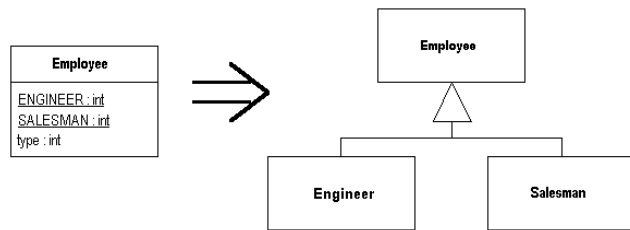


## Sostituisci Codice Tipo con Sottoclasse

- Il codice immutabile del tipo ha effetti sul comportamento della classe
- Sostituisci il codice del tipo con sottoclassi



E. Tramontana - Refactoring - Nov 2008 1

## Sostituisci Codice Tipo con Sottoclasse

Codice iniziale

```
class Employee ..
    private int _type;
    static final int ENGINEER = 0;
    static final int SALESMAN = 1;
    static final int MANAGER = 2;
    Employee (int type) {
        _type = type;
    }
    int payAmount() {
        switch (_type) {
            case ENGINEER: return _monthlySalary;
            case SALESMAN: return _monthlySalary + _commission;
            case MANAGER: return _monthlySalary + _bonus;
            default: throw new RuntimeException("Incorrect Employee");
        }
    }
}
```

E. Tramontana - Refactoring - Nov 2008 3

## Sostituisci Codice Tipo con Sottoclasse

- Motivazione
  - Il codice ha costrutti condizionali che testano il valore del codice per il tipo ed eseguono codice differente a seconda del risultato
  - Solo alcune caratteristiche sono rilevanti per oggetti con un certo codice
- Meccanica
  - Incapsulare il codice del tipo, se il codice del tipo è passato al costruttore sostituire il costruttore con un metodo factory
  - Creare una sottoclasse per ogni valore del codice del tipo. Fare override del metodo get per il tipo di codice
  - Eliminare l'attributo che tiene il codice del tipo dalla superclasse. Dichiarare come abstract il metodo per l'accesso

E. Tramontana - Refactoring - Nov 2008 2

## Sostituisci Codice Tipo con Sottoclasse

Inserisco metodo factory

```
class Employee
    static Employee create(int type) {
        return new Employee(type);
    }
```

Creo sottoclasse

```
class Engineer extends Employee {
    int getType() {
        return Employee.ENGINEER;
    }
}
```

Modifico metodo factory

```
class Employee
    static Employee createEngineer() {
        return new Engineer();
    }
}
```

E. Tramontana - Refactoring - Nov 2008 4

## Sostituisci Codice Tipo con Sottoclasse

Override del metodo get...

```
class Engineer extends Employee {
    int getType() {
        return Employee.ENGINEER;
    }
    int payAmount() {
        return _monthlySalary;
    }
}
```

Elimino attributi e rendo abstract i metodi

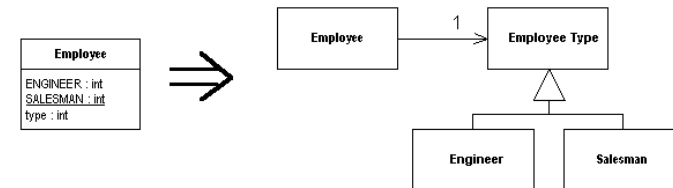
```
class Employee
    static Employee createEngineer() {
        return new Engineer();
    }
    abstract int payAmount();
    abstract int getType();
```

E. Tramontana - Refactoring - Nov 2008

5

## Sostituisci Codice Tipo con State

- Il codice del tipo ha effetti sul comportamento della classe, ma non è possibile usare sottoclassi
- Sostituisci il codice del tipo con il design pattern State



E. Tramontana - Refactoring - Nov 2008

6

## Sostituisci Codice Tipo con State

- Motivazione
  - Simile al refactoring precedente, stesse motivazioni, ma non è possibile fare sottoclassi, per es. perché esistono già sottoclassi per altre ragioni
- Meccanica
  - Incapsulare il codice del tipo
  - Creare una nuova classe, col nome in base a tale tipo, questa è la classe State
  - Aggiungere sottoclassi alla classe State per ognuno del ramo del codice esistente
  - Creare una query abstract che ritorna il tipo sulla classe State
  - Creare un attributo nella vecchia classe per la classe State
  - Aggiustare il codice nella vecchia classe in modo da delegare
  - Aggiustare il codice che gestisce tipi per gestire la singola sottoclasse

E. Tramontana - Refactoring - Nov 2008

7

## Sostituisci Codice Tipo con State

Codice iniziale

```
class Employee ..
    private int _type;
    static final int ENGINEER = 0;
    static final int SALESMAN = 1;
    static final int MANAGER = 2;
    Employee (int type) {
        _type = type;
    }
    int payAmount() {
        switch (_type) {
            case ENGINEER: return _monthlySalary;
            case SALESMAN: return _monthlySalary + _commission;
            case MANAGER: return _monthlySalary + _bonus;
            default: throw new RuntimeException("Incorrect Employee");
        }
    }
}
```

E. Tramontana - Refactoring - Nov 2008

8

## Sostituisci Codice Tipo con State

```
Inserisco classe State
abstract class EmployeeType {
    static final int ENGINEER = 0;
    ...
    abstract int getTypeCode();
}

class Engineer extends EmployeeType {
    int getTypeCode() {
        return EmployeeType.ENGINEER;
    }
    int payAmount() {
        return _monthlySalary;
    }
}
```

## Sostituisci Codice Tipo con State

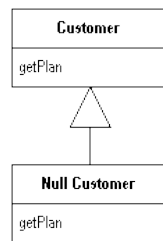
```
class Employee {
    private EmployeeType _type;
    int getType() { return _type.getTypeCode(); }
    int payAmount() { return _type.payAmount(); }
    void setType(int arg) {
        _type = EmployeeType.newType(int code);
    }
}

abstract class EmployeeType {
    ...
    static EmployeeType newType(int code) {
        switch (code) {
            case ENGINEER: return new Engineer();
        }
    }
}
```

## Refactoring Introduci Oggetto null

- Si hanno nel codice vari confronti con un valore null
- Invece di restituire un valore null da un metodo findCustomer() restituire una istanza di un oggetto

```
Customer customer = findCustomer(...);
...
if (customer == null) {
    name = "occupant"
} else {
    name = customer.getName()
}
if (customer == null) {
    ...
```



## Introduci Oggetto null

- Introdurre

```
public class NullCustomer extends Customer {
    public String getName() {
        return "occupant";
    }
}
```

```
Customer customer = findCustomer(...);
name = customer.getName();
```

- La condizione scompare