Refactoring – Patterns

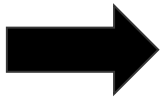
Gruppe	Fokus	Beispiele
Composing Methods	Struktur/Schnitt von Methoden	• Extract Method • Extract Variable • Replace Temp with Query
Moving Features between Objects	Verschieben von Funktionalität zwischen Klassen	• Move Method • Move Field • Extract Class
Organizing Data	Verwaltung/Kapselung von Daten	• Encapsulate Field • Replace Magic Number
Simplifying Conditional Expressions	Vereinfachen von logischen Ausdrücken	• Decompose Conditional
Simplifying Method Calls	Vereinfachen von Methodenaufrufen	• Rename Method • Add Parameter
Dealing with Generalization	Erzeugen von Vererbungshierarchien Verschieben von Funktionalität zwischen Vererbungsstrukturen	• Pull Up Field • Pull Up Method • Extract Superclass

Refactoring – Beispiele

- Encapsulate Field

Indikator	Problem	Lösung
Feld besitzt public Modifier	• Fehlende Kapselung • Direkte Manipulation von außen	• Feld wird private • Klasse stellt Accessors bereit

```
public class Person {
  public String name;
}
```



```
public class Person {
  private String name;

  public void setName(String name) {
     this.name = name;
  }

  public String getName() {
     return name;
  }
}
```

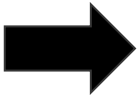
Refactoring – Beispiele

- Replace Temp With Query

Indikator	Problem	Lösung
Temporäre Variable speichert das Ergebnis einer Berechnung, wird jedoch öfter benötigt	Temporäre Variablen sind nur in eingeschränktem Kontext verfügbar → Code Duplication	Verschieben des Ausdrucks in eine eigene Methode

```
public double calculatePrice(Order order) {
   double taxFee = order.getPrice() * 0.2;
   if(order.getPrice() > 10.000) {...}
   ...
}

public void printBill(Order order) {
  double taxFee = order.getPrice() * 0.2;
  print(order.price);
  print(order.taxFee);
  print(order.price + order.taxFee);
}
```



```
public double calculatePrice(Order order) {
   double taxFee = calculateTaxFee(order.getPrice());
   if(order.getPrice() > 10.000) {...}
   ...
}

public void printBill(Order order) {
  double taxFee = calculateTaxFee(order.getPrice());
  print(order.price);
  print(order.taxFee);
  print(order.price + order.taxFee);
}


private double calculateTaxFee(double price) {
   return price * 0.2;
}
```

Refactoring – Beispiele

- Extract Method

Indikator	Problem	Lösung
• Lange Methode • Kommentare erforderlich um Methode zu verstehen	• Fehlende Verständlichkeit • Komplexe Logik • Keine Wiederverwendbarkeit	• Auslagern in eigene Methode(n) • Code an alter Stelle wird durch Methodenaufruf ersetzt

```
public void processData (String
pathIn) {

  //read data
  File inputFile = ...
  List<Person> dataList = ...

  //sort data
  for(Person p: dataList ) {...}
}
```

```
public void processData(String pathIn) {
  List<Person> dataList = readData(pathIn);
  List<Person> sortedList = sortData(dataList);
}

private List<Person> readData(String pathIn)
{...}

private List<Person> sortData(List data) {...}
```

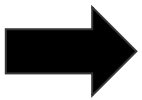
Refactoring – Beispiele

- Move Method

Indikator	Problem	Lösung
• Eine Methode wird von einer anderen Klasse häufiger verwendet als von der Klasse selbst • Eine Methode verwendet mehrere Funktionen einer anderen Klasse	• Hohe Kopplung zwischen Klassen (Abhängigkeit!) • Klasse hat zu viele Verantwortlichkeiten	• Verschieben der Methode

```
public class Product {

  public double calculatePrice(Order
o) {
     if(o.isInternationalOrder())
{...}
     else if(o.type == OrderType.A)
{...}
     ...
  }
}
```



```
public class Order {
    private boolean internationalOrder;
    private OrderType type;

    public double calculatePrice() {
        if(internationalOrder) {...}
        else if(type == OrderType.A) {...}
        ...
    }
}
```

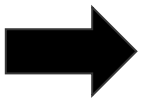
Refactoring – Beispiele

- Extract Class

Indikator	Problem	Lösung
• Eine Klasse hat mehr als eine Verantwortlichkeit • Umfangreiche Klasse mit vielen Daten • Subset an Daten ändert sich immer gemeinsam	• Verständlichkeit des Codes sinkt • Wiederverwendbarkeit eingeschränkt • Vererbung schwer möglich	• Herauslösen von Funktionalität in separate Klasse • Herstellen einer Relation zwischen den Klassen, sofern erforderlich

```
public class Person {
  private String name;
  ...
  private String street;
  private String zipCode;
  private String city;
  private Country country;

  public String printAddress() {
...}
}
```



```
public class Person {
  private String name;
  ...
  private Address address;
}

public class Address {
  private String street;
  private String zipCode;
  private String city;
  private Country country;

   public String printAddress() {...}
}
```