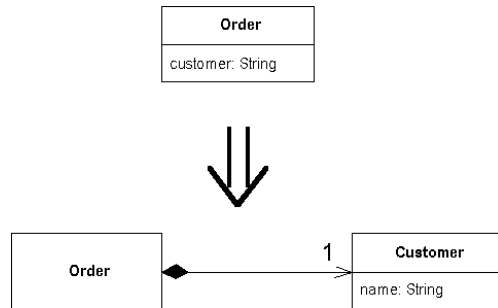


Refactoring Sostituisci Valore con Oggetto

- Un item (attributo) ha bisogno di dati aggiuntivi o di comportamenti
- Cambia l'item in un oggetto



E. Tramontana - Refactoring - Ott 2008 1

Sostituisci Valore con Oggetto

- Motivazione
 - Durante lo sviluppo si capisce che un dato semplice non è più così semplice
 - Es. un numero di telefono rappresentato come una stringa ha bisogno del prefisso e di codice per formattarlo
 - Cambia il dato in un oggetto
- Meccanica
 - Crea una classe per il valore. Crea un attributo final dello stesso tipo che aveva il valore nella classe originaria. Aggiungi un metodo getter ed il costruttore che prende il valore
 - Nella classe sorgente cambia il tipo dell'attributo al tipo della classe creata
 - Nella classe sorgente usa il metodo getter nella classe creata
 - Se il costruttore della classe sorgente usa il valore, assegnalo usando il costruttore della nuova classe
 - Cambia il metodo setter per usare la nuova classe (nuova istanza)

E. Tramontana - Refactoring - Ott 2008 2

Sostituisci Valore con Oggetto

```
Codice iniziale
class Order...
    private String _customer;
    public Order(String customer) {
        _customer = customer;
    }
    public String getCustomer() {
        return _customer;
    }
    public void setCustomer(String arg){
        _customer = arg;
    }
}
```

```
Creazione classe Customer
class Customer {
    private final String _name;
    public Customer(String name){
        _name = name;
    }
    public String getName() {
        return _name;
    }
}
```

E. Tramontana - Refactoring - Ott 2008 3

Sostituisci Valore con Oggetto

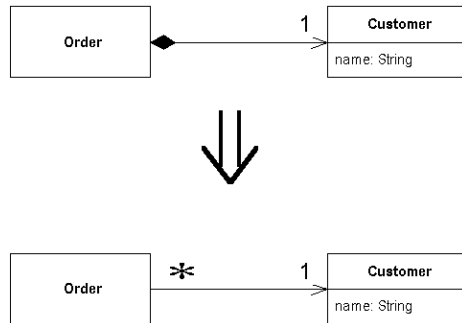
```
Codice finale
class Order...
    private Customer _customer;
    public Order (String customer) {
        _customer = new Customer(customer);
    }
    public String getCustomer() {
        return _customer.getName();
    }
    public void setCustomer(String arg){
        _customer = new Customer(arg);
    }
}
```

- Il metodo setter crea una nuova istanza di customer poiché l'attributo stringa rappresentava un valore, quindi anche per l'oggetto customer ho un valore

E. Tramontana - Refactoring - Ott 2008 4

Refactoring Cambia Valore in Riferimento

- Hai una classe con molte istanze uguali che vuoi sostituire con un singolo oggetto
- Cambia l'oggetto in un oggetto riferimento



E. Tramontana - Refactoring - Ott 2008

5

Cambia Valore in Riferimento

- Motivazione
 - Gli oggetti riferimento sono oggetti che rappresentano qualcosa del mondo reale (come cliente o account) e per verificare se due oggetti sono uguali si usa l'identità (riferimento) dell'oggetto
 - Gli oggetti valore sono definiti in modo completo dal valore dei loro dati (per es. date o denaro). Non importa quante copie ci sono: tanti oggetti possono contenere lo stesso valore. Per confrontare due oggetti occorre confrontare i loro valori e fare override del metodo equals
- Meccanica
 1. Usare Sostituisci Costruttore con Factory Method
 2. Decidere chi fornisce accesso all'oggetto
 3. Decidere se gli oggetti sono creati inizialmente o quando servono
 4. Cambiare il Factory Method in modo da restituire l'oggetto riferimento

E. Tramontana - Refactoring - Ott 2008

6

Cambia Valore in Riferimento

- La classe Customer è usata dalla classe Order

Codice iniziale

```
class Order...
    private Customer _customer;
    public Order(String customerName) {
        _customer = new Customer(customerName);
    }
    public String getCustomerName() {
        return _customer.getName();
    }
    public void setCustomer(String customerName){
        _customer = new Customer(customerName);
    }
}
```

- Al momento Customer è un oggetto valore. Ogni ordine ha il suo oggetto Customer sebbene siano per lo stesso cliente reale
- Dovrebbe esserci un solo oggetto Customer per un certo nome del cliente

E. Tramontana - Refactoring - Ott 2008

7

Richiamo su design pattern Factory Method

- Il design pattern Factory Method permette di incapsulare decisioni su istanziazioni di oggetti
 - Soluzione: il metodo Factory è un metodo di un ConcreteCreator che restituisce una istanza di una classe ConcreteProduct
 - Il metodo Factory decide: (i) quale sottoclasse usare per restituire l'istanza richiesta; (ii) se creare l'istanza o riusarne una creata
 - Conseguenze: il client che invoca il metodo Factory sa solo che l'istanza sarà di tipo Product, quindi si lega all'interfaccia Product e non all'implementazione ConcreteProduct

```
Product p = CCreator.getProduct();
```

```
public class CCreator {
    public static Product getProduct() {
        return new ConcreteProduct();
    }
}
```

E. Tramontana - Refactoring - Ott 2008

8

Variante Factory Method

- Variante del design pattern: il metodo Factory è un metodo static della classe di cui si vuole una istanza

```
Product p = Product.getProduct();
public class Product {
    public static Product getProduct() { return new Product(); }
}
```
- Vantaggi di questa variante rispetto all'uso del costruttore:
 - Il metodo Factory (getProduct) ha un nome significativo e posso avere tanti metodi Factory con la stessa signature (getNewProduct, getProductByName, etc.)
 - Non è richiesto creare una istanza ad ogni invocazione del metodo Factory. La classe può gestire le proprie istanze ed i client sono liberi da questa responsabilità

```
public static Product getProduct() { return existProduct; }
```
 - Il metodo Factory può restituire una istanza di una sottoclasse return new CProduct();
 - Riduce la lista di parametri passati, il nome del metodo può indicare il valore dei parametri

```
public static Product getToyProduct() { return new Product(0, 10, -1); }
```
 - Riduce la specifica lunga per tipi parametrizzati, il metodo li specifica al suo interno

```
public static Dictionary getX() { return new Hashtable<String, Order>();}
```

E. Tramontana - Refactoring - Ott 2008 9

Cambia Valore in Riferimento

- Passo 1: Usare Sostituisci Costruttore con Factory Method
- Codice dopo aver creato il Factory Method

```
class Order...
    private Customer _customer;
    public Order(String customerName) {
        _customer = Customer.create(customerName);
    }
}
```

```
class Customer {
    public static Customer create(String name){
        return new Customer(name);
    }
    private Customer(String name){
        _name = name;
    }
}
```

E. Tramontana - Refactoring - Ott 2008 10

Cambia Valore in Riferimento

- Passo 2: Decidere chi fornisce gli oggetti Customer
- Uso un attributo statico nella classe Customer

```
class Customer...
    private static Dictionary _instances = new Hashtable();
```

- Passo 3: Decidere se gli oggetti sono creati inizialmente o quando servono
- Creo gli oggetti Customer a priori e li inserisco nella hashtable

```
public static void loadCustomers() {
    new Customer("Lemon Car Hire").store();
    new Customer("Orange Coffee Machine").store();
}
```

```
private void store() {
    _instances.put(this.getName(), this);
}
```

E. Tramontana - Refactoring - Ott 2008 11

Cambia Valore in Riferimento

- Passo 4: Cambiare il Factory Method in modo da restituire l'oggetto riferimento
- Restituisco un cliente già creato, selezionando l'oggetto in base al nome cliente.
- Rinomino il metodo Factory create per far capire che sono già stati creati

```
public static Customer getNamed(String name) {
    return (Customer) _instances.get(name);
}
```

E. Tramontana - Refactoring - Ott 2008 12

Codice finale

```
public class Customer {
    private static Dictionary<String, Customer> _instances =
        new Hashtable<String, Customer>();
    private String name;

    private Customer(String n) { name = n; }
    public String getName() { return name; }
    public static void loadCustomers() {
        new Customer("Lemon Car Hire").store();
        new Customer("Orange Coffee Machine").store();
    }
    private void store() {
        _instances.put(this.getName(), this);
    }

    public static Customer getNamed(String name) {
        return (Customer) _instances.get(name);
    }
}
```

E. Tramontana - Refactoring - Ott 2008 13

Codice finale

```
public class Order {
    private Customer _customer;
    public Order(String customerName) {
        _customer = Customer.getNamed(customerName);
    }
}

public class Test {
    public static void main(String[] args) {
        Customer.loadCustomers();
        Order o = new Order("Lemon Car Hire");
    }
}
```

E. Tramontana - Refactoring - Ott 2008 14

Cambia Valore in Riferimento

- Considerazioni sulla soluzione introdotta nella classe Customer
 - L'attributo static dictionary evita di avere una classe che tiene solo l'insieme dei riferimenti a Customer, e di tenere l'insieme di riferimenti in una classe che non ha niente a che fare con Customer
 - La creazione delle istanze di Customer è fatta nella stessa classe e tale classe può gestire bene il tempo di creazione di istanze, l'accesso alle istanze, il numero di istanze da creare, etc.
 - Il metodo Factory getNamed: (i) ha un nome significativo, (ii) fornisce un punto di accesso per le istanze, (iii) è molto più flessibile dell'uso di new