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: eliminare frammenti di codice duplicati all'interno di rami condizionali
- Per l'esempio visto prima, si è passati dalla versione 0.2 alla versione 0.9 usando la tecnica di Refactoring **Estrai Metodo**
- La versione 1.2, eliminando due attributi, previene eventuali effetti collaterali (ovvero non voluti) di altre operazioni

Refactoring

- L'idea del refactoring è di riconoscere che è difficile ottenere fin dall'inizio una progettazione adeguata e quindi si deve cambiare. La nuova progettazione consente di incorporare più facilmente nuovi requisiti
- 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 manutenere
  - Si evita di introdurre un **debito tecnico** all'interno della progettazione

Perché Fare Refactoring

- Perché fare refactoring: per migliora la progettazione, senza refactoring la progettazione si deteriora mano a mano che si apportano modifiche al codice
- Il Refactoring rende il codice più facile da capire, perché la struttura del codice viene migliorata, il codice duplicato rimosso, etc. **Aiuta a scoprire difetti (bug):** fare refactoring promuove la comprensione profonda del codice. Permette di programmare più velocemente, perché una buona progettazione è più facile da cambiare
- Come si fa refactoring: usando le tecniche di refactoring. Dopo aver fatto refactoring, 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 intervenire

Sintomi

- Per capire se è necessario il refactoring, si possono controllare vari sintomi (**bad smell**, o code smell)
  - Codice duplicato. Usare tecnica Estrai Metodo
  - Metodi lunghi o con commenti all'interno. Usare tecnica Estrai Metodo (insieme a Sostituisci Temp con Query)
  - Classe di grandi dimensioni: tiene tante istanze e/o ha tante linee di codice. Usare tecnica Estrai Classe
  - Lunga lista di parametri per un metodo. Usare tecnica Introduci Oggetto Parametro
  - Cambiamenti divergenti: cambiamenti a parti diverse per motivi diversi. Usare tecnica Estrai Classe

Tecnica (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 somma) {
  printBanner();
  System.out.println(“nome: ” + nome);
  System.out.println(“tot: ” + somma);
}
```

- Diventa

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


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

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 della chiamata alla variabile locale del metodo sorgente
- Sostituire nel metodo sorgente il codice estratto con una chiamata al metodo nuovo

Esempio Estrai Metodo

```
public class Ordine {
    private double importo;
    public double getImporto() {
        return importo;
    }
}
// prima del refactoring
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 { // dopo il refactoring
    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);
    }
}
```

(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;
}
```


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

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

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;
}
```

(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);
```


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

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

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;
}
```

(4) Sposta Metodo

- Un metodo usa più caratteristiche (attributi e operazioni) di un'altra classe che non di quella in cui è definito
- Creare un nuovo metodo con un corpo di metodo simile nella classe che il metodo usa maggiormente. Nel corpo del vecchio metodo invocare il nuovo metodo, oppure rimuovere il vecchio metodo
- Motivazioni
  - Due classi sono strettamente accoppiate
  - Spostare un metodo rende le responsabilità delle classi più chiare
  - Un metodo usa i metodi di un'altra classe più di quanto usa i metodi della propria classe

Sposta Metodo: Meccanismi

- Esaminare attributi e metodi usati dal metodo da spostare per decidere se spostare anche questi (spostarli se sono usati solo dal metodo trasferito)
- Controllare che il metodo da spostare non sia dichiarato anche in superclassi e sottoclassi
- Dichiarare il metodo nella classe target
- Copiare il metodo nella classe target e sistemare le chiamate a metodi della classe origine e della classe target
- Richiamare il metodo nella classe target con l'opportuno riferimento
- Decidere se rimuovere il metodo originario o tenerlo per delegare (chiamare al suo interno il nuovo metodo nell'altra classe)

(5) Estrai Classe

- Una classe fa il lavoro che dovrebbero fare due classi
  - Es. la classe Persona ha attributi per tenere prefisso e numero di telefono
- Si crea una nuova classe e si spostano alcuni attributi e metodi dalla vecchia alla nuova classe
- Motivazione
  - Una classe dovrebbe avere una responsabilità, ma durante lo sviluppo si aggiungono attributi e metodi e la classe cresce
  - Una classe grande è difficile da comprendere
  - Se un sotto-insieme di dati ed un sotto-insieme di metodi dovrebbero andare insieme, o se un sotto-insieme di dati di solito viene cambiato insieme, o tali dati dipendono tra loro, allora è bene dividere la classe
  - Un altro segno è che la sottoclasse di tale classe coinvolge solo poche caratteristiche della classe

Estrai Classe: Meccanismi

- Decidere quali responsabilità separare
- Creare una nuova classe per tali responsabilità
- Inserire una dipendenza tra la vecchia e la nuova classe: un attributo nella vecchia classe che tiene una istanza della nuova classe
- Spostare attributi e metodi dalla vecchia classe alla nuova classe, iniziando dai metodi di più basso livello