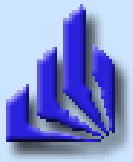
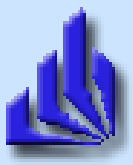
Refactoring – eine Einführung

Was ist Refactoring?
Katalog von Transformationen, Einsatzszenario,
Transformationen aus dem Bereich
„Methoden zusammenstellen“

Ein Vortrag im Rahmen des Seminars
„**Refactoring in eXtreme Programming**“
von Dietrich Travkin



Übersicht



1. Was ist Refactoring?

1.1 Definition

Refactoring (engl. für Refaktorisieren bzw. Refaktorisierung)

Schrittweise, systematische Transformation von objektorientierter Software mit Beibehaltung des beobachtbaren Verhaltens und den Zielen:

- **Lesbarkeit des Codes zu erhöhen**
- **Weiterentwicklung zu vereinfachen**

Refaktorisierung = eine Transformation
Refaktorisieren = eine Reihe von Transformationen anwenden

Verhalten sichern durch umfassende Tests.

1.2. Ziele

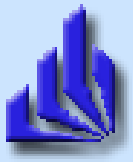
Softwareentwicklung vereinfachen durch:

- Code verständlicher machen
- Softwarestruktur verbessern

- Fehler minimieren
- Fehlersuche erleichtern

- Programmiervorgang beschleunigen

- **Qualität und Wert der Software steigern**



Übersicht

2.1. Vorgehen im Allgemeinen

Wann refaktorisieren?

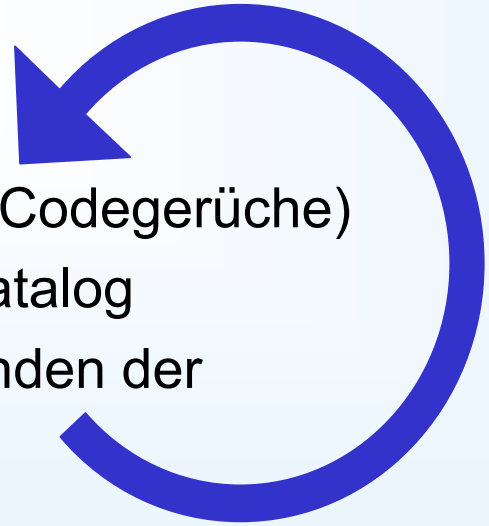
- während des Programmiervorgangs
- vor dem Hinzufügen neuer Funktionalität

Vor dem Refaktorisieren:

Implementieren einer soliden Menge von Testfällen

Der Ablauf:

1. Identifikation der Anwendungsstelle (Codegerüche)
2. Wahl der Transformation aus dem Katalog
3. Systematisches, schrittweises Anwenden der Transformation und Testen



2.2. Codegerüche

Hinweise auf schlechten, zu verbessernden Code
→ Anwendungsstellen von Refaktorisierungen

Das Buch „Refactoring“ enthält:

- Katalog mit 22 Codegeruch-Beschreibungen
- Tabelle, die jedem Geruch Refaktorisierungen zuordnet

Geruch	Refaktorisierungen
Alternative Klassen mit verschiedenen Schnittstellen, S. 80	Methode umbenennen (279) Methode verschieben (139)
Ausgeschlagenes Erbe, S. 81	Vererbung durch Delegation ersetzen (363)
Datenklassen, S. 81	Collection kapseln (211) Feld kapseln (209) Methode verschieben (139)

Die Tabelle hilft, schnell eine passende Transformation zu finden.

2.2. Codegerüche – Beispiele

Duplizierter Code

- Redundanz erschwert die Weiterentwicklung

➢ Doppelten Code zu einer Methode zusammenfassen

Lange Methode

- schwer zu verstehen, viele Kommentare nötig

➢ lange Methoden in mehrere kurze mit klarem Namen aufspalten

Kommentare

- oft Deodorant für übel riechenden, unverständlichen Code

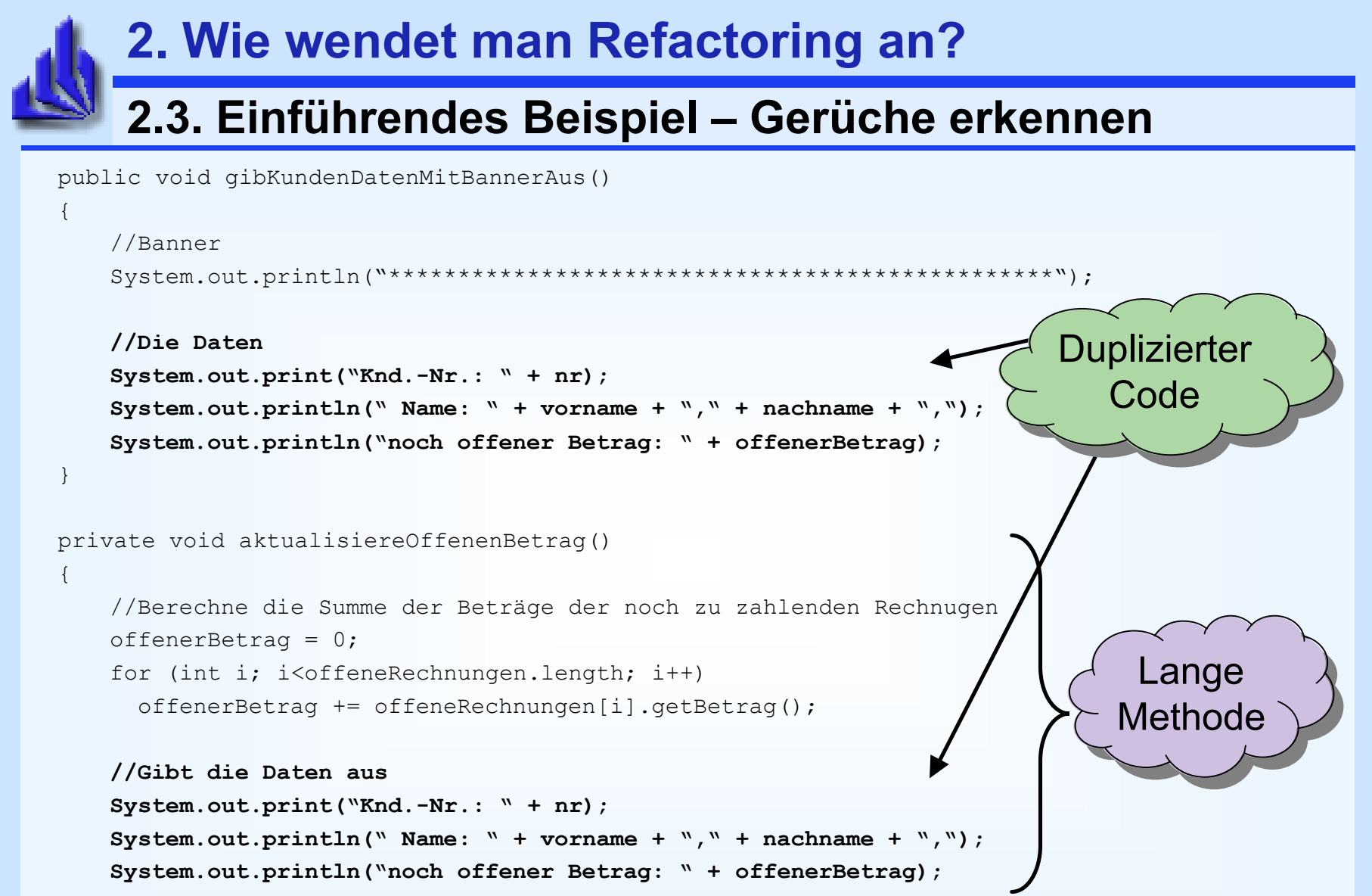
➢ refaktorisieren, bis Kommentare unnötig sind

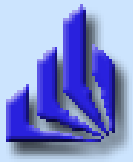
➢ Namen selbstbeschreibend wählen

Gute Kommentare begründen, warum man etwas tut
→ hilfreich für spätere Weiterentwicklung

2.3. Einführendes Beispiel – Gerüche erkennen

```
public void gibKundenDatenMitBannerAus()
{
    //Banner
    System.out.println("***********************************************");

    //Die Daten
    System.out.print("Knd.-Nr.: " + nr);
    System.out.println(" Name: " + vorname + "," + nachname + ",");
    System.out.println("noch offener Betrag: " + offenerBetrag);
}

private void aktualisiereOffenenBetrag()
{
    //Berechne die Summe der Beträge der noch zu zahlenden Rechnugen
    offenerBetrag = 0;
    for (int i; i<offeneRechnungen.length; i++)
      offenerBetrag += offeneRechnungen[i].getBetrag();

    //Gibt die Daten aus
    System.out.print("Knd.-Nr.: " + nr);
    System.out.println(" Name: " + vorname + "," + nachname + ",");
    System.out.println("noch offener Betrag: " + offenerBetrag);
}
```





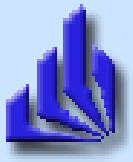
2. Wie wendet man Refactoring an?

2.3. Einführendes Beispiel – Methode extrahieren

```
public void gibKundenDatenMitBannerAus()
{
    //Banner
    System.out.println("************************************************");

    //Die Daten
    System.out.print("Knd.-Nr.: " + nr);
    System.out.println(" Name: " + vorname + "," + nachname + ",");
    System.out.println("noch offener Betrag: " + offenerBetrag);
}

private void aktualisiereOffenenBetrag()
{
    ...
}

public void gibKundenDatenAus()
{

}
```

Anwenden der Transformation

1. **Neue Methode mit selbstbeschreibendem Namen erstellen**



2. Wie wendet man Refactoring an?

2.3. Einführendes Beispiel – Methode extrahieren

```
public void gibKundenDatenMitBannerAus()
{
    //Banner
    System.out.println("************************************************");

    //Die Daten
    System.out.print("Knd.-Nr.: " + nr);
    System.out.println(" Name: " + vorname + "," + nachname + ",");
    System.out.println("noch offener Betrag: " + offenerBetrag);
}

private void aktualisiereOffenenBetrag()
{
    ...
}

public void gibKundenDatenAus()
{
    //Die Daten
    System.out.print("Knd.-Nr.: " + nr);
    System.out.println(" Name: " + vorname + "," + nachname + ",");
    System.out.println("noch offener Betrag: " + offenerBetrag);
}
```

Anwenden der Transformation

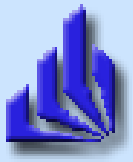
1. Neue Methode mit selbstbeschreibendem Namen erstellen
2. **Code in die neue Methode kopieren**

2.3. Einführendes Beispiel – Methode extrahieren

```
public void gibKundenDatenMitBannerAus()
{
    //Banner
    System.out.println("************************************************");

    //Die Daten
    System.out.print("Knd.-Nr.: " + nr);
    System.out.println(" Name: " + vorname + "," + nachname + ",");
    System.out.println("noch offener Betrag: " + offenerBetrag);
}

private void aktualisiereOffenenBetrag()
{
    ...
}

public void gibKundenDatenAus()
{
    //Die Daten
    System.out.print("Knd.-Nr.: " + nr);
    System.out.println(" Name: " + vorname + "," + nachname + ",");
    System.out.println("noch offener Betrag: " + offenerBetrag);
}
```

Anwenden der Transformation

1. Neue Methode mit selbstbeschreibendem Namen erstellen
2. Code in die neue Methode kopieren
3. **Nach ungültigen Referenzen suchen**



2. Wie wendet man Refactoring an?

2.3. Einführendes Beispiel – Methode extrahieren

```
public void gibKundenDatenMitBannerAus()
{
    //Banner
    System.out.println("************************************************");

    //Die Daten
    System.out.print("Knd.-Nr.: " + nr);
    System.out.println(" Name: " + vorname + "," + nachname + ",");
    System.out.println("noch offener Betrag: " + offenerBetrag);
}

private void aktualisiereOffenenBetrag()
{
    ...
}

public void gibKundenDatenAus()
{
    //Die Daten
    System.out.print("Knd.-Nr.: " + nr);
    System.out.println(" Name: " + vorname + "," + nachname + ",");
    System.out.println("noch offener Betrag: " + offenerBetrag);
}
```

Anwenden der Transformation

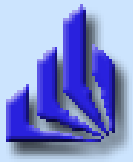
1. Neue Methode mit selbstbeschreibendem Namen erstellen
2. Code in die neue Methode kopieren
3. Nach ungültigen Referenzen suchen
4. **Code compilieren**

2.3. Einführendes Beispiel – Methode extrahieren

```
public void gibKundenDatenMitBannerAus()
{
    //Banner
    System.out.println("************************************************");

    gibKundenDatenAus();
}

private void aktualisiereOffenenBetrag()
{
    ...
}

public void gibKundenDatenAus()
{
    //Die Daten
    System.out.print("Knd.-Nr.: " + nr);
    System.out.println(" Name: " + vorname + "," + nachname + ",");
    System.out.println("noch offener Betrag: " + offenerBetrag);
}
```

Anwenden der Transformation

5. **Extrahierten Code durch Aufruf der neuen Methode ersetzen**



2. Wie wendet man Refactoring an?

2.3. Einführendes Beispiel – Methode extrahieren

```
public void gibKundenDatenMitBannerAus()
{
    //Banner
    System.out.println("************************************************");

    gibKundenDatenAus();
}

private void aktualisiereOffenenBetrag()
{
    ...
}

public void gibKundenDatenAus()
{
    //Die Daten
    System.out.print("Knd.-Nr.: " + nr);
    System.out.println(" Name: " + vorname + "," + nachname + ",");
    System.out.println("noch offener Betrag: " + offenerBetrag);
}
```

Anwenden der Transformation

5. Extrahierten Code durch Aufruf der neuen Methode ersetzen
6. **Code compilieren und testen**

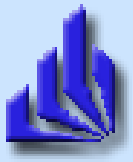
2.3. Einführendes Beispiel – Methode extrahieren

```
public void gibKundenDatenMitBannerAus()
{
    //Banner
    System.out.println("***********************************************");

    gibKundenDatenAus();
}

private void aktualisiereOffenenBetrag()
{
    //Berechne die Summe der Beträge der noch zu zahlenden Rechnugen
    offenerBetrag = 0;
    for (int i; i<offeneRechnungen.length; i++)
      offenerBetrag += offeneRechnungen[i].getBetrag();

    //Gibt die Daten aus
    System.out.print("Knd.-Nr.: " + nr);
    System.out.println(" Name: " + vorname + "," + nachname + ",");
    System.out.println("noch offener Betrag: " + offenerBetrag);
}

public void gibKundenDatenAus()
{   ...
}
```

Anwenden der Transformation

5. Extrahierten Code durch Aufruf der neuen Methode ersetzen
6. Code compilieren und testen

Wegen doppeltem Code auch hier neue Methode aufrufen!



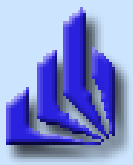
2. Wie wendet man Refactoring an?

2.3. Einführendes Beispiel – Methode extrahieren

```
public void gibKundenDatenMitBannerAus()
{
    //Banner
    System.out.println("***********************************************");

    gibKundenDatenAus();
}

private void aktualisiereOffenenBetrag()
{
    //Berechne die Summe der Beträge der noch zu zahlenden Rechnugen
    offenerBetrag = 0;
    for (int i; i<offeneRechnungen.length; i++)
      offenerBetrag += offeneRechnungen[i].getBetrag();

    gibKundenDatenAus();
}

public void gibKundenDatenAus()
{
    ...
}
```

Anwenden der Transformation

5. Extrahierten Code durch Aufruf der neuen Methode ersetzen
6. **Code compilieren und testen**



2.3. Einführendes Beispiel – Zwischenbilanz

Codegerüche als Hinweise auf schlechten Code

Methode extrahieren = wichtigste Transformation des Kapitels „Methoden zusammenstellen“

Transformation systematisch mit kleinen einfachen Schritten

Schnelle Durchführung

Testen sehr wichtig

2.4. Testen

Tests sichern **Aufrecherhaltung des beobachtbaren Verhaltens**

→beim Refaktorisieren unverzichtbar!

Testausführung muss **schnell** und **einfach** sein

→Tests automatisieren

Wie automatisieren?

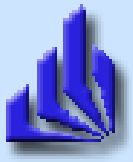
- Solide Menge von Testfällen wählen, Randbedingungen beachten
- Berechnetes mit erwartetem Ergebnis vergleichen lassen
- „O.K.“ oder Fehlermeldung zurückgeben lassen

Unterstützt vom **JUnit-Framework** von Erich Gamma und Kent Beck

Beispielausgaben:

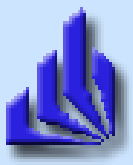
```
...
Time: 0.970

OK (3 tests)
```

```
.F
Time: 0.110

Test Results:
Run: 1 Failures: 1 Errors: 0
There was 1 failure:
1)  FileReaderTester.testRead
        expected: “m“ but was: “d“
```



Übersicht

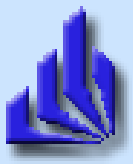
1. **Was ist Refactoring?**
   - 1.1. Definition
   - 1.2. Ziele
2. **Wie wendet man Refactoring an?**
   - 2.1. Vorgehen im Allgemeinen
   - 2.2. Codegerüche
   - 2.3. Einführendes Beispiel
   - 2.4. Testen
3. **Katalog von Transformationen**
   - 3.1. Gliederung des Katalogs
   - 3.2. Kapitel: Methoden zusammenstellen
4. **Schlussüberlegungen**
   - 4.1. Performance
   - 4.2. Probleme und Grenzen



3. Katalog von Transformationen

3.1. Gliederung des Katalogs – Aufteilung

	Anzahl Transformationen
• Methoden zusammenstellen	9
• Eigenschaften zwischen Objekten verschieben	8
• Daten organisieren	16
• Bedingte Ausdrücke vereinfachen	8
• Methodenaufrufe vereinfachen	15
• Umgang mit Generalisierung	12
• Große Refaktorisierungen	4
Insgesamt:	72



3.1. Gliederung des Katalogs – Beschreibung

Name

meist selbsterklärend und einprägsam

Anwendungssituation und Kurzbeschreibung

sehr kurz, auch in UML oder mit Beispiel-Code

Motivation

ausführlichere Begründung der Transformation

Vorgehen

systematische, exakte, schrittweise Beschreibung

Beispiele

meist mehrere, Beachtung der Sonderfälle

Methode durch Methodenobjekt ersetzen

Anwendungssituation:

Mehrere lokale Variablen in einer Methode verhindern die Anwendung von *Methode extrahieren.*

Kurzbeschreibung:

- aus der Methode eine eigene Klasse erstellen
- alle lokalen Variablen als Attribute des Objekts zu dieser Klasse verwenden
- Funktionalität der Methode an das Objekt delegieren

Motivation:

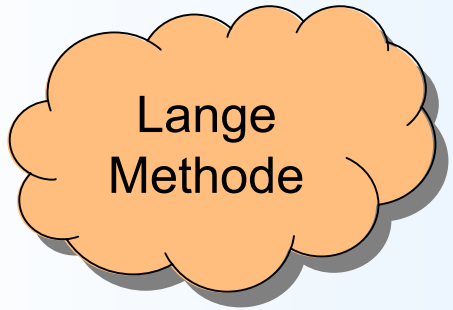
- möglichst kleine Methoden mit verständlichen Namen gestalten
- *Methode extrahieren* auch in schwierigen Situationen ermöglichen

3.2. Methode durch Methodenobjekt ersetzen

Beispiel:

Anwendung von *Methode extrahieren* hier sinnvoll, aber problematisch

```
class Kunde
{
    ...
    public int barPreisFuer(Artikel einArtikel, int menge)
    {
        int preis = menge * einArtikel.getVerkaufsPreis;
        float rabatt = getMinRabatt();
        int preisnachlass = 0;

        //Berechne gesamten Preisnachlass
        if (menge > 10)
            rabatt += 5;
        if (menge > 100)
            rabatt += 10;
        preisnachlass = (int) ( preis * rabatt / 100 );

        System.out.println("Kunde " + this + " bekommt " + rabatt + "% Rabatt.");
        return (preis - preisnachlass);
    }
}
```



3.2. Methode durch Methodenobjekt ersetzen

Beispiel:

Anwendung von *Methode extrahieren* hier sinnvoll, aber problematisch

```
class Kunde
{
    ...
    public int barPreisFuer(Artikel einArtikel, int menge)
    {
        int preis = menge * einArtikel.getVerkaufsPreis;
        float rabatt = getMinRabatt();
        int preisnachlass = 0;

        //Berechne gesamten Preisnachlass
        if (menge > 10)
            rabatt += 5;
        if (menge > 100)
            rabatt += 10;
        preisnachlass = (int) ( preis * rabatt / 100 );

        System.out.println("Kunde " + this + " bekommt " + rabatt + "% Rabatt.");
        return (preis - preisnachlass);
    }
}
```

Probleme:

- **Verwendung von Parametern**

3.2. Methode durch Methodenobjekt ersetzen

Beispiel:

Anwendung von *Methode extrahieren* hier sinnvoll, aber problematisch

```
class Kunde
{
    ...
    public int barPreisFuer(Artikel einArtikel, int menge)
    {
        int preis = menge * einArtikel.getVerkaufsPreis;
        float rabatt = getMinRabatt();
        int preisnachlass = 0;

        //Berechne gesamten Preisnachlass
        if (menge > 10)
            rabatt += 5;
        if (menge > 100)
            rabatt += 10;
        preisnachlass = (int) ( preis * rabatt / 100 );

        System.out.println("Kunde " + this + " bekommt " + rabatt + "% Rabatt.");
        return (preis - preisnachlass);
    }
}
```

Probleme:

- Verwendung von Parametern
- **Änderung mehrerer temporärer Variablen und ihre spätere Wiederverwendung**

3.2. Methode durch Methodenobjekt ersetzen

Beispiel:

```
class Kunde
{
    ...
    public int barPreisFuer(Artikel einArtikel, int menge)
    {
        int preis = menge * einArtikel.getVerkaufsPreis;
        float rabatt = getMinRabatt();
        int preisnachlass = 0;

        //Berechne gesamten Preisnachlass
        if (menge > 10)
            rabatt += 5;
        if (menge > 100)
            rabatt += 10;
        preisnachlass = (int) ( preis * rabatt / 100 );

        System.out.println("Kunde " + this + " bekommt " + rabatt + "% Rabatt.");
        return (preis - preisnachlass);
    }
}
```

Anwendung von *Methode extrahieren* hier sinnvoll, aber problematisch

Probleme:

- Verwendung von Parametern
- Änderung mehrerer temporärer Variablen und ihre spätere Wiederverwendung

Lösung: vorher *Methode durch Methodenobjekt ersetzen* auf gesamte Methode anwenden

3.2. Methode durch Methodenobjekt ersetzen

Vorgehen:

```
class Kunde
{
    ...
    public int barPreisFuer(Artikel einArtikel, int menge)
    {
        int preis = menge * einArtikel.getVerkaufsPreis;
        float rabatt = getMinRabatt();
        int preisnachlass = 0;

        //Berechne gesamten Preisnachlass
        if (menge > 10)
            rabatt += 5;
        if (menge > 100)
            rabatt += 10;
        preisnachlass = (int) ( preis * rabatt / 100 );

        System.out.println("Kunde " + this + " bekommt " + rabatt + "% Rabatt.");
        return (preis - preisnachlass);
    }
}
```

3.2. Methode durch Methodenobjekt ersetzen

Vorgehen:

1. **Neue Klasse erstellen**

```
class Kunde
{
    ...
    public int barPreisFuer(Artikel einArtikel, int menge)
    {
        int preis = menge * einArtikel.getVerkaufsPreis;
        float rabatt = getMinRabatt();
        int preisnachlass = 0;
        ...
    }
}

class BarPreisBerechner
{

}
```

3.2. Methode durch Methodenobjekt ersetzen

Vorgehen:

1. Neue Klasse erstellen
2. **Referenz auf ursprüngliches Objekt hinzufügen**

```
class Kunde
{
    ...
    public int barPreisFuer(Artikel einArtikel, int menge)
    {
        int preis = menge * einArtikel.getVerkaufsPreis;
        float rabatt = getMinRabatt();
        int preisnachlass = 0;
        ...
    }
}

class BarPreisBerechner
{
    private final Kunde kunde;
}
```

3.2. Methode durch Methodenobjekt ersetzen

Vorgehen:

1. Neue Klasse erstellen
2. Referenz auf ursprüngliches Objekt hinzufügen
3. **Parameter und temporäre Variablen als Attribute hinzufügen**

```
class Kunde
{
    ...
    public int barPreisFuer(Artikel einArtikel, int menge)
    {
        int preis = menge * einArtikel.getVerkaufsPreis;
        float rabatt = getMinRabatt();
        int preisnachlass = 0;
        ...
    }
}

class BarPreisBerechner
{
    private final Kunde kunde;
    private Artikel einArtikel;
    private int menge, preis, preisnachlass;
    private float rabatt;
}
```

3.2. Methode durch Methodenobjekt ersetzen

Vorgehen:

1. Neue Klasse erstellen
2. Referenz auf ursprüngliches Objekt hinzufügen
3. Parameter und temporäre Variablen als Attribute hinzufügen
4. **Konstruktor hinzufügen**

```
class Kunde { ...
    public int barPreisFuer(Artikel einArtikel, int menge)
    { ...  }
}

class BarPreisBerechner
{
    private final Kunde kunde;
    private Artikel einArtikel;
    private int menge, preis, preisnachlass;
    private float rabatt;

    BarPreisBerechner(Kunde iKunde, Artikel iArtikel, int iMenge)
    {
        kunde = iKunde; einArtikel = iArtikel; menge = iMenge;
    }
}
```

3.2. Methode durch Methodenobjekt ersetzen

Vorgehen:

1. Neue Klasse erstellen
2. Referenz auf ursprüngliches Objekt hinzufügen
3. Parameter und temporäre Variablen als Attribute hinzufügen
4. Konstruktor hinzufügen
5. **Methode *compute( )* erstellen**

```
class Kunde { ...
    public int barPreisFuer(Artikel einArtikel, int menge)
    { ...  }
}

class BarPreisBerechner
{
    private final Kunde kunde;
    private Artikel einArtikel;
    private int menge, preis, preisnachlass;
    private float rabatt;

    BarPreisBerechner(Kunde iKunde, Artikel iArtikel, int iMenge)
    {
        kunde = iKunde; einArtikel = iArtikel; menge = iMenge;
    }

    int compute()
    {

    }
}
```

3.2. Methode durch Methodenobjekt ersetzen

Vorgehen:

1. Neue Klasse erstellen
2. Referenz auf ursprüngliches Objekt hinzufügen
3. Parameter und temporäre Variablen als Attribute hinzufügen
4. Konstruktor hinzufügen
5. Methode *compute( )* erstellen
6. **Rumpf der Original-methode einfügen**

```
class BarPreisBerechner
{
    private final Kunde kunde;
    ...

    int compute()
    {
        int preis = menge * einArtikel.getVerkaufsPreis;
        float rabatt = kunde.getMinRabatt();
        int preisnachlass = 0;

        //Berechne gesamten Preisnachlass
        if (menge > 10)
            rabatt += 5;
        if (menge > 100)
            rabatt += 10;
        preisnachlass = (int) ( preis * rabatt / 100 );

        System.out.println("Kunde " + this + " bekommt " + rabatt + "% Rabatt.");
        return (preis – preisnachlass);
    }
}
```

3.2. Methode durch Methodenobjekt ersetzen

Vorgehen:

1. Neue Klasse erstellen
2. Referenz auf ursprüngliches Objekt hinzufügen
3. Parameter und temporäre Variablen als Attribute hinzufügen
4. Konstruktor hinzufügen
5. Methode *compute( )* erstellen
6. Rumpf der Original-methode einfügen
7. **Compilieren**

```
class BarPreisBerechner
{
    private final Kunde kunde;
    ...

    int compute()
    {
        int preis = menge * einArtikel.getVerkaufsPreis;
        float rabatt = kunde.getMinRabatt();
        int preisnachlass = 0;

        //Berechne gesamten Preisnachlass
        if (menge > 10)
            rabatt += 5;
        if (menge > 100)
            rabatt += 10;
        preisnachlass = (int) ( preis * rabatt / 100 );

        System.out.println("Kunde " + this + " bekommt " + rabatt + "% Rabatt.");
        return (preis - preisnachlass);
    }
}
```

3.2. Methode durch Methodenobjekt ersetzen

Vorgehen:

```
class Kunde
{
    ...
    public int barPreisFuer(Artikel einArtikel, int menge)
    {
        return
          new BarPreisBerechner(this, einArtikel, menge).
          compute();
    }
}

class BarPreisBerechner
{
    private final Kunde kunde;
    ...

    int compute()
    {
        ...
        return (preis - preisnachlass);
    }
}
```

1. Neue Klasse erstellen
2. Referenz auf ursprüngliches Objekt hinzufügen
3. Parameter und temporäre Variablen als Attribute hinzufügen
4. Konstruktor hinzufügen
5. Methode *compute( )* erstellen
6. Rumpf der Original-methode einfügen
7. Compilieren
8. **In der Originalmethode das neue Objekt erstellen und *compute( )* ausführen**

3.2. Methode durch Methodenobjekt ersetzen

Vorgehen:

```
class Kunde
{
    ...
    public int barPreisFuer(Artikel einArtikel, int menge)
    {
        return
          new BarPreisBerechner(this, einArtikel, menge).
          compute();
    }
}

class BarPreisBerechner
{
    private final Kunde kunde;
    ...

    int compute()
    {
        ...
        return (preis - preisnachlass);
    }
}
```

1. Neue Klasse erstellen
2. Referenz auf ursprüngliches Objekt hinzufügen
3. Parameter und temporäre Variablen als Attribute hinzufügen
4. Konstruktor hinzufügen
5. Methode *compute( )* erstellen
6. Rumpf der Original-methode einfügen
7. Compilieren
8. In der Originalmethode das neue Objekt erstellen und *compute( )* ausführen
9. **Compilieren und testen**

3.2. Methode durch Methodenobjekt ersetzen

Ergebnis:

Keine lokalen Variablen in dem zu extrahierenden Codefragment

➔ Anwendung von *Methode extrahieren* einfach

```
class Kunde { ... }

class BarPreisBerechner
{    ...
     int compute()
     {
          int preis = menge * einArtikel.getVerkaufsPreis;
          float rabatt = kunde.getMinRabatt();
          int preisnachlass = 0;

          //Berechne gesamten Preisnachlass
          if (menge > 10)
               rabatt += 5;
          if (menge > 100)
               rabatt += 10;
          preisnachlass = (int) ( preis * rabatt / 100 );

          System.out.println("Kunde " + this + " bekommt " + rabatt + "% Rabatt.");
          return (preis - preisnachlass);
     }
}
```

3.2. Methode durch Methodenobjekt ersetzen

```
class BarPreisBerechner
{   ...
    int compute()
    {
        int preis = menge * einArtikel.getVerkaufsPreis;
        float rabatt = kunde.getMinRabatt();
        int preisnachlass = 0;

        berechneGesamtenPreisnachlass();

        System.out.println("Kunde " + this + " bekommt " + rabatt + "% Rabatt.");
        return (preis - preisnachlass);
    }

    void berechneGesamtenPreisnachlass()
    {
        //Berechne gesamten Preisnachlass
        if (menge > 10)
            rabatt += 5;
        if (menge > 100)
            rabatt += 10;
        preisnachlass = (int) ( preis * rabatt / 100 );
    }
}
```

Keine lokalen Variablen in dem zu extrahierenden Codefragment

➔ Anwendung von *Methode extrahieren* einfach

Die Schritte:

1. Methode erstellen
2. Zu extrahierenden Code kopieren
3. Compilieren
4. Ursprünglichen Code durch Methodenaufruf ersetzen
5. Compilieren und testen

3.2. Methode durch Methodenobjekt ersetzen – Bilanz

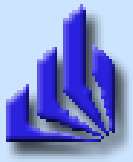
Kurze Methoden bevorzugt

Lokale Variablen erschweren Anwendung von *Methode extrahieren*

Methode durch Methodenobjekt ersetzen hilft

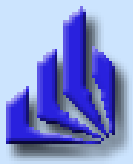
Auch hier: einfache Schritte, schnelle Durchführung, Tests

Zyklisches Refaktorisieren



Übersicht

1. **Was ist Refactoring?**
   - 1.1. Definition
   - 1.2. Ziele
2. **Wie wendet man Refactoring an?**
   - 2.1. Vorgehen im Allgemeinen
   - 2.2. Codegerüche
   - 2.3. Einführendes Beispiel
   - 2.4. Testen
3. **Katalog von Transformationen**
   - 3.1. Gliederung des Katalogs
   - 3.2. Kapitel: Methoden zusammenstellen
4. **Schlussüberlegungen**
   - 4.1. Performance
   - 4.2. Probleme und Grenzen



4. Schlussüberlegungen

4.1. Performance

Während der Entwicklung refaktorisieren.

Laufzeitoptimierung erst zum Schluss!

Nur kritische Bereiche optimieren!

Dadurch:

- schnellere Entwicklung, mehr Zeit für Optimierung
- kleinere Teile zum Tunen, übersichtlicher
- Code verständlicher

Programme nutzen meist nur kleinen Teil des Codes

→ nur 10% des Codes muss optimiert werden

4.2. Probleme und Grenzen

Finden von Referenzierungen

- Wegen dynamischer Methodenbindung, Vererbung manuelle bzw. Editor-Suche problematisch
  - erreicht oft nicht alle Referenzen
  - falsche Ersetzungen möglich

→ Programmanalyse-Werkzeuge helfen

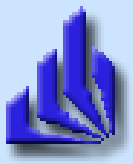
Änderung von Schnittstellen

- Öffentliche Schnittstellen (alle Benutzer erreichbar)
  - Änderungen aufwendig, aber möglich
- Veröffentlichte Schnittstellen (nicht alle Benutzer erreichbar)
  - Änderungen nicht möglich →neue Schnittstelle nutzt alte Schnittstelle

Nebenläufigkeit

- Refaktorisierungen im Katalog nur für einen Prozess entwickelt

Code zu fehlerhaft



Zusammenfassung

Refaktorisieren

vereinfacht Weiterentwicklung objektorientierten Codes

Codegerüche als Hinweis auf schlechten Code

Anwendung in schnellen, systematischen, kleinen Schritten

Absicherung durch umfassende Tests

Trotz Tests mit Bedacht refaktorisieren!

Fragen?