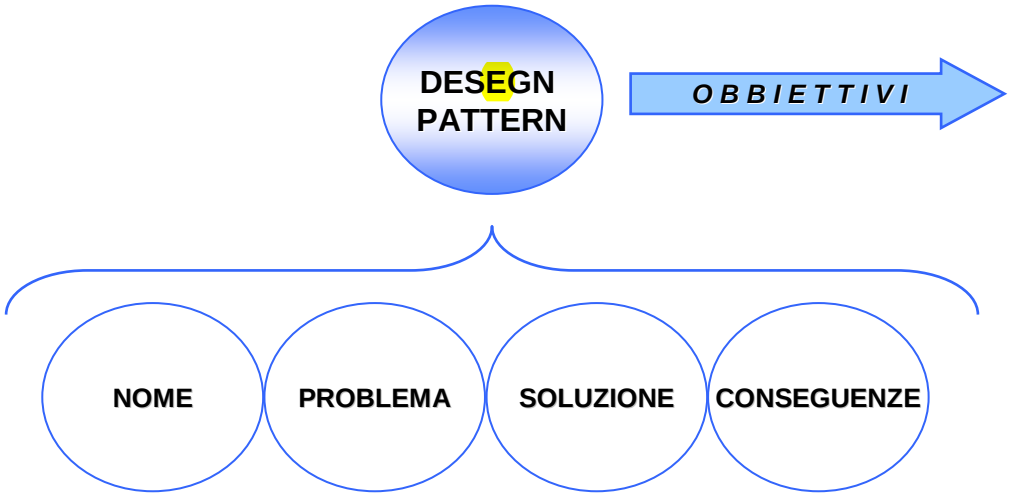
Duplicated code

```
public int somma_mul_3 (ArrayList v){

    int k=0;
    int i;
    int e;
    for(i=0; i<v.size(); i++){
      e=(int)v.get(i);
      if ( e % 3==0 )
      k=k+e;

    }
   return k;

}
```

```
public int conta_pari (ArrayList v) {

    int k=0;
    int i;
    int e;
    for(i=0; i<v.size(); i++) {
      e=(int)v.get(i);
      if ( e % 2==0 )
      k=k+1;

    }
   return k;

}
```

Verso un progetto ideale: "DESIGN PATTERN"

Nel settore dell'ingegneria del software un design pattern (testualmente "modello di progettazione") è una valida soluzione progettuale, inerente a un certo contesto, per un determinato problema che ricorre nell'attività di sviluppo software.



OBBIETTIVI

- **Codificare la buona progettazione**
- **Vocabolario comune**
- **Descrizione mirata del progetto**
- **Valido supporto per la manutenzione**

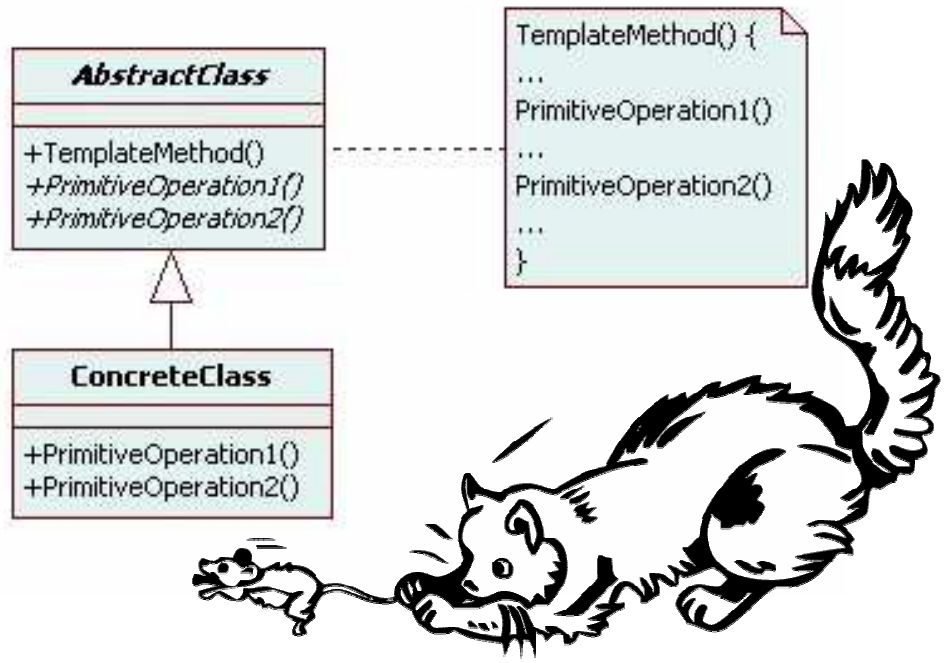
Catalogazione dei "DESIGN PATTERNS"

	1°	2°	3°
Factory method	Creazionale	Classe	Costruzione
Abstract factory	Creazionale	Oggetto	Costruzione
Builder	Creazionale	Oggetto	Costruzione
Prototype	Creazionale	Oggetto	Costruzione
Singleton	Creazionale	Oggetto	Responsabilità
Adapter	Comportamentale	Classe	Interfaccia
Bridge	Comportamentale	Oggetto	Interfaccia
Composite	Comportamentale	Oggetto	Interfaccia
Decorator	Comportamentale	Oggetto	Estensione
Facade	Comportamentale	Oggetto	Interfaccia
Proxy	Comportamentale	Oggetto	Responsabilità
Interpreter	Strutturale	Classe	Operazione
Template method	Strutturale	Classe	Operazione
Chain of responsibility	Strutturale	Oggetto	Responsabilità
Command	Strutturale	Oggetto	Operazione
Iterator	Strutturale	Oggetto	Estensione
Mediator	Strutturale	Oggetto	Responsabilità
Memento	Strutturale	Oggetto	Costruzione
Flyweight	Strutturale	Oggetto	Responsabilità
Observer	Strutturale	Oggetto	Responsabilità
State	Strutturale	Oggetto	Operazione
Strategy	Strutturale	Oggetto	Operazione
Visitor	Strutturale	Oggetto	Estensione

"DESIGN PATTERNS", vincenti sui "BAD SMELLS"!

Esempio: **Template method** → ~~**Duplicated code**~~

Il basilare intento del pattern Template Method è quello di definire un algoritmo all'interno di un metodo, rimandando a classi subordinate, la possibilità di ridefinire alcuni passi dello stesso algoritmo, senza dover implementare nuovamente la struttura di quest'ultimo. Pertanto la parte invariante dell'algoritmo è lasciata nella superclasse mentre alle classi derivate spetta il compito di implementare il comportamento soggetto a trasformazioni.



PROBLEMA

Duplicated code

```
public int somma_mul_3 (ArrayList v){

    int k=0;
    int i;
    int e;
    for(i=0; i<v.size(); i++){
      e=(int)v.get(i);
      if ( e % 3==0 )
      k=k+e;

    }
    return k;

}
```

```
public int conta_pari (ArrayList v) {

    int k=0;
    int i;
    int e;
    for(i=0; i<v.size(); i++) {
      e=(int)v.get(i);
      if ( e % 2==0 )
      k=k+1;

    }
    return k;

}
```

SOLUZIONE

```
public abstract class Studio_vet {

  public int calcola(ArrayList v){
     int k=0;
     int e;
     for(int i=0; i<v.size(); i++){
      e=(int)v.get(i);
      k=k+idoneo(e);
     }
     return k;
  }

  public abstract int idoneo(int componente);
}
```

Template method

Hook method

Classe astratta ABSTRACT CLASS

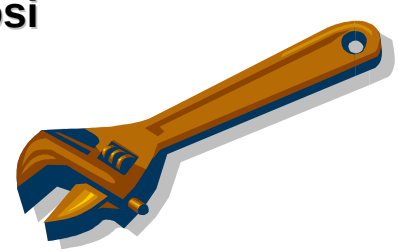
Classi CONCRETE CLASS

```
public class Multiplo_3 extends Contatore {

     public int idoneo(int e) {
      if ( (e % 3)==0 )
        return e;
        else
        return 0;
     }

}
```

```
public class Multiplo_2 extends Contatore {

     public int idoneo(int e) {
      if ( (e % 2)==0 )
        return 1;
        else
        return 0;
     }

}
```

DESIGN PATTERNS: casi di effettiva utilità

- L'istanziazione di un oggetto con un'esplicita classe lega il sistema ad una specifica implementazione.
- Dipendenza da un'operazione particolare.
- Risulta difficile la portabilità di un software su altre piattaforme
- Modifiche a cascata.
- Sussiste dipendenza tra gli oggetti e determinati algoritmi.
- Un sistema è reso fortemente complicato dai numerosi accoppiamenti esistenti tra classi.
- Estensione funzionalità attraverso sottoclassi.
- La modifica ad una classe non è fattibile.



DESIGN PATTERNS: precisazioni

- Sebbene i design pattern utilizzano metodi e classi concernenti un linguaggio orientato agli oggetti, essi, rispetto a quest'ultimo risultano pressoché indipendenti e pertanto, da un punto di vista logico, ricoprono un livello superiore.

- Infine è bene ricordare che i design pattern non devono essere applicati in modo indiscriminato pena un progetto complicato e gravoso sulle prestazioni.