

# Refactoring

- Per passare dalla versione 0.2 alla versione 0.9 è stato usata la tecnica di Refactoring Estrai Metodo
- La versione 1.2, eliminando due attributi, previene eventuali effetti collaterali (ovvero non voluti) di altre operazioni

1

Prof. Tramontana - Marzo 2019

# Refactoring

- Refactoring è il processo che cambia un sistema software in modo che il comportamento esterno del sistema non cambi, ovvero i requisiti funzionali soddisfatti sono gli stessi, per far sì che la struttura interna sia migliorata
- Si migliora la progettazione a poco a poco, durante e dopo l'implementazione del codice
- Esempio semplice: consolidare (ovvero eliminare) frammenti di codice duplicati all'interno di rami condizionali
- Libro Consigliato: Fowler et al. Refactoring: Improving the Design of Existing Code. Pearson Addison-Wesley. 1999

2

Prof. Tramontana - Marzo 2019

# Refactoring

- L'idea del refactoring è di riconoscere che è difficile ottenere fin dall'inizio una progettazione adeguata (e un codice adeguato) e quindi si deve cambiare
- Questi cambiamenti permettono quindi di incorporare più facilmente nuovi requisiti, o requisiti che cambiano
- Il refactoring fornisce tecniche per evolvere la progettazione 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 della progettazione

3

Prof. Tramontana - Marzo 2019

# Perché Fare Refactoring

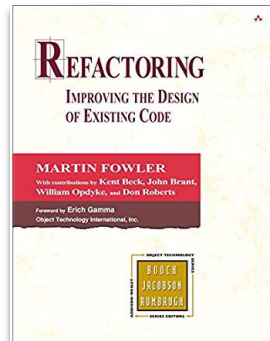
- Migliora la progettazione del sistema software
  - Senza refactoring la progettazione si deteriora mano a mano che si apportano modifiche al codice
- Rende il codice più facile da capire, perché la struttura del codice viene migliorata, il codice duplicato rimosso, etc.
- Aiuta a scoprire bug, fare 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 una buona progettazione è più facile da cambiare

4

Prof. Tramontana - Marzo 2019

# Come Si Fa Refactoring

- Usare tecniche di refactoring [Fowler]
- Fare test costantemente: 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
- [Fowler] Fowler. Refactoring Improving the Design of Existing Code. Addison-Wesley



Prof. Tramontana - Marzo 2019

5

# (1) Estrai Metodo

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

```
public void printDovuto(double amount) {  
    printBanner();  
    System.out.println("nome: " + nome);  
    System.out.println("tot: " + somma);  
}
```

- Diventa

```
public void printConto(double somma) {  
    printBanner();  
    printDettagli(somma);  
}  
  
public void printDettagli(double amount) {  
    System.out.println("nome: " + nome);  
    System.out.println("tot: " + somma);  
}
```

6

Prof. Tramontana - Marzo 2019

## Estrai Metodo: Motivazioni

- Cercare metodi lunghi o codice che necessita di commenti per essere comprensibile
- Ridurre la lunghezza dei metodi, poiché metodi piccoli sono con più probabilità utilizzabili richiamandoli da altri metodi
- Scegliere un buon nome per i metodi piccoli, che esprime ciò che il metodo fa
- I metodi di alto livello, quelli che invocano i metodi piccoli, sono più facili da comprendere, sembrano essere costituiti da una sequenza di commenti
- Fare overriding (ridefinire nella sottoclasse) è più semplice se si hanno metodi piccoli

7

Prof. Tramontana - Marzo 2019

## Estrai Metodo: Meccanismi

- Creare un nuovo metodo il cui nome comunica l'intenzione del metodo
- Copiare il codice estratto dal metodo sorgente al nuovo metodo
- Guardare se il codice estratto ha variabili locali al metodo sorgente e far diventare tali variabili parametri del metodo nuovo
- Se alcune variabili sono usate solo all'interno del codice estratto farle diventare variabili temporanee del nuovo metodo
- Se una variabile locale al metodo sorgente è modificata dal codice estratto, vedere se è possibile far sì che il metodo estratto sia una query e assegnare il risultato alla variabile locale del metodo sorgente
- Sostituire nel metodo sorgente il codice estratto con una chiamata al metodo nuovo

8

Prof. Tramontana - Marzo 2019

## Esempio Estrai Metodo

```
public class Ordine {
    private double importo;
    public double getImporto() {
        return importo;
    }
}

public class Stampe {
    private ArrayList<Ordine> ordini;
    private String nome = "Mike";

    public void printDovuto() {
        Iterator<Ordine> i = ordini.iterator();
        double tot = 0;

        // scrivi banner
        System.out.println("-----");
        System.out.println("- Cliente Dare -");
        System.out.println("-----");

        // calcola totale
        while (i.hasNext()) {
            Ordine o = i.next();
            tot += o.getImporto();
        }

        // scrivi dettagli
        System.out.println("nome: " + nome);
        System.out.println("tot: " + tot);
    }
}
```

```
public class Stampe2 {
    private ArrayList<Ordine> ordini;
    private String nome = "Mike";

    public void printDovuto() {
        printBanner();
        double tot = getTotale();
        printDettagli(tot);
    }

    public double getTotale() {
        Iterator<Ordine> i = ordini.iterator();
        double tot = 0;
        while (i.hasNext()) {
            Ordine o = i.next();
            tot += o.getImporto();
        }
        return tot;
    }

    public void printBanner() {
        System.out.println("-----");
        System.out.println("- Cliente Dare -");
        System.out.println("-----");
    }

    public void printDettagli(double somma) {
        System.out.println("nome: " + nome);
        System.out.println("tot: " + somma);
    }
}
```

Prof. Tramontana - Marzo 2019

## (2) Sostituisci Temp Con Query

- Una variabile temporanea (temp) è usata per tenere il risultato di un'espressione
- Estrarre l'espressione ed inserirla in un metodo. Sostituire tutti i riferimenti a temp con la chiamata al metodo che incapsula l'espressione. Il nuovo metodo può essere richiamato anche altrove

```
double prezzoBase = quantita * prezzo;
```

- Diventa

```
private double prezzoBase() {
    return quantita * prezzo;
}
```

10

Prof. Tramontana - Marzo 2019

## Sostituisci T Con Q: Motivazioni

- Variabili temp sono temporanee e locali. Possono essere viste solo all'interno del contesto del metodo e per questo inducono ad avere metodi lunghi, perché è il solo modo per raggiungere tali variabili
- Con la sostituzione effettuata tramite il refactoring, qualsiasi metodo può avere il dato
- Spesso si effettua il refactoring (2) Sostituisci Temp con Query prima di (1) Estrai Metodo
- Questo refactoring è facile se temp è assegnata solo una volta

11

Prof. Tramontana - Marzo 2019

## Sostituisci T Con Q: Meccanismi

- Cercare una variabile temporanea assegnata solo una volta
- Dichiarare temp final
- Compilare (per verificare che è assegnata una volta sola)
- Estrarre la parte destra dell'assegnazione e creare un metodo

12

Prof. Tramontana - Marzo 2019

## Esempio Sostituisci Temp Con Query

```
private double quantita, prezzo;

public double getPrezzo1() {
    double prezzoBase = quantita * prezzo;
    double sconto;
    if (prezzoBase > 1000) sconto = 0.95;
    else sconto = 0.98;
    return prezzoBase * sconto;
}
```

- Diventa, sostituendo entrambe le variabili `prezzoBase` e `sconto`

```
private double quantita, prezzo;

public double getPrezzo2() {
    return prezzoBase() * sconto();
}

private double prezzoBase() {
    return quantita * prezzo;
}

private double sconto() {
    if (prezzoBase() > 1000) return 0.95;
    return 0.98;
}
```

13

Prof. Tramontana - Marzo 2019

## Dividi Var Temp: Motivazioni

- Le variabili temporanee (`temp`) hanno vari usi. Alcuni usi portano ad assegnare `temp` più volte, es. variabili che cambiano ad ogni passata di un ciclo (queste assegnazioni multiple sono ok)
- Variabili `temp` che tengono il risultato di un metodo lungo, per essere usate dopo, dovrebbero essere assegnate una volta soltanto
- Se sono assegnate più di una volta allora hanno più di una responsabilità all'interno del metodo. Ogni variabile dovrebbe avere una sola responsabilità e dovrebbe essere sostituita con una `temp` per ciascuna responsabilità
- Usare una stessa `temp` per due cose diverse confonde il lettore

15

Prof. Tramontana - Marzo 2019

## (3) Dividi Variabile Temp

- Una variabile temporanea è assegnata più di una volta, ma non è una variabile assegnata in un loop o usata per accumulare valori

- Usare una variabile separata per ciascun assegnamento

```
double temp = 2 * (height + width);
System.out.println(temp);
temp = height * width;
System.out.println(temp);
```

- Diventa

```
final double perim = 2 * (height + width);
System.out.println(perim);
final double area = height * width;
System.out.println(area);
```

14

Prof. Tramontana - Marzo 2019

## Dividi Var Temp: Meccanismi

- Cambiare il nome della variabile `temp` al momento della dichiarazione e alla sua prima assegnazione
- Dichiarare la nuova `temp` come `final`
- Cambiare tutti i riferimenti a `temp` fino alla sua seconda assegnazione
- Dichiarare una nuova `temp` per la seconda assegnazione
- Compilare e testare
- Ripetere per singoli passi, ogni passo rinomina una dichiarazione e cambia i riferimenti fino alla prossima assegnazione

16

Prof. Tramontana - Marzo 2019

# Esempio Dividi Var Temp

```
private double primaryForce, secondaryForce, mass, delay;

public double getDistanceTravelled1(int time) {
    double result;
    double acc = primaryForce / mass; // prima assegnazione
    int primaryTime = (int) Math.min(time, delay);
    result = 0.5 * acc * primaryTime * primaryTime;
    int secondT = (int) (time - delay);
    if (secondT > 0) {
        double primaryVel = acc * delay;
        acc = (primaryForce + secondaryForce) / mass; // seconda assegnazione
        result += primaryVel * secondT + 0.5 * acc * secondT * secondT;
    }
    return result;
}
```

- Diventa

```
public double getDistanceTravelled2(int time) {
    double result;
    final double primAcc = primaryForce / mass;
    int primaryTime = (int) Math.min(time, delay);
    result = 0.5 * primAcc * primaryTime * primaryTime;
    int secondT = (int) (time - delay);
    if (secondT > 0) {
        double primaryVel = primAcc * delay;
        final double secondAcc = (primaryForce + secondaryForce) / mass;
        result += primaryVel * secondT + 0.5 * secondAcc * secondT * secondT;
    }
    return result;
}
```