

# Refactoring

- E' il processo che cambia un sistema software in modo che
  - Il comportamento esterno del sistema non cambi
    - Ovvero i requisiti funzionali sono gli stessi
    - La struttura interna sia migliorata
- Chiamato anche
  - Miglioramento del design dopo che è stato scritto il codice
  - Ovvero, i cambiamenti sono della categoria di modifiche preventive
- Libro Consigliato
  - Fowler et al. *Refactoring: Improving the Design of Existing Code*. Pearson Addison-Wesley. 1999

# Un semplice esempio

- Consolidare (ovvero eliminare) frammenti di codice condizionale duplicati

```
if (isSpecialDeal()) {
    total = price * 0.95;
    send();
} else {
    total = price * 0.98;
    send();
}
```
- Diventa

```
if (isSpecialDeal()) {
    total = price * 0.95;
} else {
    total = price * 0.98;
}
send();
```

# Refactoring

- L'idea del refactoring è di riconoscere che è difficile fare fin dall'inizio un buon design (e codice) e, mano a mano che i requisiti cambiano, il design deve essere cambiato
  - Il refactoring fornisce tecniche per evolvere il design in piccoli passi
- Vantaggi
  - Spesso la dimensione del codice si riduce dopo il refactoring
  - Le strutture complicate si trasformano in strutture più semplici più facili da capire e mantenere
  - Si evita di introdurre un debito tecnico all'interno del design

# Principi del refactoring

- Refactoring
  - Un cambiamento alla struttura interna del sistema software per rendere più facile capirlo e modificarlo senza cambiarne il comportamento osservabile
- Lo scopo del refactoring è
  - Rendere il software più facile da capire e modificare
- In contrasto, migliorare le performance
  - Non cambia le funzionalità ma la struttura interna spesso rendendo il codice più difficile da capire
- In genere, durante lo sviluppo si hanno due fasi
  - Aggiungere una funzionalità, senza ripulire il codice
  - Refactoring, non aggiungere funzionalità, ma rendere più facile capire e modificare il codice

## Come si fa?

- Come si fa refactoring
  - Usare pattern di refactoring [Fowler]
  - Fare test costantemente
    - Come in extreme programming, prima scrivere i test, dopo fare refactoring e verificare che tutti i test siano ancora superati
    - Se un test non è superato, il refactoring ha compromesso qualcosa che funzionava, si è subito avvertiti e bisogna subito intervenire

## Perché fare refactoring?

- Refactoring
  - Migliora il design del software
    - Senza refactoring il design si deteriora mano a mano che si apportano modifiche al software
  - Rende il software più facile da capire
    - Perché la struttura del software viene migliorata, il codice duplicato rimosso, etc.
  - Aiuta a scoprire bug
    - Fare refactoring promuove la comprensione profonda del codice e questo aiuta il programmatore a trovare bug o anticipare potenziali bug
  - Permette di programmare più velocemente
    - Perché un buon design è più facile da cambiare

## Quando?

- Fare refactoring in queste occasioni
  - La terza volta che hai a che fare con la stessa cosa
    - La prima volta la implementi
    - La seconda volta che la vedi o la modifichi ti accorgi che non va bene
    - La terza volta che la vedi o la modifichi fai refactoring
  - Quando si aggiunge una nuova funzionalità
    - Prima di aggiungerla in modo da rendere più facile l'inserimento
    - Oppure dopo, per ripulire il codice dopo l'aggiunta
  - Quando occorre correggere un bug
  - Quando si ispeziona il codice, per verificarne la correttezza
- Ovvero
  - Cercare "bad smell" nel codice ed applicare una delle tecniche di refactoring per curare il problema

## Refactoring ① Estrai Metodo

- Un frammento di codice può essere raggruppato
- Far diventare quel frammento un metodo il cui nome spiega lo scopo del frammento

```
void printOwing(double amount) {
    printBanner();
    System.out.println("name: " + _name);
    System.out.println("amount: " + amount);
}

• Diventa
void printOwing(double amount) {
    printBanner();
    printDetails(amount);
}

void printDetails(double amount) {
    System.out.println("name: " + _name);
    System.out.println("amount: " + amount);
}
```

## ① Estrai Metodo

- Motivazioni
  - Cercare metodi lunghi o codice che necessita di commenti per essere comprensibile
  - Metodi piccoli sono con più probabilità usabili da parte di altri metodi
  - I metodi di alto livello, quelli che invocano altri metodi, sono più facili da comprendere, se il nome dei metodi è scelto bene
  - Fare overriding è più semplice se si hanno metodi piccoli
  - Valutare se il nome del metodo esprime ciò che il metodo fa
- Meccanica
  - Creare un nuovo metodo il cui nome comunica l'intenzione del metodo
  - Copiare il codice estratto dal metodo sorgente sul nuovo metodo
  - Guardare se il codice estratto ha variabili che sono locali al metodo sorgente e far diventare tali variabili parametri
  - Se delle variabili sono usate solo all'interno del codice estratto farle diventare variabili del nuovo metodo
  - Se una variabile locale è modificata dal codice estratto assegnare risultato
  - Sostituire il codice estratto con una chiamata al metodo

E. Tramontana - Refactoring - Ott 2008

9

## ① Estrai Metodo

```
void printOwing() {
    Enumeration e = orders.elements();
    double out = 0;

    // print banner
    System.out.println("-----");
    System.out.println("- Customer Owes -");
    System.out.println("-----");

    // calculate out
    while (e.hasMoreElements()) {
        Order each = (Order) e.nextElement();
        out += each.getAmount();
    }

    // print details
    System.out.println("name: " + _name);
    System.out.println("amount: " + out);
}
```

```
void printOwing() {
    printBanner();
    double out = getOwe();
    printDetails(out);
}
double getOwe() {
    Enumeration e = orders.elements();
    double out = 0;
    while (e.hasMoreElements()) {
        Order each = (Order) e.nextElement();
        out += each.getAmount();
    }
}
void printDetails(double amount) {
    System.out.println("name: " + _name);
    System.out.println("amount: " + amount);
}
void printBanner() {
    System.out.println("-----");
    System.out.println("- Customer Owes -");
    System.out.println("-----");
}
```

E. Tramontana - Refactoring - Ott 2008

10