

CODE REFACTORING

Ingegneria del Software 2023-2024



AGENDA

- Cosa vuole dire **refactoring**?
 - Circa ristrutturare il codice ...
- Perchè fare refactoring?
- Legacy systems, reverse engineering, re-engineering
- Concetto di **code smell**
- Processo di refactoring
- Catalogo dei refactoring
- Alcuni refactoring
 - Non tutti sono tantissimi
- Esercizio Finale
 - disponibile su Aulaweb

Già sentito?

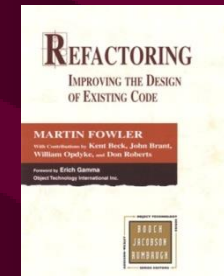


Obiettivo della lezione: solo un introduzione ...

Fowler's definition

“A change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior”

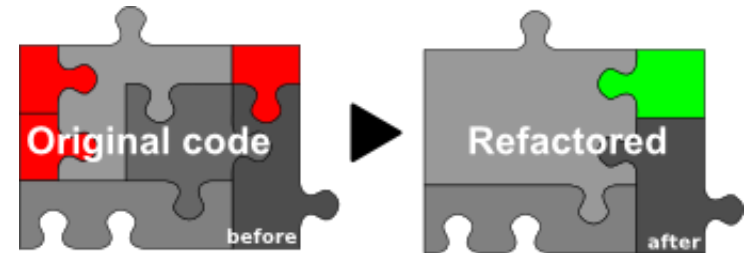
(Martin Fowler, *Refactoring*, page 53)



*Refactoring – Improving the **Design** of Existing Code*
Martin Fowler
Addison Wesley, 2000

QUINDI? COSA VUOLE DIRE REFACTORING?

- Semplificando: **una riorganizzazione, ristrutturazione del codice** (di solito OO)
 - Viene modificata la struttura, non il behavior!
 - esempio: $(x^2-1) = (x+1)(x-1)$
 - Con intenzione di migliorare le cose ...



- **Attenzione:**
 - Non vuol dire “fix a bug”!
 - Non vuol dire cambiare il linguaggio (es. C to Java)
 - Non vuol dire stravolgere il design
 - Non vuol dire cambiare la piattaforma
 - es. JavaEE to .Net

SISTEMI EREDITATI (LEGACY SYSTEM)

- Sistemi per i quali **l'attività di manutenzione è diventata prevalente** su ogni altra
 - Sono stati implementati diversi anni fa
 - La loro tecnologia (linguaggi di programmazione, stile di codifica, hw) è diventata obsoleta Cobol, RPG, BAL, Algol, Jovial, PL/I
 - Sono stati mantenuti per un lungo periodo
 - La loro struttura **si è deteriorata** e non è facile comprendere il codice
 - La loro documentazione (se esiste) non è allineata
 - Gli autori originali non sono più disponibili
 - **Contengono 'regole di business' che non sono documentate altrove**
 - **Non possono essere sostituiti facilmente**
 - **Rappresentano un grosso investimento per l'azienda**

LEGACY SYSTEM (IL MITO ...)

DON'T TOUCH IT!!

If you modify it, we will
****never**** get it working again!!

?



COME CONVIVERE CON UN LEGACY SYSTEM?

- **Obiettivo**: migliorare la qualità del SW e contenere i costi!!!
Generando la documentazione o modificando il codice
- Approcci principali:
 - **Redocumentation** = processo che mira a produrre una vista del codice “alternativa” utile per understanding
 - **Restructuring / refactoring** = trasformazione di codice “mal-strutturato” in codice “ben-strutturato”
 - Es. eliminare “i goto”: solo programmazione strutturata
 - **Reverse engineering** = creazione del design e delle specifiche a partire dal codice
 - **Re-engineering** = reverse engineering + modifica di specifiche e design + forward engineering (creazione di un nuovo sistema basato su specifiche e design rivisitati)
 - Es. - Porting da RPG a C++
 - Porting da applicazione desktop a Web

REVERSE ENGINEERING

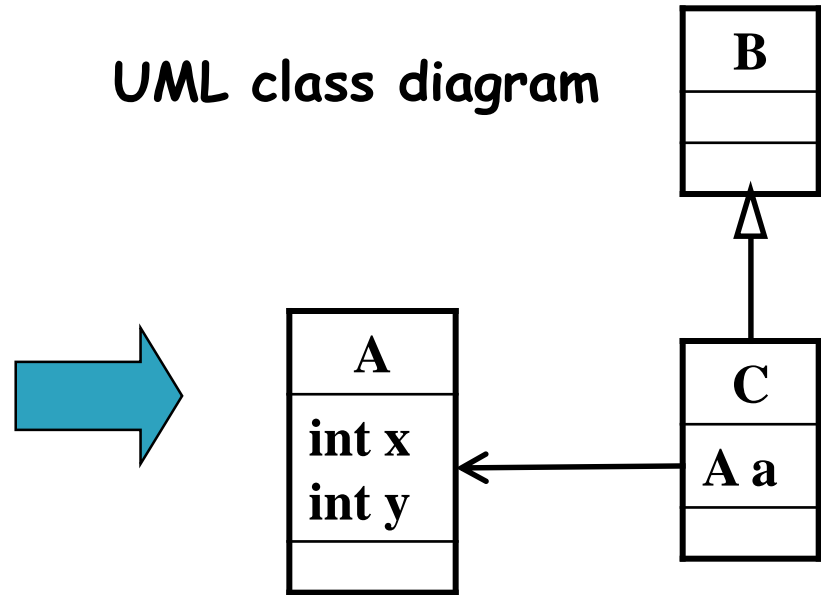
Java code

```
class A {  
    int x, y;  
}
```

```
class B {  
}
```

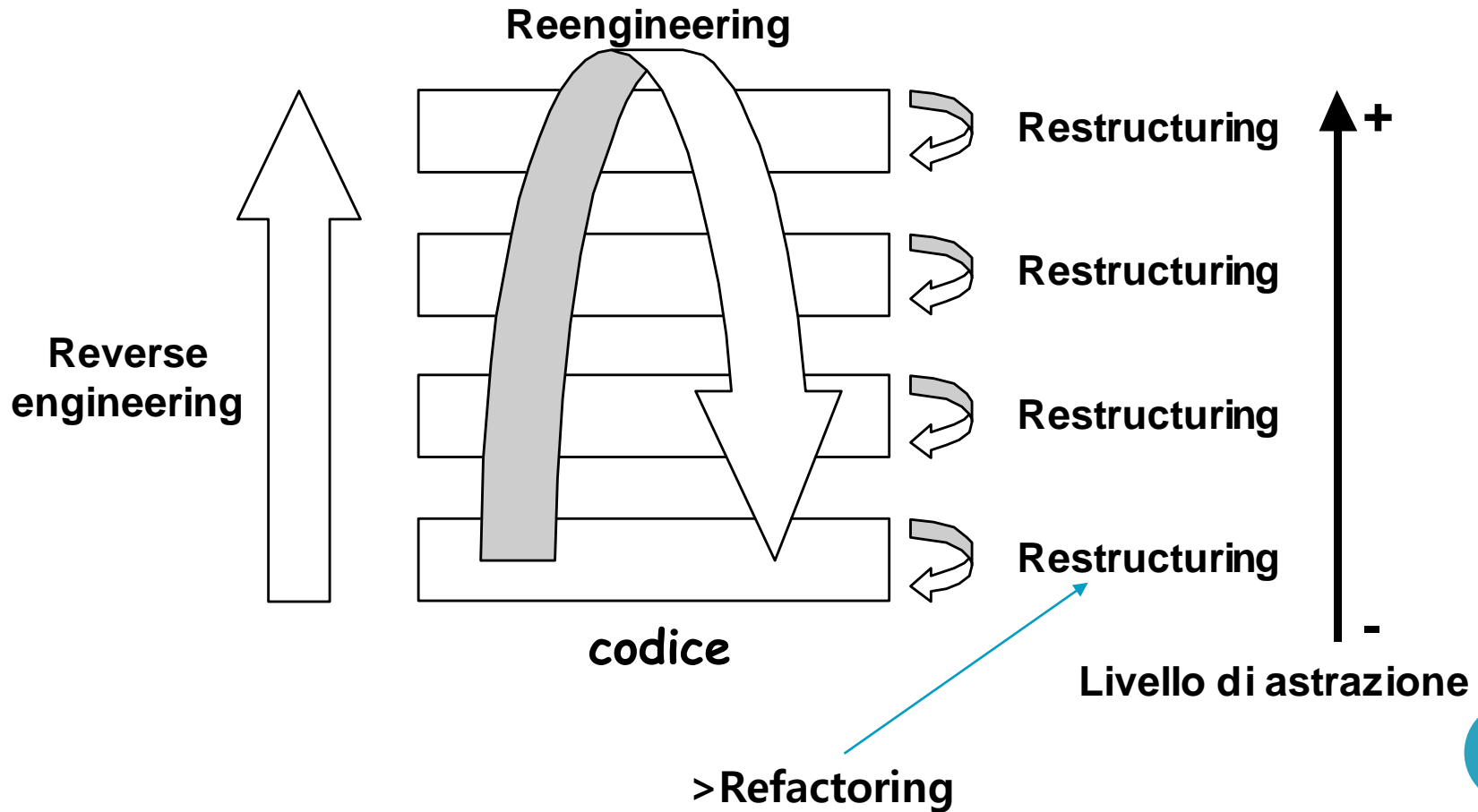
```
class C extends B {  
    A a;  
}
```

UML class diagram

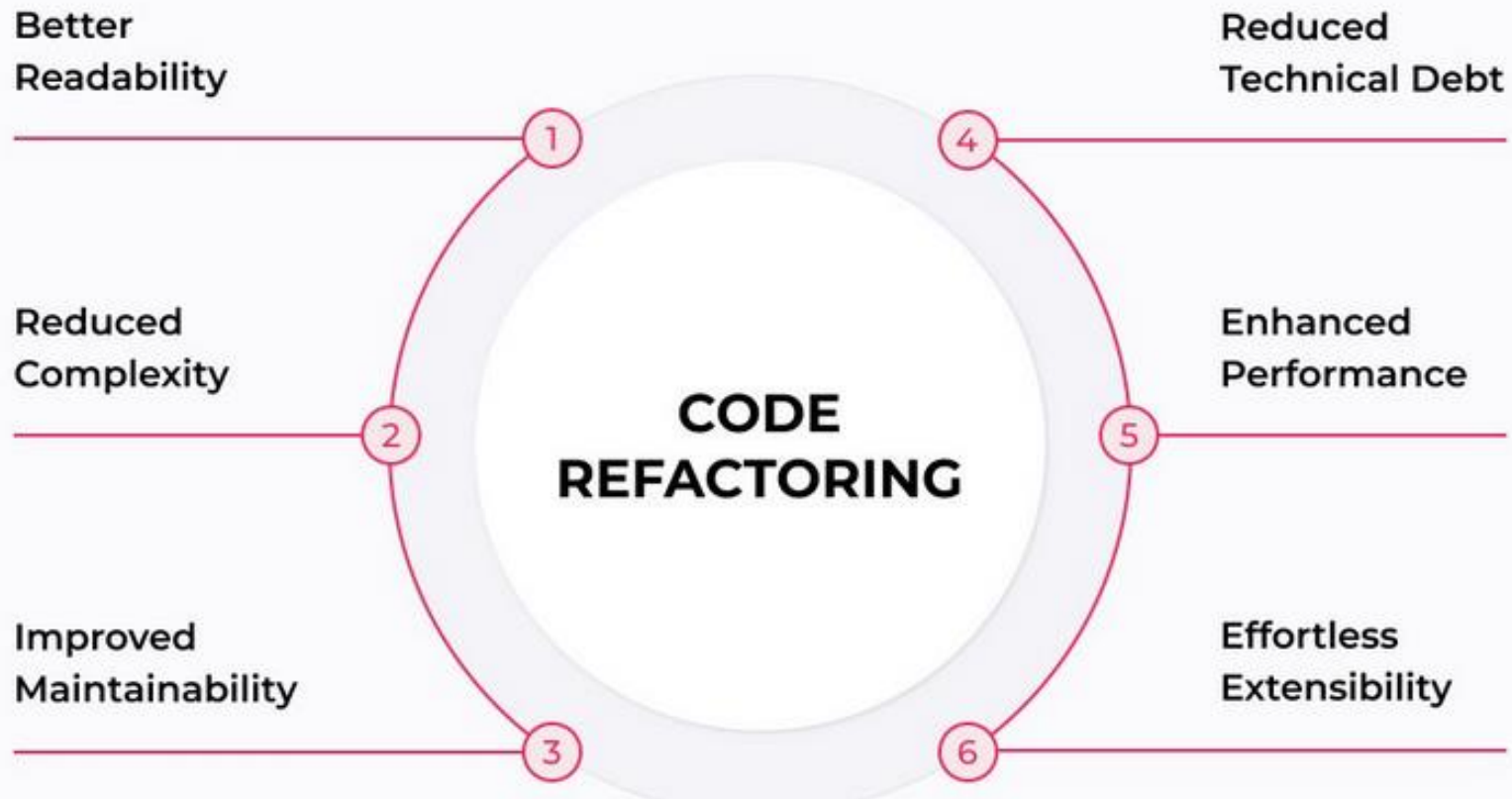


Esiste anche la definizione forte: inferire i requisiti/specifiche

REENGINEERING, REVERSE E RESTRUCTURING

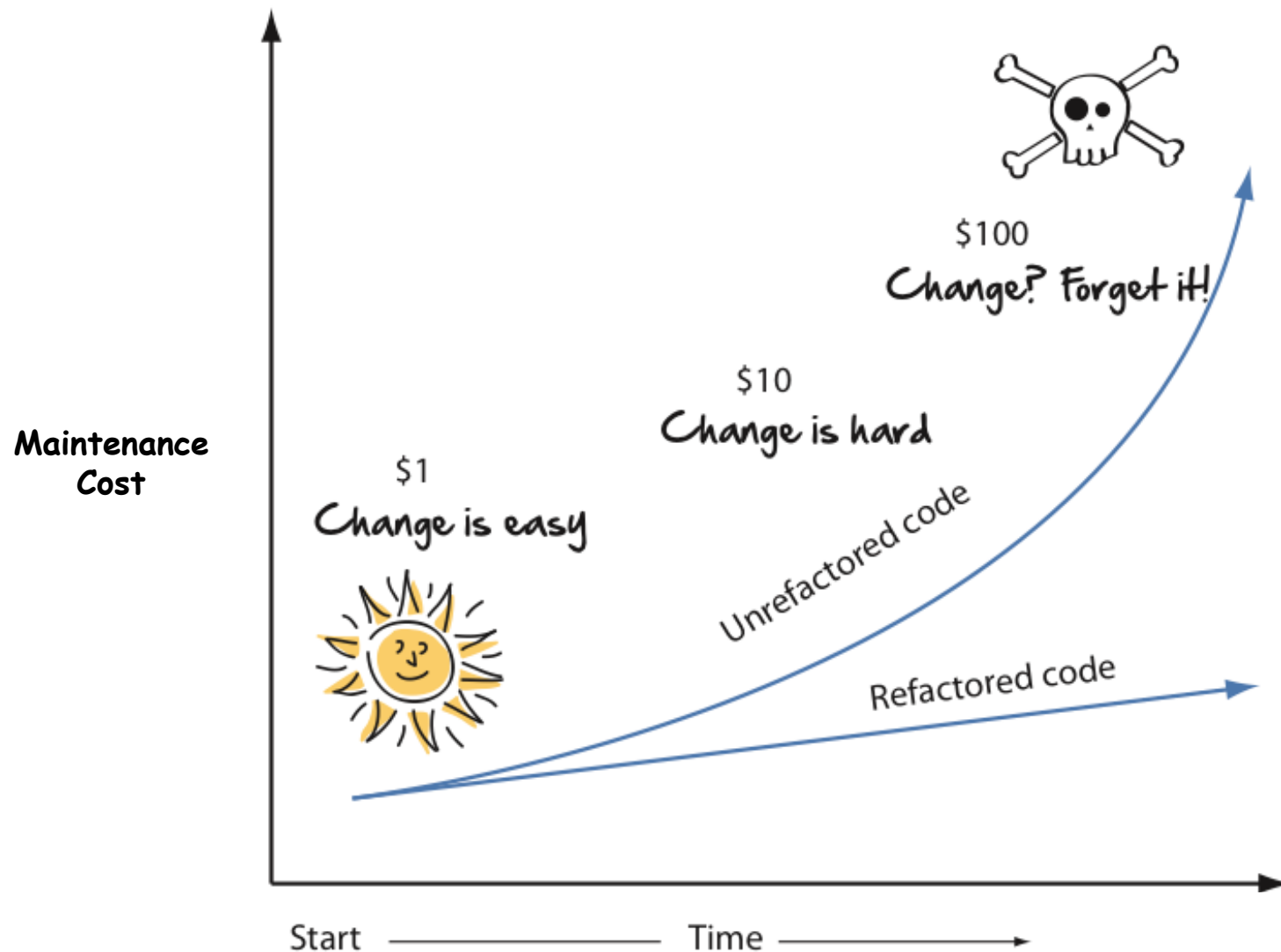


PERCHÉ FARE REFACTORING?



Ma anche semplificare fase di testing e limitare design decay/erosion

REFACTORING E MANUTENZIONE



QUANDO 'FARE' REFACTORING?

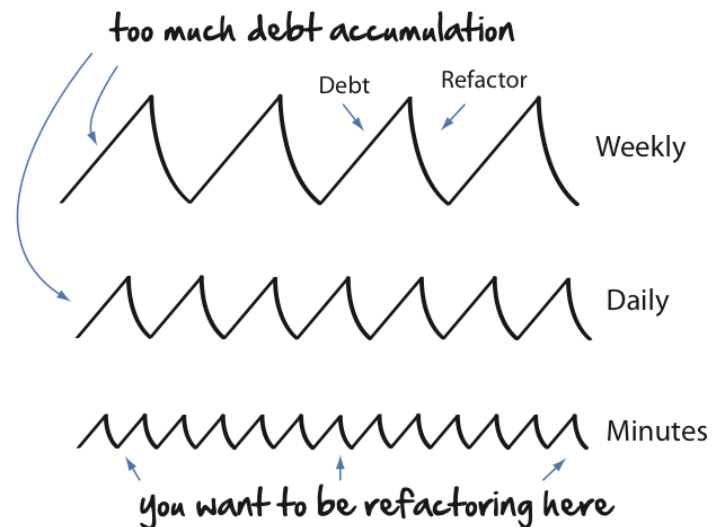
- Non pianificare:

“**due settimane di refactoring ogni due mesi**”!

- Rifattorizzare quando:

- Si vuole aggiungere una nuova funzionalità al sistema
 - il refactoring aiuta a farlo meglio e più velocemente
- Quando si fissa un bug
- Quando viene rilevato un “**code smell**”

Applicare il refactoring il più spesso possibile durante lo sviluppo!



CODE SMELL

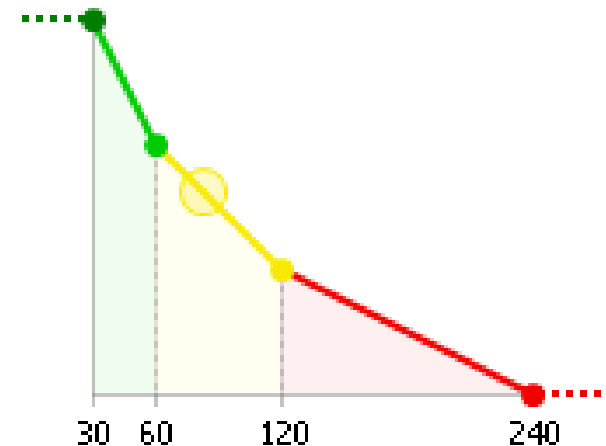
- **Indicatore** ‘che qualcosa nel codice non va bene..’
 - Indicatore non certezza!
 - Potrebbe essere solo un qualcosa di stile, oppure qualcosa che riduce understanding oppure nascosto c'è un problema più grave ...
 - Spesso i tool che calcolano le metriche del software ci indicano quali sono i **code smell** presenti
 - **Porzioni di codice da migliorare**
 - Ad esempio Stan4J che ci avverte quando un metodo è troppo *lungooooooooooooooooo ...*



What is that smell???
Did you write that code?

STAN4J

- E' possibile fissare varie soglie per le metriche
 - CC, ELOC, LCOM,
 - *Maggiore è il valore di LCOM peggiore è la coesione tra i metodi della classe*
- STAN output: **Traffic Light Ratings**
 - Per esempio: le ELOC per un metodo fino a 30 sono perfette, da 30 a 60 vanno ancora bene, da 60 a 120 iniziano a essere troppe ma sopra 120 costituiscono uno smell ...
 - Semaforo rosso!!!!



ESEMPI DI CODE SMELL

○ ‘Troppo’ codice

- Long method
- Large class
- Duplicated code (clone)
- Dead code (code that is not executed)
- Long parameter List



○ ‘Non abbastanza’ codice

- Classes with little code
- Data class
 - Solo field e getter/setter
- Empty catch clauses

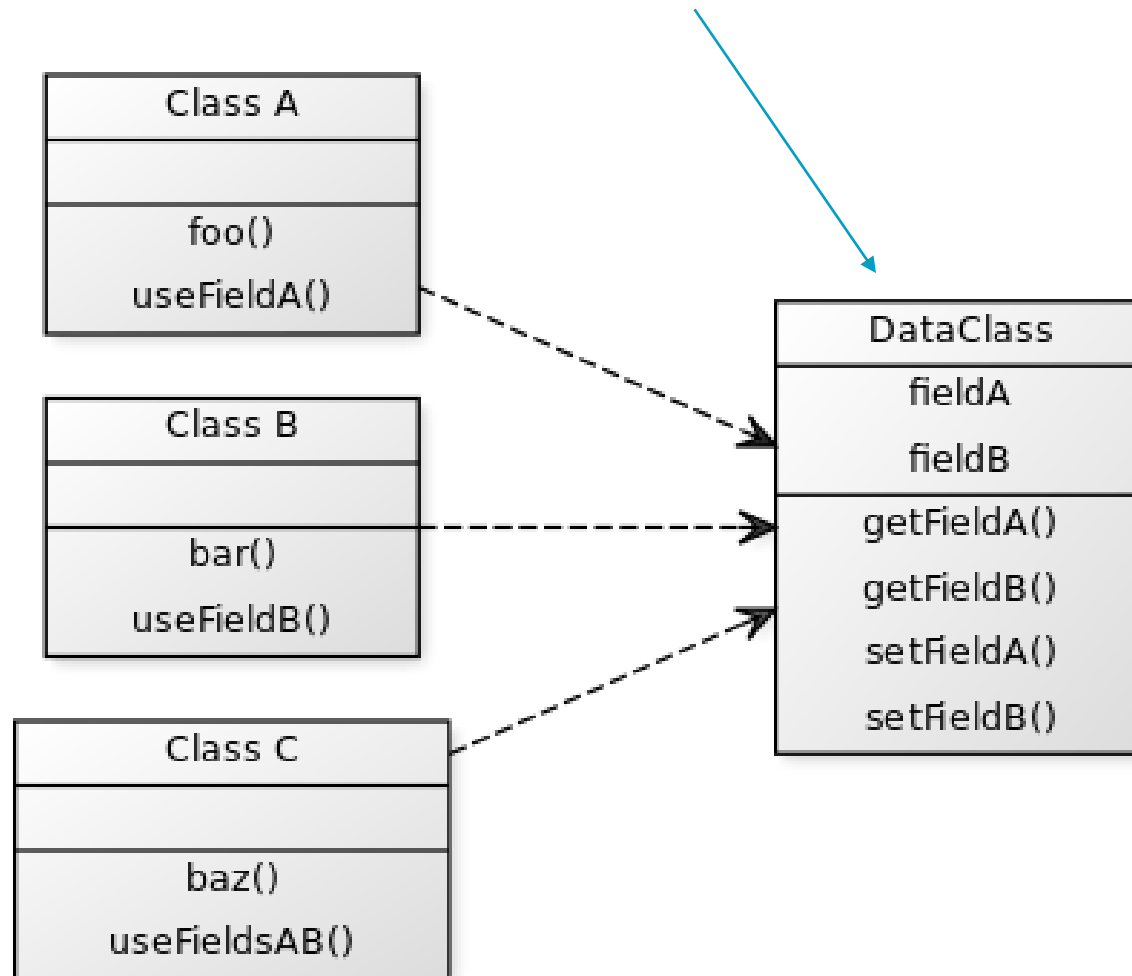
```
string text = "";  
try  
{  
    text = File.ReadAllText("test.txt");  
} catch { }
```

○ Al di fuori del codice

- Excessive commenting ...

DATA CLASS

Non contiene metodi 'con logica applicativa'

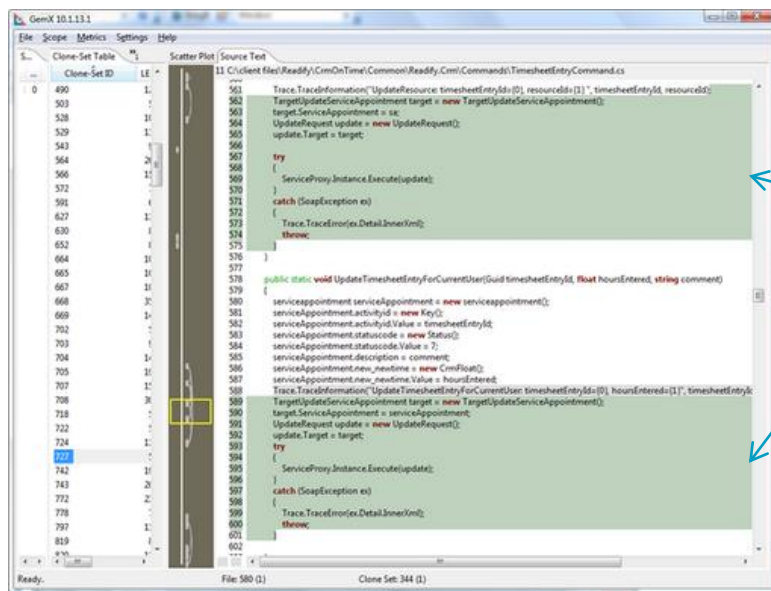


Accoppiamento elevato!

CLONI SOFTWARE



- Codice duplicato
 - Con o senza modifiche ...
- Causano “**bug propagation**” e problemi di manutenzione
- **Problema serio:**
 - Il 5% - 20% di un sistema software è codice duplicato



Software clone

ESEMPI DI CLONI

Due classi nello stesso package

```
01 package test;
02
03 public class TestFileOne {
04
05     public int factorial(int n){
06         if(n == 0){
07             return 1;
08         }else{
09             return n * factorial(n-1);
10         }
11     }
12 }
```

```
13 public int gcdOne(int a, int b) {
14     while (b != 0) {
15         if (a > b) {
16             a = a - b;
17         } else {
18             b = b - a;
19         }
20     }
21     return a;
22 }
```

```
23
24 public int mul(int a, int b){
25     int n = 0;
26     for(int i = 0; i < b; i++){
27         n += a;
28     }
29     return n;
30 }
31 }
```

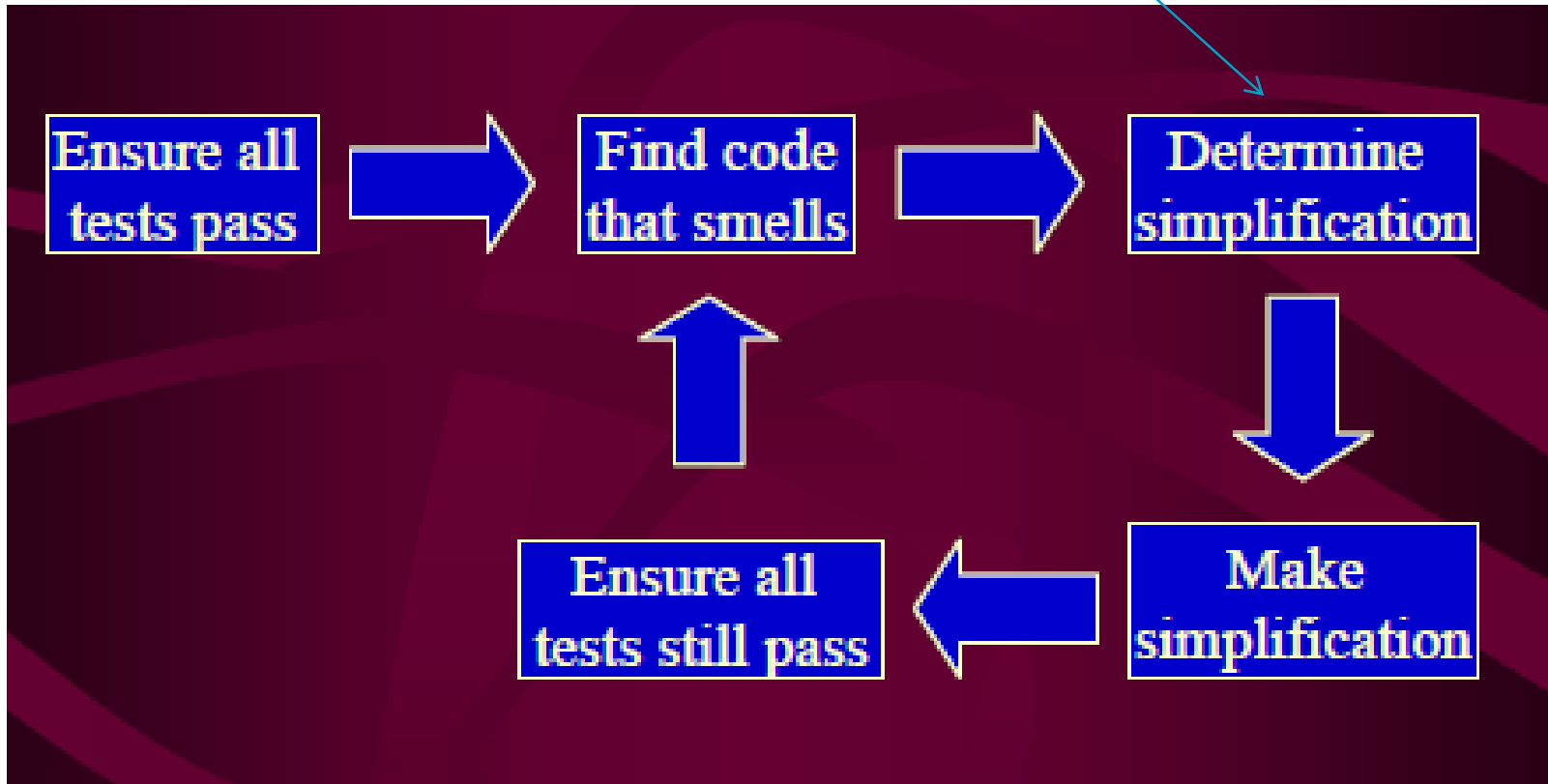
```
01 package test;
02
03 public class TestFileTwo {
04
05     public int factorial(int n){
06         if(n == 0){
07             return 1;
08         }else{
09             return n * factorial(n-1);
10         }
11     }
12 }
```

```
13 public int gcdTwo(int c, int d) {
14     while (d != 0) {
15         if (c > d) {
16             c = c - d;
17         } else {
18             d = d - c;
19         }
20     }
21     return c;
22 }
```

```
23
24 public double mul(double a, long b){
25     double x = 0.0;
26     for(long i = 0; i < b; i++){
27         x += a;
28     }
29     return x;
30 }
31 }
```

PROCESSO DI REFACTORING

Determina il refactoring adatto

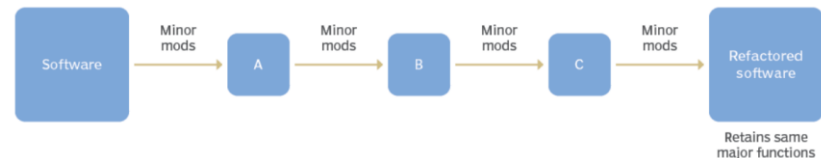


Forte link con il software testing (automatizzato)!

LA LEZIONE PIÙ IMPORTANTE

Il “ritmo” del refactoring

1. Trova/identifica code smell
2. Modifica (**piccola**) del codice
 - Seguendo una procedura definita in modo preciso
 - **Catalogo dei refactorings**
3. Compila
4. Esegui i test



Non fare il passo più lungo della gamba...



DA SMELL A REFACTORING

○ Cheatsheet

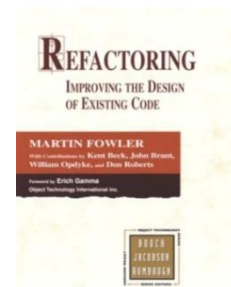
- <https://www.industriallogic.com/img/blog/2005/09/smellstorefactorings.pdf>

Smell	Refactoring
Long Method: In their description of this smell, Fowler and Beck explain several good reasons why short methods are superior to long methods. A principal reason involves the sharing of logic. <u>Two long methods may very well contain duplicated code. Yet if you break those methods into smaller methods, you can often find ways for the two to share logic.</u> Fowler and Beck also describe how small methods help explain code. If you don't understand what a chunk of code does and you extract that code to a small, well-named method, it will be easier to understand the original code. Systems that have a majority of small methods tend to be easier to extend and maintain because they're easier to understand and contain less duplication. [F 76, K 40]	Extract Method [F 110]
	Compose Method [K 123]
	Introduce Parameter Object [F 295]
	Move Accumulation to Collecting Parameter [K 313]
	Move Accumulation to Visitor [K 320]
	Decompose Conditional [F 238]
	Preserve Whole Object [F 288]
	Replace Conditional Dispatcher with Command [K 191]
	Replace Conditional Logic with Strategy [K 129]
	Replace Method with Method Object [F 135]
Long Parameter List: Long lists of parameters in a method, though common in procedural code, are difficult to understand and likely to be volatile. Consider which objects this method really needs to do its job - it's okay to make the method to do some work to track down the data it needs. [F 78]	Replace Temp with Query [F 120]
	Replace Parameter with Method [F 292]
	Introduce Parameter Object [F 295]
...	Preserve Whole Object [F 288]
	Use Delegates [F 157]

F - Fowler, Martin. Refactoring: Improving the Design of Existing Code
K - Kerievsky, Joshua. Refactoring to Patterns

CATALOGO DEI REFACTORING

Name	Description
Expand Accessors	Expands single-line getter or setter code onto multiple lines.
Expand Assignment	Expands this short form assignment to a full assignment.
Expand Getter	Expands single-line getter code onto multiple lines.
Expand Lambda Expression	Converts a lambda expression to an equivalent anonymous method.
Expand Null Coalescing O...	Converts a null coalescing operation to an equivalent ternary expression.
Expand Setter	Expands single-line setter code onto multiple lines.
Expand Ternary Expression	Expands active ternary expression to the if statement.
Extract ContentPlaceholder	Moves the selected content from a .master page to a new page.
Extract ContentPlaceHold...	Moves the content that is *outside* of the selection (in the file) to a new page.
Extract Function	Creates a new function within the enclosing namespace (or module).
Extract Interface	Generates a new interface from the public members of this class.
Extract Method	Creates a new method from the selected code block. The new method is named after the selected code block.
Extract Method to Type	Creates a new method from the selected code block and moves it to the specified type.
Extract Property	Creates a new property from the selected code block. The new property is named after the selected code block.
Extract Script	Extracts JavaScript code to an external file.
Extract String to Resource	Extracts this string to a resource file.
Extract String to Resourc...	Extracts all matching strings in the file to a resource file.
Extract Style (class)	Converts an inline style to a named class style.
Extract Style (id)	Converts an inline style to a named ID style.
Extract to XAML Resource	Extracts this string to a XAML resource file.
Extract to XAML Resourc...	Extracts all matching strings in the file to a XAML resource file.
Extract to XAML Template	Moves this template to the resource section of the file.
Extract UserControl	Creates a UserControl for the selected block including content.
Extract XML Literal to Res...	Extracts this XML literal to a resource file.
Flatten Conditional	Makes simplification of the selected condition statement.
For to ForEach	Converts a for loop into a foreach loop.
ForEach to For	Converts a foreach loop into a for loop.
Initialize Conditionally	Moves the variable initialization to an else block of the substatement.
Inline Alias	Replaces all references to a type or a namespace alias with the full name.



○ Molti refactoring sono “piccoli e semplici”

- **low-level refactoring**

- *Rename method*
- *Extract method*

○ I low-level refactoring sono i building-blocks per (o abilitano) i **refactoring più complessi**

...

- Es. quelli che introducono i design pattern

- *Replace conditional polymorphism*

with 23

List a molto lunga, in continua espansione ...

COME ESEGUIRE IL REFACTORING?

- Due possibilità:
 1. Eseguire **manualmente** i refactoring
 2. Utilizzare un **tool di supporto**
- Chiaramente usare un tool è la soluzione più comoda
 - Se cambio il nome ad una classe quanti cambi devo fare nel codice?
 - Almeno il costruttore
 - Il nome del file
 - Tutte le altre classi che la usano
 - ...
- Tuttavia spesso i tool forniscono solo refactoring semplici


In entrambi i casi è meglio sempre controllare ri-eseguendo i casi di test!



Safer Refactorings

Authors

Authors and affiliations

Anna Maria Eilertsen, Anya Helene Bagge, Volker Stolz 

Conference paper

First Online: 05 October 2016

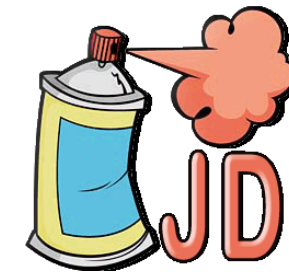
Part of the [Lecture Notes in Computer Science](#) book series (LNCS, volume 9952)

Abstract

Refactorings often require semantic correctness conditions that amount to software model checking. However, IDEs such as Eclipse's Java Development Tools implement far simpler checks on the structure of the code. This leads to the phenomenon that a seemingly innocuous refactoring can change the behaviour of the program. In this paper we demonstrate our technique of introducing runtime checks for two particular refactorings for the Java programming language: Extract And Move Method, and Extract Local Variable. These checks can, in combination with unit tests, detect changed behaviour and allow identification of which specific refactoring step introduced the deviant behaviour.

TOOLS

- Gli IDE di solito hanno incorporato alcuni refactoring semplici
 - Es. Eclipse, VS, VS code, IntelliJ
- Esistono poi tool o plug-in specifici che **suggeriscono** refactoring più complessi
 - **JDeodorant** (plug-in Eclipse) che trova alcuni Smell e applica i refactoring associati (a richiesta dell'utente)
 - God Class → *Extract Class refactoring*
 - **ReSharper** che è un plug-in per Microsoft Visual Studio (vedi TAP)
 - **GitHub Copilot** è un tool basato su AI che aiuta nello sviluppo e refactoring



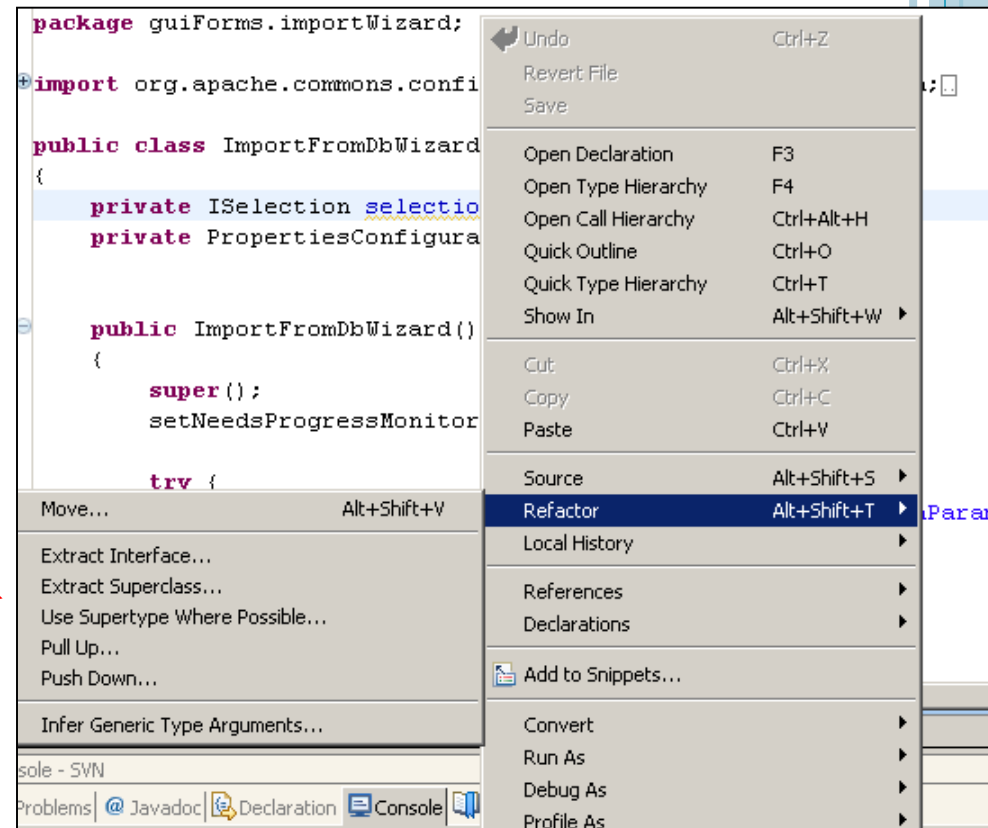
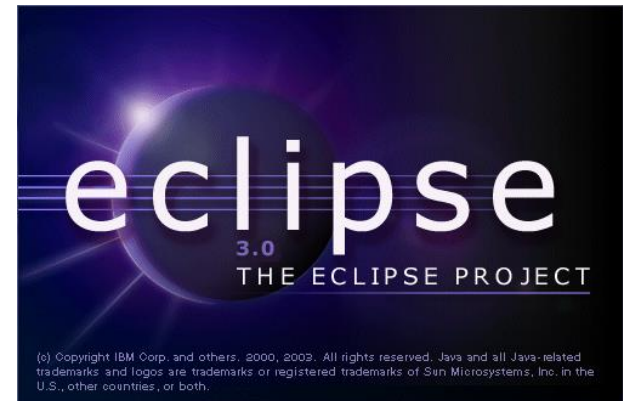
CON ECLIPSE

- ▶ Per effettuare un'operazione di refactoring:

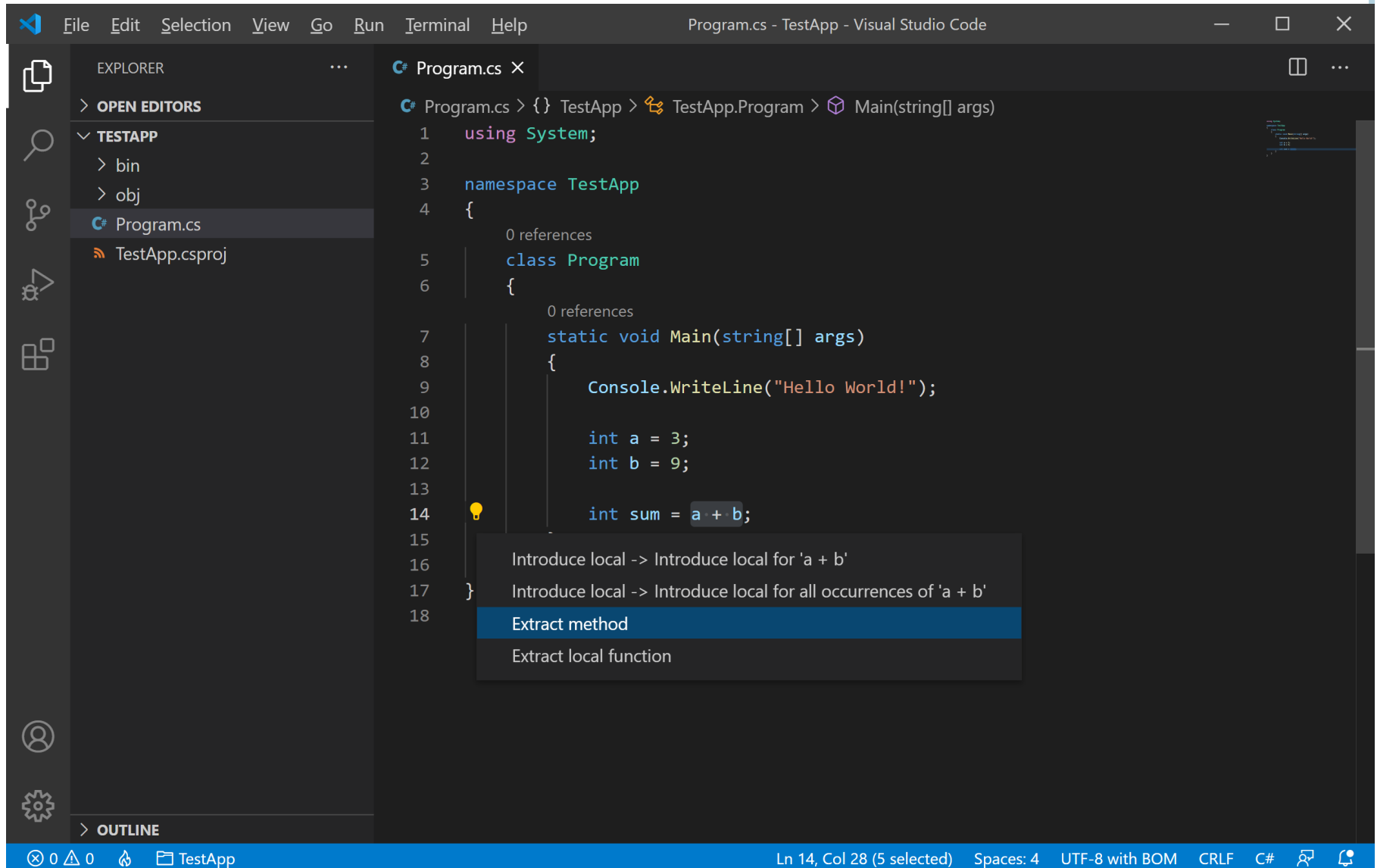
1. evidenziare l'area di interesse
2. “click” tasto destro del mouse
3. “click” Refactor

- ▶ Varie operazioni di refactoring proposte

- ▶ solo quelle possibili in quella posizione ...



CON VS CODE



ESPLICITARE LE INNER-CLASS ANONIME (ECLIPSE)

```
package com.allmycode.gui;

import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

import javax.swing.JFrame;

class CreateFrame {

    static JFrame frame;

    public static void main(String args[]) {
        frame = new JFrame();
        frame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                frame.dispose();
                System.exit(0);
            }
        });
        frame.setSize(100, 100);
        frame.setVisible(true);
    }
}
```



```
package com.allmycode.gui;

import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

import javax.swing.JFrame;

class CreateFrame {

    private static final class MyWindowAdapter
        extends WindowAdapter {

        public void windowClosing(WindowEvent e) {
            frame.dispose();
            System.exit(0);
        }
    }

    static JFrame frame;

    public static void main(String args[]) {
        frame = new JFrame();
        frame.addWindowListener(new MyWindowAdapter());
        frame.setSize(100, 100);
        frame.setVisible(true);
    }
}
```

Inner-class con nome

Inner class anonima: viene contemporaneamente definita e istanziata ma non ha nome

EXTRACT METHOD (ECLIPSE E VS CODE)

- preleva un blocco di istruzioni da un metodo
- crea un nuovo metodo **privato** nella stessa classe
- sostituisce il blocco con la chiamata al nuovo metodo

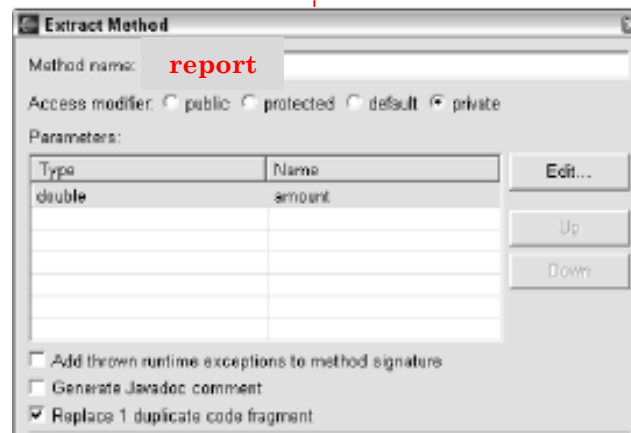
```
public class Account {  
    String name;  
  
    double balance;  
  
    void doDeposit(double amount) {  
        balance += amount;  
        System.out.println("Name: " + name);  
        System.out.println("Transaction amount: " + amount);  
        System.out.println("Ending balance: " + balance);  
    }  
  
    void doWithdrawl(double amount) {  
        balance -= amount;  
        System.out.println("Name: " + name);  
        System.out.println("Transaction amount: " + amount);  
        System.out.println("Ending balance: " + balance);  
    }  
}
```

Cloni!



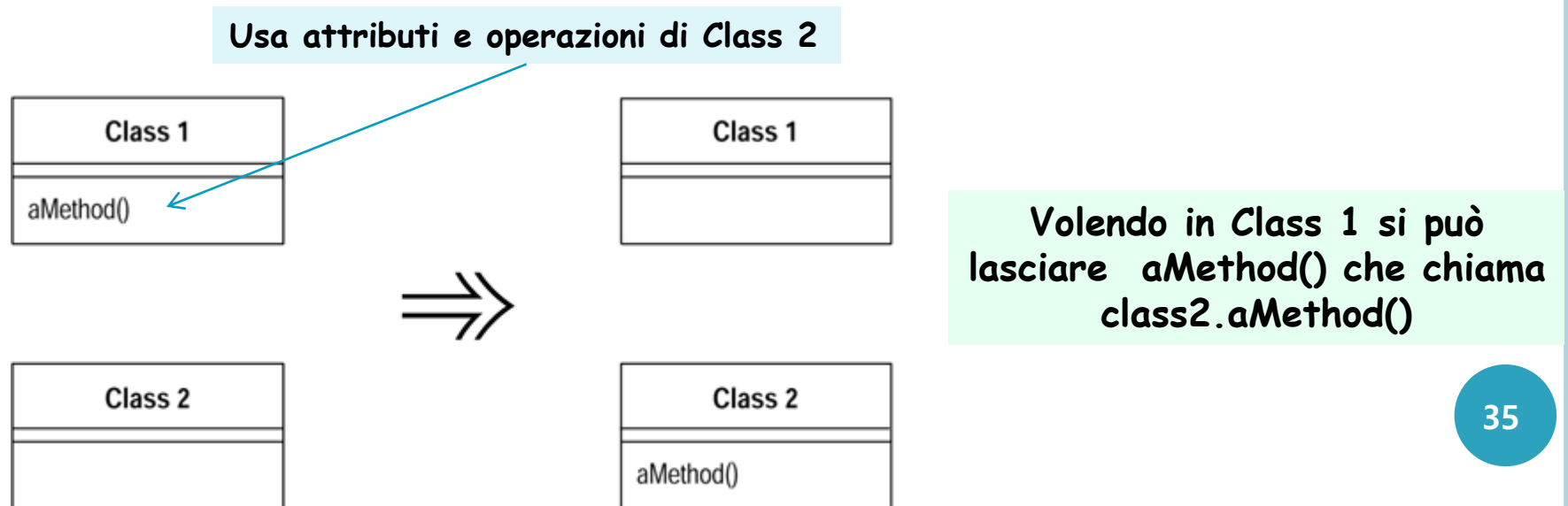
```
public class Account {  
    String name;  
  
    double balance;  
  
    void doDeposit(double amount) {  
        balance += amount;  
        report(amount);  
    }  
  
    void doWithdrawl(double amount) {  
        balance -= amount;  
        report(amount);  
    }  
  
    private void report(double amount) {  
        System.out.println("Name: " + name);  
        System.out.println("Transaction amount: " + amount);  
        System.out.println("Ending balance: " + balance);  
    }  
}
```

Nuovo metodo privato



MOVE METHOD (ECLIPSE E VS CODE)

- Si applica quando le classi hanno “**troppo behavior**” oppure quando abbiamo classi che collaborano troppo o sono troppo accoppiate ...
 - **Idea:** spostando i metodi si “semplificano” una o più classi
- Refactoring facile, **il difficile però è trovare i metodi da spostare!**
 - Buoni candidati sono metodi che sembrano riferirsi più ad altre classi che alla classe a cui appartengono



MOVE METHOD: ESEMPIO

```
class Project {  
    Person[] participants;  
}
```

```
class Person {  
    int id;  
    boolean participate(Project p) {  
        for(int i=0; i<p.participants.length; i++) {  
            if (p.participants[i].id == id) return(true);  
        }  
        return(false);  
    }  
}
```

... if (x.participate(p)) ...

Project

0..*

participants

Person

id: int

Participate()

Code smell (message chains)!

persona 'x' partecipa al progetto 'p'?



```
class Project {  
    Person[] participants;  
    boolean participate(Person x) {  
        for(int i=0; i<participants.length; i++) {  
            if (participants[i].id == x.id) return(true);  
        }  
        return(false);  
    }  
}
```

```
class Person {  
    int id;  
}
```

... if (p.participate(x)) ...

Project

0..*

participants

Person

id: int

Participate()

LineaOrdine
-quantità : int
-importo : Euro
+getProdotto()
+getQuantità()
+getImporto()
+setImporto(i : Euro)

REPLACE TEMP WITH QUERY

Metodo `getImporto()`

`_quantity` e `_itemPrice` sono attributi di classe

```
double basePrice = _quantity * _itemPrice;
if (basePrice > 1000)
    return basePrice * 0.95;
else
    return basePrice * 0.98;
```



```
double basePrice() {
    return _quantity * _itemPrice;
}
```

```
...
if (basePrice() > 1000)
    return basePrice() * 0.95;
else
    return basePrice() * 0.98; .
...
```

- Le variabili locali possono essere viste solo nel contesto di un metodo così incoraggiano ad avere metodi lunghi

- Rimpiazzando le variabili locali con un **metodo query**, ogni metodo della classe può ottenere quell'informazione

- Tre chiamate di metodo ma spesso abilita **Extract Method!**

REPLACE PARAMETER WITH METHOD

- Metodi che hanno **molte parametri sono difficili da capire** e la lista dei parametri dovrebbe essere ridotta il più possibile ...
- Se un metodo può ottenere un valore che gli è stato passato allora dovrebbe fare quel calcolo per ottenerlo

```
int basePrice = _quantity * _itemPrice;  
discountLevel = getDiscountLevel();  
double finalPrice = discountedPrice (basePrice, discountLevel);
```

→ Perché calcolarlo qui e poi passarlo?



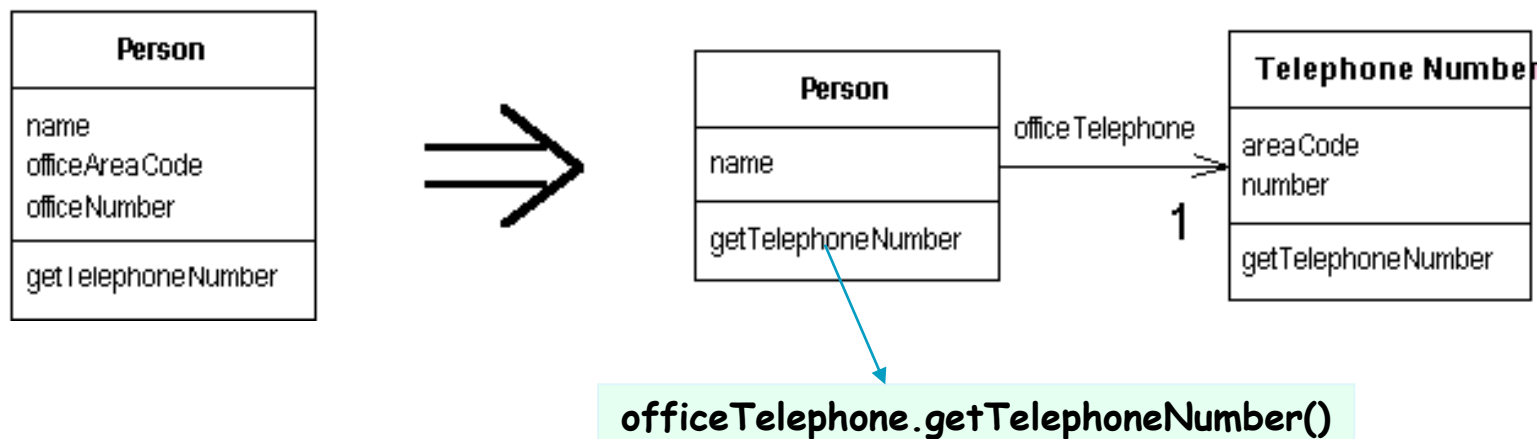
```
int basePrice = _quantity * _itemPrice;  
double finalPrice = discountedPrice (basePrice);
```

→ Adesso discountedPrice() recupera direttamente discountLevel

EXTRACT CLASS

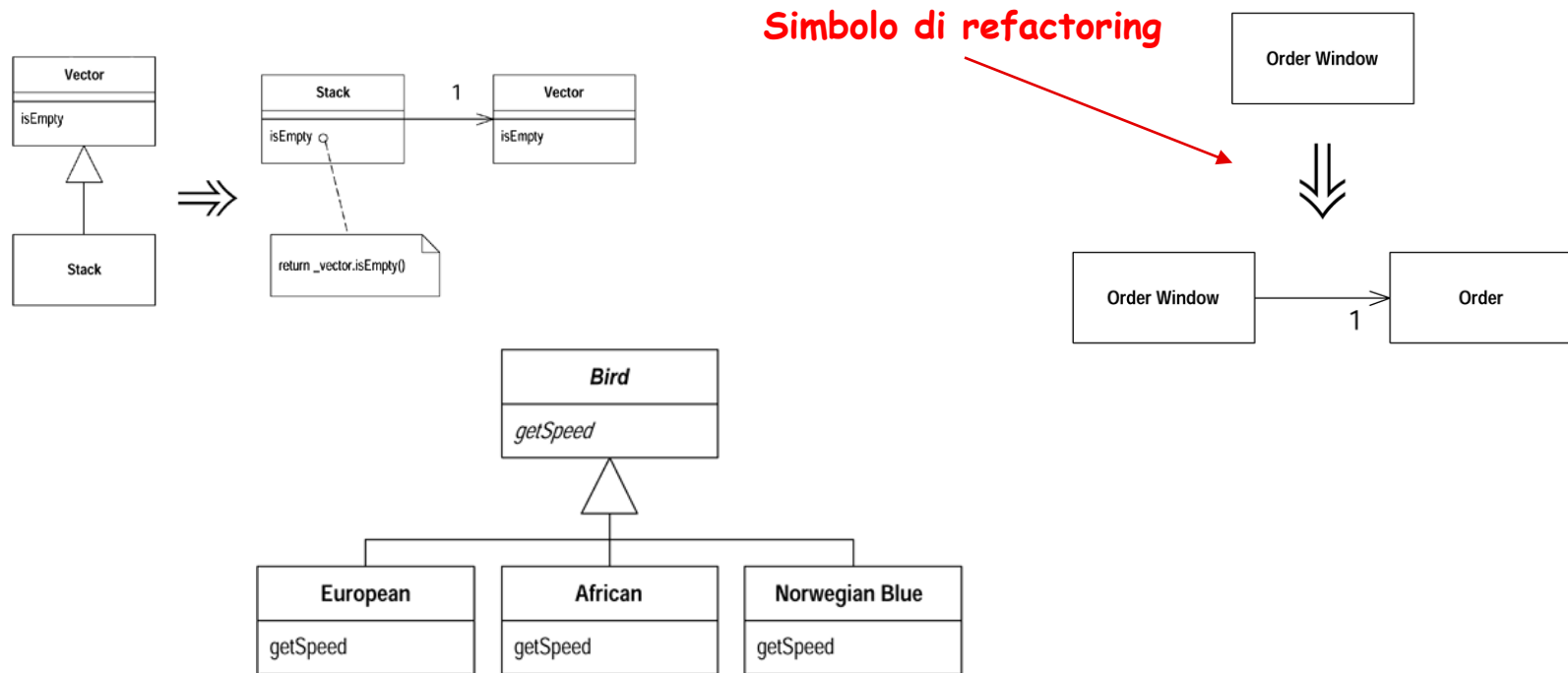


- Esiste una classe che fa troppo
 - **God class** o **Blob class** (poco coesa)
- Creiamo una nuova classe e spostiamo in questa alcuni attributi e operazioni
- Se non vogliamo modificare l'interfaccia lasciamo le operazioni “che delegano” alla nuova classe tutto il lavoro



REFACTORING PIÙ COMPLESSI

- Replace Inheritance with Delegation
- Replace Conditional with Polymorphism
- Separate Domain from Presentation

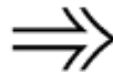
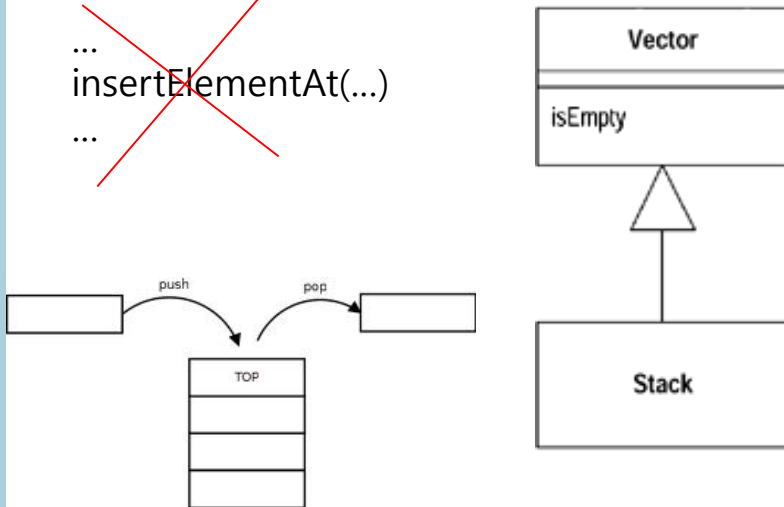


Non sono implementati direttamente in tool/plug-in

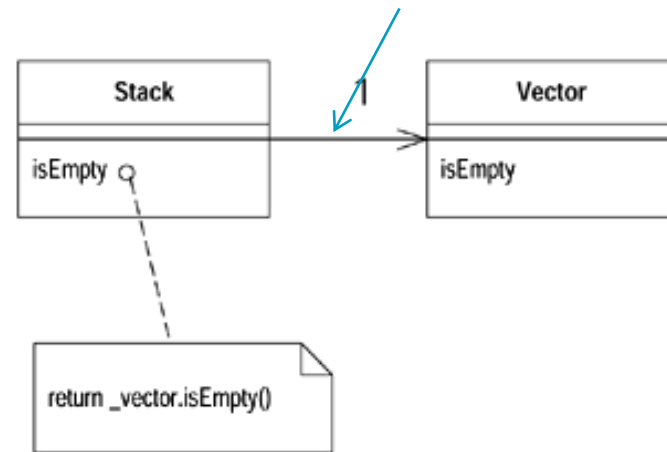
REPLACE INHERITANCE WITH DELEGATION

- Una sottoclasse usa solo parte di una superclasse e non vuole ereditare il resto ...
- Ad esempio se mi costruisco uno **Stack** a partire da un **Vector** eredito anche il metodo *insertElementAt()* che per uno Stack non ha senso ...

~~...
insertElementAt(...)
...~~



private Vector _vector = new Vector();



**In questo modo in Stack
abbiamo solo le operazioni
che ci servono!**

```
public Object pop() { // non viene gestito caso stack vuoto
    Object result = _vector.removeElementAt(_vector.size()-1);
    return result;
}
```

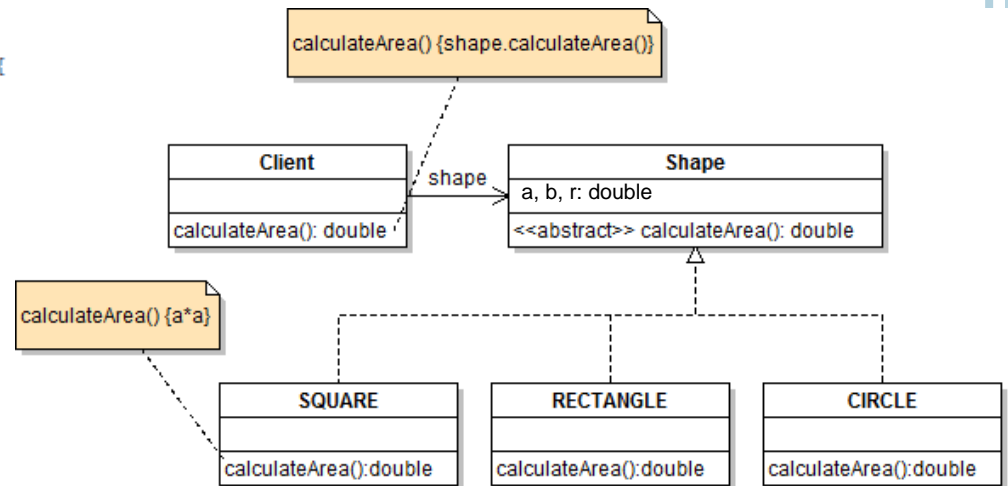
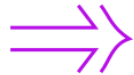
```
public void push(Object element) {
    _vector.add(element);
}
```

REPLACE CONDITIONAL WITH POLYMORPHISM

Esiste una **condizione** che sceglie differenti comportamenti a seconda del tipo/valore di una variabile

```
public class Client {  
    private double a;  
    private double b;  
    private double r;  
    ...  
    public double calculateArea(int shape) {  
        double area = 0;  
        switch(shape) {  
            case SQUARE:  
                area = a * a;  
                break;  
            case RECTANGLE:  
                area = a * b;  
                break;  
            case CIRCLE:  
                area = Math.PI * r * r;  
                break;  
        }  
        return area;  
    }  
}
```

Condizione di scelta



Design pattern State!

Extract Class (nuova classe = 'Shape') – spostare calculateArea() e gli attributi in Shape

Creare una sottoclasse per ogni 'case' – creare il metodo calculateArea() che override il metodo base presente in Shape per ogni classe

Rendere il metodo base calculateArea() di Shape astratto

SEPARATE DOMAIN FROM PRESENTATION (SOLO IDEA DI BASE)

- Abbiamo una **GUI** che contiene anche la **Business Logic**
 - Non c'è “separation of concerns” ...

Orders

Menu

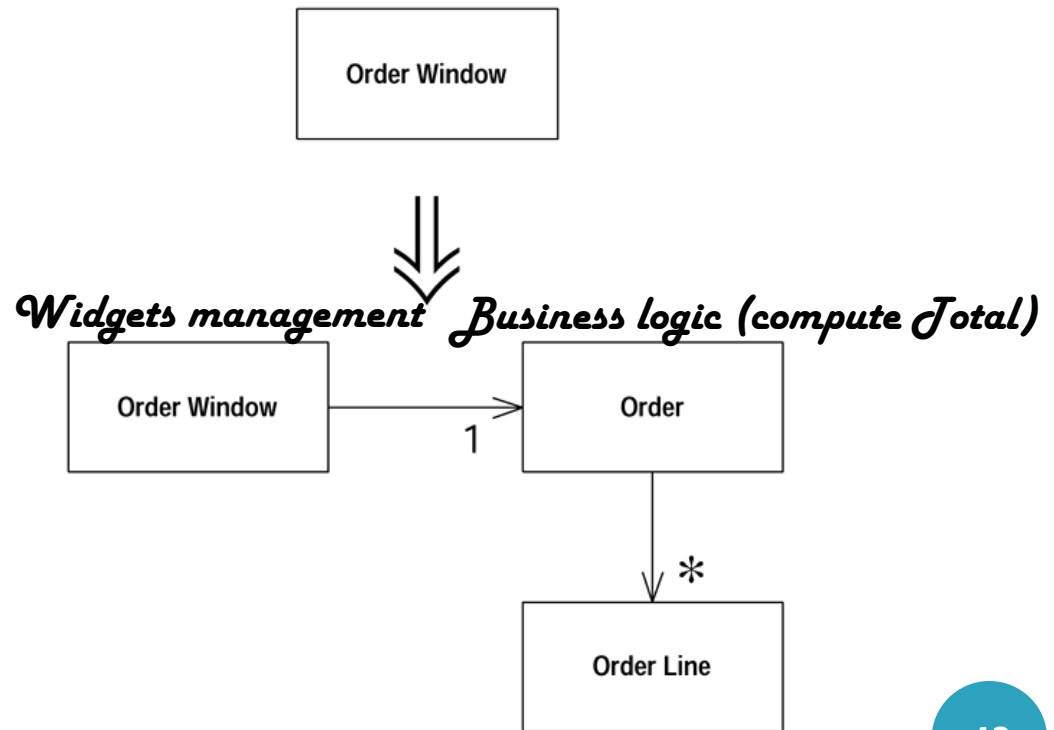
Order #

Customer

Product	Quantity	Price
Macallan	21	\$887.16
Talisker	15	\$585.49
Glenlivet	19	\$691.13
Lagavullin	5	\$261.22

Total Price

Tutto in una classe
OrderWindow



SEPARATE DOMAIN FROM PRESENTATION (SOLO IDEA DI BASE)

- Abbiamo una **GUI** che contiene anche la **Business Logic**
 - Non c'è “separation of concerns” ...

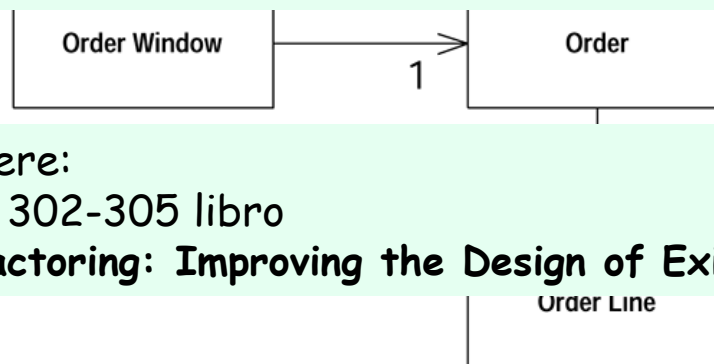
Product	Quantity	Price
Macallan	21	\$887.16
Talisker	15	\$585.49
Glenlivet	19	\$691.13
Lagavullin	5	\$261.22

Total Price: \$2,425.00

**Tutto in una classe
*OrderWindow***

Occorre applicare diversi refactoring + semplici:

- Extract Class
- Move Field
- Move Method
- Extract Method
- ...



Vedere:

Pag. 302-305 libro

Refactoring: Improving the Design of Existing Code

MATERIALE E RIFERIMENTI

- Per realizzare la seguente presentazione sono stati utilizzati:

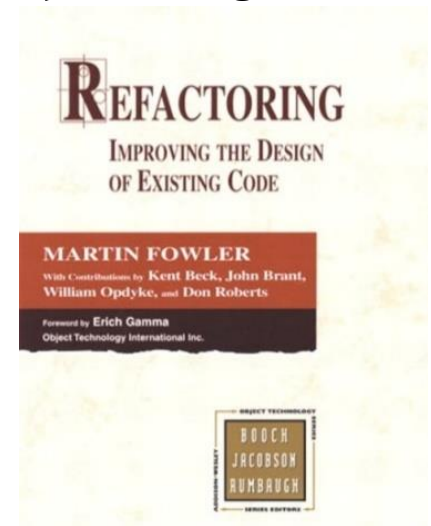
- Fowler, Martin *Refactoring – Improving the Design of Existing Code*, Addison Wesley, 2000

- Cap 1 e 2

- Fowler, Martin. Refactoring Home Page.

www.refactoring.com

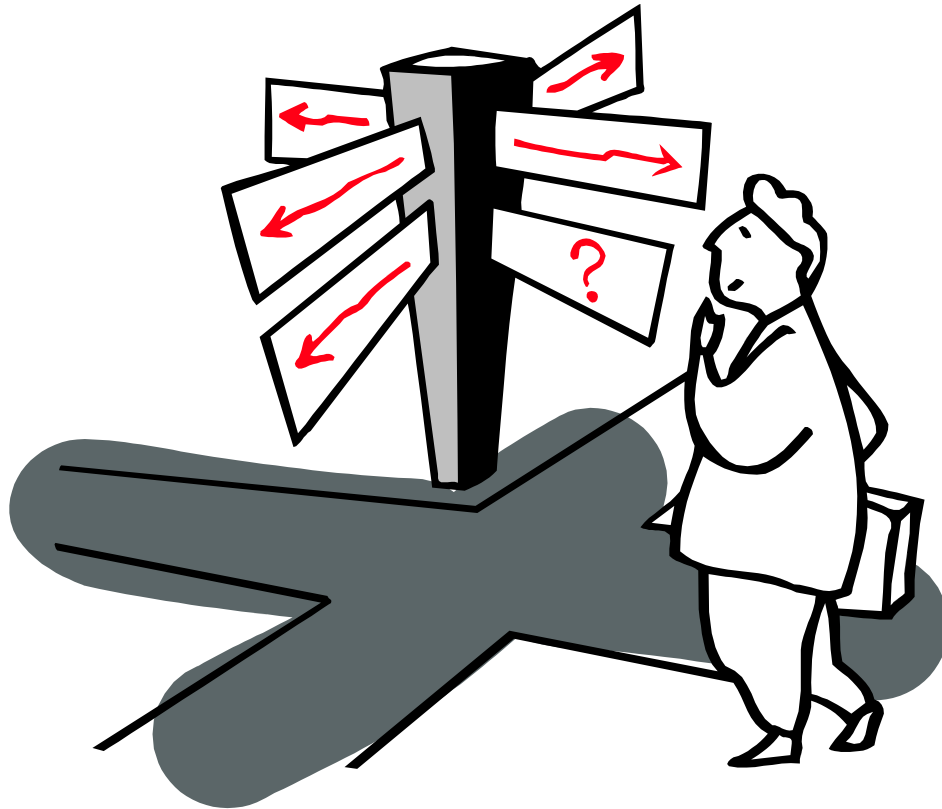
<https://www.refactoring.com/catalog/>



- <http://sourcemaking.com/refactoring>



THE END ...



Demande?