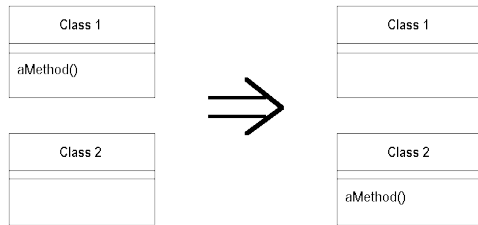


Refactoring ⑤ Sposta Metodo

- Un metodo sta usando più caratteristiche (attributi e operazioni) di un'altra classe che non quella in cui è definito
- Crea un nuovo metodo con un corpo simile nella classe che il metodo usa maggiormente. Nel corpo del vecchio metodo invoca il nuovo metodo, oppure rimuovi il vecchio metodo



E. Tramontana - Refactoring - Ott 2008

1

⑤ Sposta Metodo

- Motivazione
 - Due classi sono strettamente accoppiate
 - Spostare un metodo rende le responsabilità delle classi più chiare
 - Un metodo riferisce un altro oggetto più di quanto riferisce i metodi della propria classe
- Meccanica
 - Esaminare attributi e metodi usati dal metodo m per decidere se spostare anche questi (spostarli se sono usati solo dal metodo m)
 - Controllare che il metodo m non sia dichiarato anche in superclassi e sottoclassi
 - Dichiarare il metodo m nella classe target
 - Copiare il metodo m 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

E. Tramontana - Refactoring - Ott 2008

2

⑤ Sposta Metodo - Codice iniziale

```
class Account {
    private AccountType _type;
    private int _daysOverdrawn;
    double overdraftCharge() {
        if (_type.isPremium()) {
            double result = 10;
            if (_daysOverdrawn > 7) {
                result += (_daysOverdrawn - 7) * 0.85;
            }
            return result;
        }
        return _daysOverdrawn * 1.75;
    }
    double bankCharge() {
        double result = 4.5;
        if (_daysOverdrawn > 0)
            result += overdraftCharge();
        return result;
    }
}
```

E. Tramontana - Refactoring - Ott 2008

3

⑤ Sposta Metodo

- Considerazioni
 - Abbiamo solo due tipi di account, standard e premium, ma aggiungeremo altri tipi, ognuno che calcola l'ammontare overdraft in modo diverso
 - Spostiamo il metodo overdraftCharge() nella classe AccountType
 - overdraftCharge() richiede solo l'attributo _daysOverdrawn per i calcoli, ma questo attributo è specifico per l'account quindi deve stare nella classe account
 - Quindi _daysOverdrawn verrà passato al metodo overdraftCharge() nella classe AccountType

E. Tramontana - Refactoring - Ott 2008

4

⑤ Sposta Metodo

- Codice finale

```
class AccountType {
    ...
    double overdraftCharge(int daysOverdrawn) {
        if (isPremium()) {
            double result = 10;
            if (daysOverdrawn > 7) {
                result += (daysOverdrawn - 7) * 0.85;
            }
            return result;
        }
        return daysOverdrawn * 1.75;
    }
    ...
}
```

- Notare il parametro daysOverdrawn e la chiamata isPremium() direttamente sulla stessa istanza

E. Tramontana - Refactoring - Ott 2008

5

⑤ Sposta Metodo

- Nella classe Account

```
double overdraftCharge() {
    return _type.overdraftCharge(_daysOverdrawn);
}
```

- Oppure se non vogliamo overdraftCharge() in Account possiamo cambiare il metodo bankCharge per invocare direttamente il metodo overdraftCharge() di AccountType

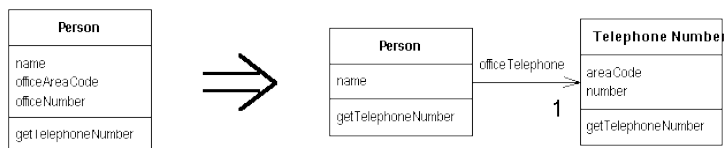
```
double bankCharge() {
    double result = 4.5;
    if (_daysOverdrawn > 0)
        result += _type.overdraftCharge(_daysOverdrawn);
    return result;
}
```

E. Tramontana - Refactoring - Ott 2008

6

Refactoring ⑥ Estrai Classe

- Una classe fa il lavoro che dovrebbero fare due classi
- Crea una nuova classe e sposta i rilevanti attributi e metodi dalla vecchia alla nuova classe



E. Tramontana - Refactoring - Ott 2008

7

⑥ Estrai Classe

- Motivazione
 - Una classe dovrebbe avere poche responsabilità, ma durante lo sviluppo si aggiungono attributi e metodi e la classe cresce
 - Una classe troppo grande è difficile da comprendere
 - Se un sottoinsieme di dati ed un sottoinsieme di metodi dovrebbero andare insieme, o se un sottoinsieme 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
- Meccanica
 - Decidere quali responsabilità separare
 - Creare una nuova classe per tali responsabilità
 - Inserire una dipendenza tra la vecchia e la nuova classe
 - Usare Sposta Attributo
 - Usare ⑤ Sposta Metodo iniziando dai metodi di più basso livello

E. Tramontana - Refactoring - Ott 2008

8

⑥ Estrai Classe

```
class Person...
  private String _name;
  private String _officeAreaCode;
  private String _officeNumber;
  public String getName() {
    return _name; }
  public String getTelephoneNumber() {
    return "(" + _officeAreaCode + ") " + _officeNumber;
  }
  String getOfficeAreaCode() {
    return _officeAreaCode; }
  void setOfficeAreaCode(String arg) {
    _officeAreaCode = arg; }
  String getOfficeNumber() {
    return _officeNumber; }
  void setOfficeNumber(String arg) {
    _officeNumber = arg; }
```

E. Tramontana - Refactoring - Ott 2008 9

⑥ Estrai Classe

- Separiamo da questa il numero di telefono
- ```
class TelephoneNumber { }
```
- Mettiamo un collegamento tra persona e numero di telefono
- ```
class Person
  private TelephoneNumber _officeTelephone = new TelephoneNumber();
```
- Spostiamo alcuni attributi ed alcuni metodi
- ```
class TelephoneNumber {
 private String _areaCode;
 String getAreaCode() { return _areaCode; }
 void setAreaCode(String arg) { _areaCode = arg; }
 ...
}
```

E. Tramontana - Refactoring - Ott 2008 10

## ⑥ Estrai Classe

```
class Person...
 public String getName() { return _name; }
 public String getTelephoneNumber() {
 return _officeTelephone.getTelephoneNumber();
 }
 TelephoneNumber getOfficeTelephone() {
 return _officeTelephone;
 }
}
```

- Quanto devono conoscere i client della nuova classe?
  - Posso nascondere completamente e fornire metodi che delegano ad essa
  - Posso restituire il riferimento all'oggetto e lasciare che i client ne modifichino i valori

E. Tramontana - Refactoring - Ott 2008 11

## Refactoring ⑦ Inline Classe

- Una classe che non fa molto può essere incorporata in un'altra classe
- E' il contrario di ⑥ Estrai Classe



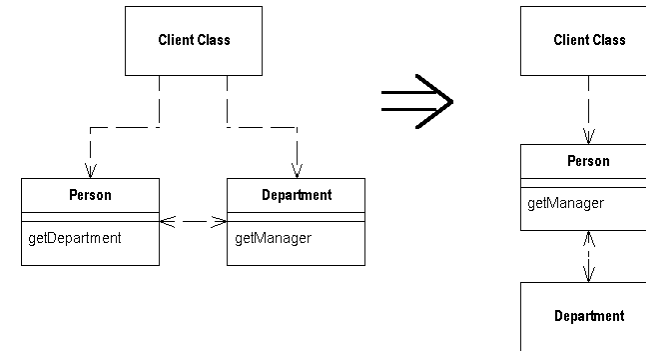
E. Tramontana - Refactoring - Ott 2008 12

## ⑦ Inline Classe

- Motivazione
  - Dopo aver applicato altri refactoring una classe risulta avere poca responsabilità
  - Inserisco tale classe nella classe che la usa di più
- Meccanica
  - Dichiarare nella classe inglobante tutti i metodi della classe da assorbire
  - Cambiare tutti i riferimenti alla classe piccola a riferimenti della classe inglobante
  - Usare ⑤ Sposta Metodo e Sposta Attributo

## Refactoring ⑧ Nascondi Delegato

- Un client chiama un delegato di un oggetto
- Creare un metodo nel server che nasconde il delegato



## ⑧ Nascondi Delegato

- Motivazioni
  - Incapsulamento significa che un oggetto dovrebbe conoscere poco di altre parti del sistema
  - Se un client chiama un metodo di un attributo di una classe server allora il client conosce il delegato
  - Quando il delegato cambia, anche il client cambia :-)
- Meccanica
  - Per ogni metodo usato sul delegato creare un metodo sul server che delega la chiamata
  - Aggiustare il client a chiamare il server

```
Person manager = john.getDepartment().getManager();
```

Diventa

```
Person manager = john.getManager();
```

## Refactoring ⑨ Introduci Metodo Straniero

- Una classe server dovrebbe avere un metodo aggiuntivo, ma non si può modificare la classe
  - Per es. è una classe di libreria
- Creare un metodo nella classe client che ha come primo argomento la classe server

```
Date newStart = new Date (previousEnd.getYear(),
 previousEnd.getMonth(), previousEnd.getDate() + 1);
```

- Diventa

```
Date newStart = nextDay(previousEnd);
private static Date nextDay(Date arg) {
 // foreign method, should be on date
 return new Date (arg.getYear(),arg.getMonth(), arg.getDate() + 1);
}
```

## ⑨ Introduci Metodo Straniero

- Motivazione
  - Un servizio utile non è disponibile
  - La classe non si può modificare
- Meccanica
  - Creare un metodo nella classe client per il servizio desiderato
  - Usare una istanza del server come primo parametro
  - Commentare il metodo come “metodo straniero, dovrebbe stare in server”

E. Tramontana - Refactoring - Ott 2008 17

## ⑩ Introduci Estensione Locale

- Una classe server dovrebbe avere alcuni metodi aggiuntivi, ma non si può modificare la classe
- Creare una classe che contiene tali metodi e rendere questa classe una sottoclasse o un wrapper della classe originale

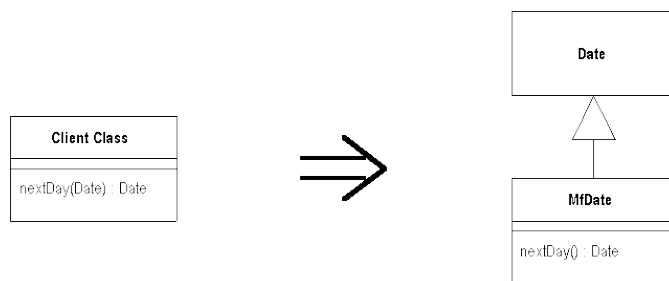
```
class myDate extends Date {
 public myDate(String dateString) { super(dateString); }
 public Date nextDay() ...
 public int dayOfYear() ...
```

- Il costruttore di myDate deve delegare al costruttore di Date
- Con il wrapper

```
class myDateWrap {
 private Date original;
 public myDateWrap(String dateString) { original = new Date(dateString); }
 public int getYear() { return original.getYear(); } // etc.
 public Date nextDay() ...
```

E. Tramontana - Refactoring - Ott 2008 18

## ⑩ Introduci Estensione Locale



E. Tramontana - Refactoring - Ott 2008 19