DHBW
Duale Hochschule
Baden-Württemberg
Karlsruhe

ERBAUER

Lars Briem

(briem.lars@googlemail.com)

Duale Hochschule Baden Württemberg - Standort Karlsruhe

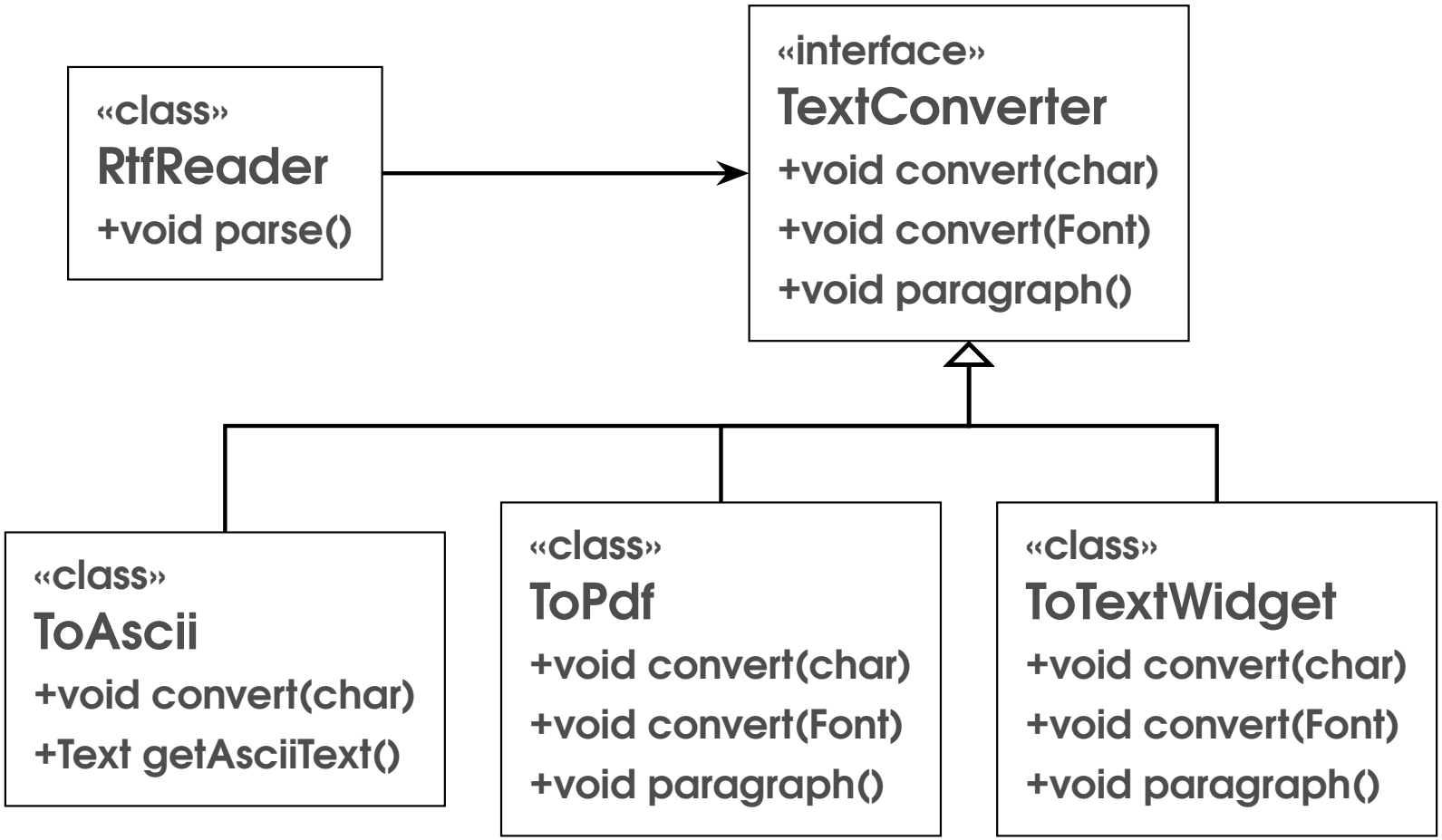
Erbauer

- Trennung der Erstellung von komplexen Objekten von ihrer Repräsentation
- Gleicher Erstellungsprozess bzw. Konstruktionsprozess kann unterschiedliche Repräsentation erzeugen
- Klassifikation
  - Objektbasiertes Erzeugungsmuster
  - Eher kurzlebig

Erbauer – Motivation

- Wiederverwendung einer komplexen Logik zur Umwandlung von Objekten
- Erzeugungslogik für verschiedene Formate von Konvertierungs- bzw. Konstruktionslogik trennen
- Schrittweise Erzeugung von komplexen Produkten
- Wiederverwendung der Erzeugungs- bzw. Konstruktionslogik unabhängig voneinander

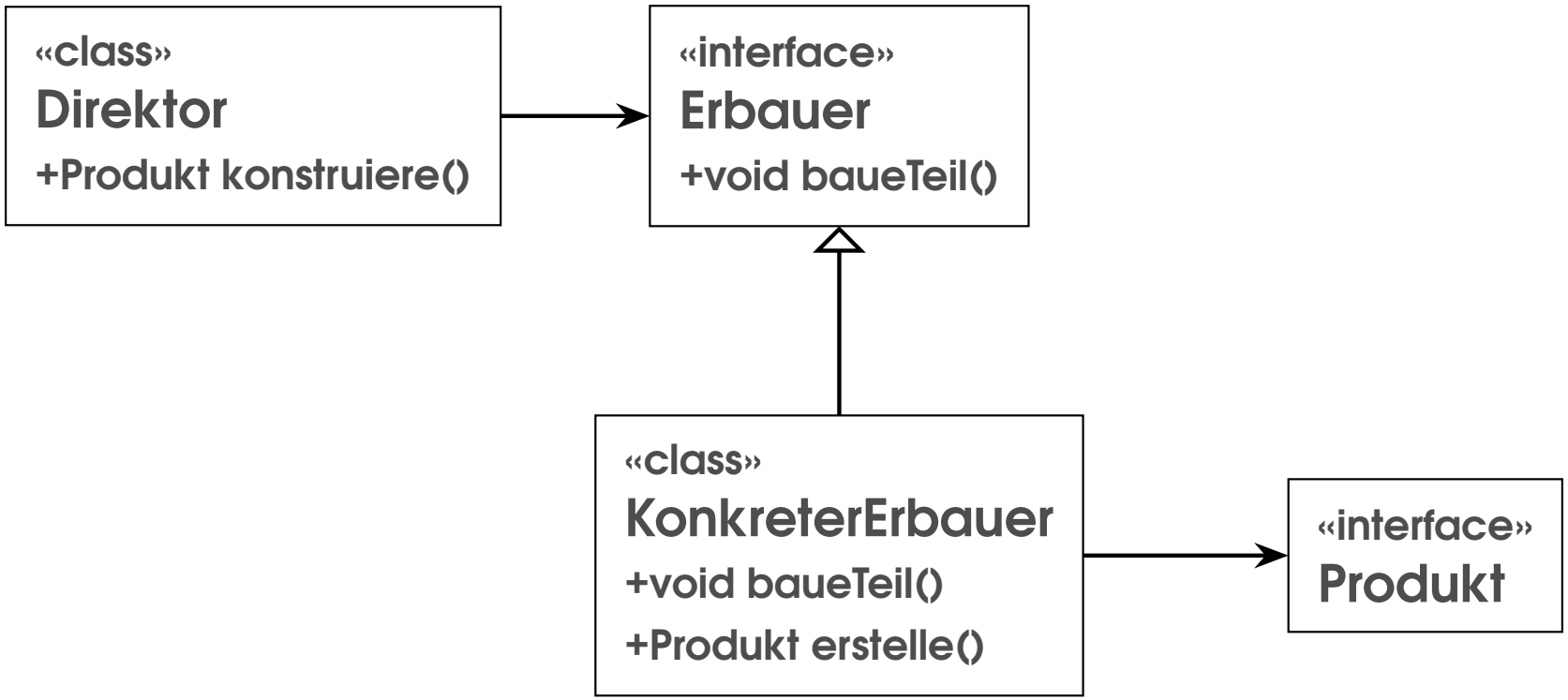
Erbauer – Beispiel



Erbauer – Anwendung,. . .

- wenn der Algorithmus zur Erzeugung komplexer Objekte unabhängig von den Teilen bzw. der Zusammensetzung dieser sein soll
- wenn die Erstellung der Objekte verschiedene Repräsentationen zur Folge hat

Erbauer – Struktur



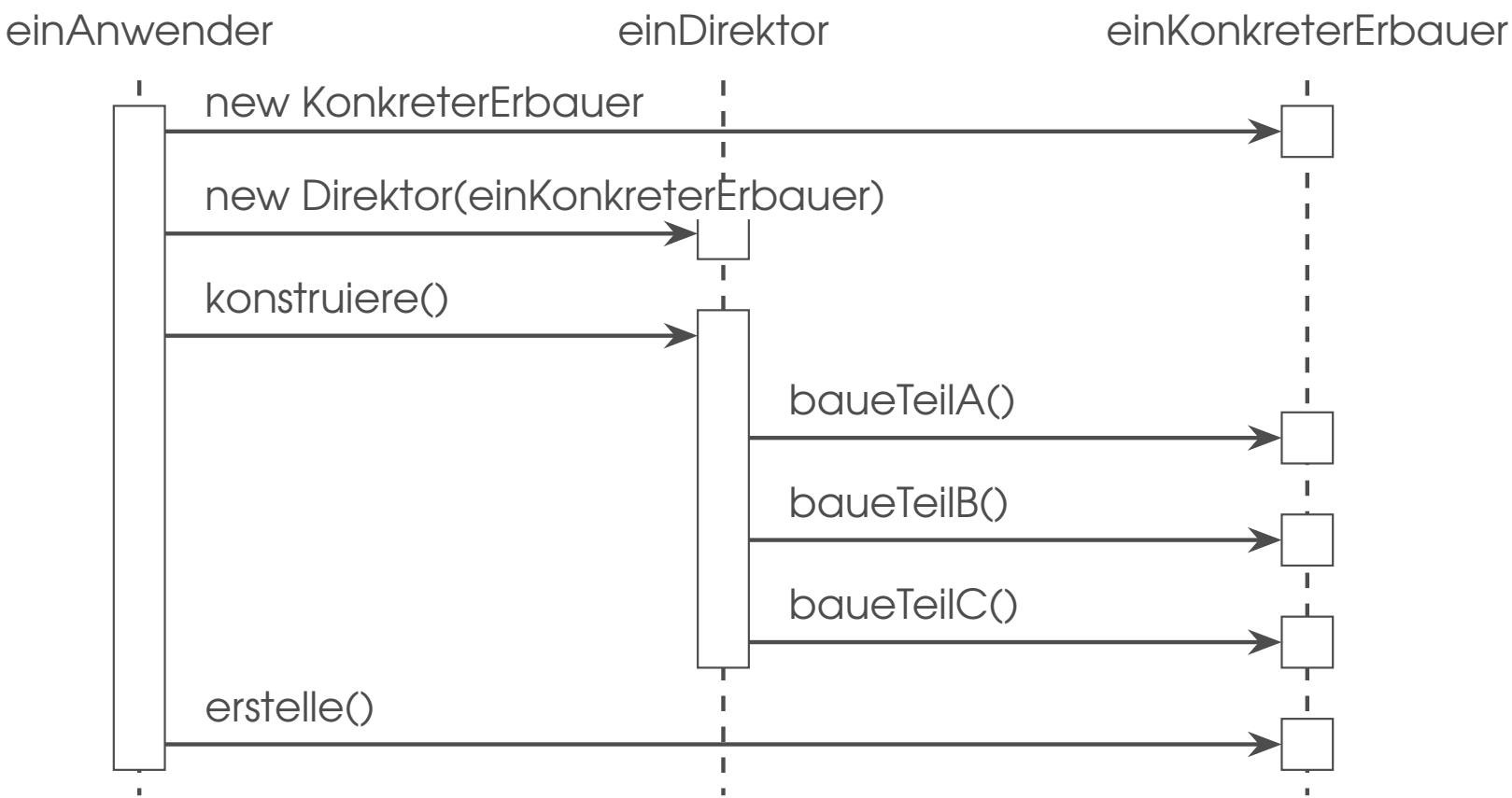
Erbauer – Akteure

- Erbauer
  - Definiert die Schnittstelle zur Erstellung der Teile eines Produkts
- Konkreter Erbauer
  - Erzeugt, konstruiert und setzt verschiedene Teile des Produkts zusammen
  - Implementiert Erbauer Schnittstelle
  - Definiert und verwaltet erstellte Teile
  - Stellt Möglichkeit zur Verfügung das Produkt zu erzeugen

Erbauer – Akteure

- Direktor
  - Konstruiert ein Produkt mit Hilfe des Erbauers
- Produkt
  - Repräsentiert komplex erzeugtes Objekt
  - Konkreter Erbauer erzeugt die interne Repräsentation und definiert den Prozess zum Zusammenfügen der Teile zu einem Ganzen
  - Schließt Klassen mit ein, die die internen Teile beschreiben, enthält Schnittstellen zum Zusammenfügen der Teile

Erbauer – Interaktion der Akteure



Erbauer – Auswirkungen

- Interne Repräsentation des Produkts kann variieren
  - Verstecken der internen Repräsentation und Zusammensetzung
- Genaue Kontrolle über den Konstruktionsprozess
  - Schritt für Schritt Erstellung der Produkte
  - Direktor gibt erst zurück, wenn fertig konstrukiert

Erbauer – Auswirkungen

- Trennung von Code zur Erstellung und Repräsentation
  - Erhöhte Modularität
  - Konkreter Erbauer enthält Code zur Erzeugung einzelner Teile des Produkts
  - Trennung der Verantwortung von Konvertierung bzw. Konstruktion (Direktor) und konkreter Erzeugung (Konkreter Erbauer)

Erbauer – Beispiel

```
public class User {
  private final String firstname;       // required
  private final String lastname;        // required
  private final Role role;              // required
  private final String emailAddress;    // optional
  private final String telephoneNumber; // optional
  private final String roomNumber;      // optional

  public User(String firstname, String lastname, Role role,
    String emailAddress, String telephoneNumber, String roomNumber) {
    super();
    this.firstname = firstname;
    this.lastname = lastname;
    this.role = role;
    this.emailAddress = emailAddress;
    this.telephoneNumber = telephoneNumber;
    this.roomNumber = roomNumber;
  }
}
```

Erbauer – Beispiel

```
public class User {
  private final String firstname;       // required
  private final String lastname;        // required
  private final Role role;              // required
  private final String emailAddress;    // optional
  private final String telephoneNumber; // optional
  private final String roomNumber;      // optional

  public User(String firstname, String lastname, Role role,
    String emailAddress, String telephoneNumber, String roomNumber) {
    super();
    this.firstname = firstname;
    this.lastname = lastname;
    this.role = role;
    this.emailAddress = emailAddress;
    this.telephoneNumber = telephoneNumber;
    this.roomNumber = roomNumber;
  }
}

User teacher = new User("Lars", "Briem", Role.teacher, null, null, null);
```

Erbauer – Beispiel

```
public class User {
  ...
  public User(String firstname, String lastname, Role role,
    String emailAddress, String telephoneNumber, String roomNumber) {
    ...
  }
  public User(String firstname, String lastname, Role role) {
    this(String firstname, String lastname, Role role, null, null, null);
  }
  public User(String firstname, String lastname, Role role, String eMailAddress) {
    this(String firstname, String lastname, Role role, eMailAddress, null, null);
  }
}

User teacher = new User("Lars", "Briem", Role.teacher);
User student = new User("Kurs", "Sprecher", Role.student, "student@dhbw.de");
```

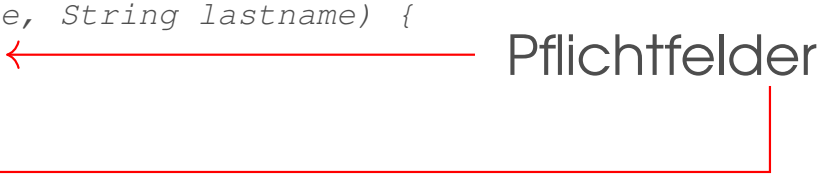
$\Rightarrow$ Beliebige Kombination der Parameter bei gleichem Typ nicht möglich

Erbauer – Beispiel

```
public final class CreateUser {
  private String firstname;
  private String lastname;
  private Role role;
  private String emailAddress;
  private String telephoneNumber;
  private String roomNumber;
  private CreateUser(String firstname, String lastname) {
    this.firstname = firstname;
    this.lastname = lastname;
  }
  public static CreateUser named(String firstname, String lastname) {
    return new CreateUser(firstname, lastname);
  }
  public CreateableUser as(Role role) {
    this.role = role;
    return new CreateableUser();
  }
  public class CreateableUser {
    ...
  }
  private User build() {
    return new User(this.firstname, this.lastname, this.role,
      this.emailAddress, this.telephoneNumber, this.roomNumber);
  }
}
```

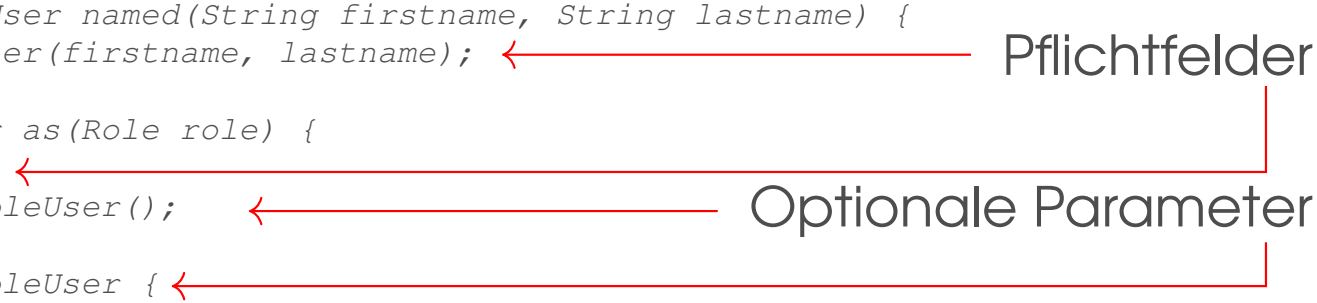
Erbauer – Beispiel

```
public final class CreateUser {
  private String firstname;
  private String lastname;
  private Role role;
  private String emailAddress;
  private String telephoneNumber;
  private String roomNumber;
  private CreateUser(String firstname, String lastname) {
    this.firstname = firstname;
    this.lastname = lastname;
  }
  public static CreateUser named(String firstname, String lastname) {
    return new CreateUser(firstname, lastname);
  }
  public CreateableUser as(Role role) {
    this.role = role;
    return new CreateableUser();
  }
  public class CreateableUser {
    ...
  }
  private User build() {
    return new User(this.firstname, this.lastname, this.role,
      this.emailAddress, this.telephoneNumber, this.roomNumber);
  }
}
```



Erbauer – Beispiel

```
public final class CreateUser {
  private String firstname;
  private String lastname;
  private Role role;
  private String emailAddress;
  private String telephoneNumber;
  private String roomNumber;
  private CreateUser(String firstname, String lastname) {
    this.firstname = firstname;
    this.lastname = lastname;
  }
  public static CreateUser named(String firstname, String lastname) {
    return new CreateUser(firstname, lastname);
  }
  public CreateableUser as(Role role) {
    this.role = role;
    return new CreateableUser();
  }
  public class CreateableUser {
    ...
  }
  private User build() {
    return new User(this.firstname, this.lastname, this.role,
      this.emailAddress, this.telephoneNumber, this.roomNumber);
  }
}
```



Erbauer – Beispiel

```
public final class CreateUser {
        ...
        public class CreateableUser {
                private CreateableUser() {}
                public CreateableUser withTelephoneNumber(String telephoneNumber) {
                        CreateUser.this.telephoneNumber = telephoneNumber;
                        return this;
                }
                public CreateableUser withEmailAddress(String emailAddress) {
                        CreateUser.this.emailAddress = emailAddress;
                        return this;
                }
                public CreateableUser withRoomNumber(String roomNumber) {
                        CreateUser.this.roomNumber = roomNumber;
                        return this;
                }
                public User build() {
                        return CreateUser.this.build();
                }
        }
        ...
}
User teacher = CreateUser.named("Lars", "Briem").as(Role.teacher).build();
User student = CreateUser.named("Kurs", "Sprecher").as(Role.student)
  .withEmailAddress("student@dhbw.de").build();
```

Erbauer – Weitere Hinweise

- Erbauer Schnittstelle so generell wie möglich halten
  - alle konkreten Erbauer abdecken
  - ohne an Details der Erbauer gebunden zu sein
- Keine abstrakte Oberklasse für Produkte
  - Interna der Produkte sind zu unterschiedlich
  - Schablonenmethoden reichen nicht aus

Erbauer – Weitere Hinweise

- Leere Methoden anstelle abstrakter Methoden in Erbauer
  - Konkrete Erbauer müssen nur notwendiges implementieren
- Fungiert als "named Parameter" zur Erzeugung großer Objekte, die nicht selbst in der Hand sind
  - In diesem Fall "Konkreter Erbauer" meist ausreichend

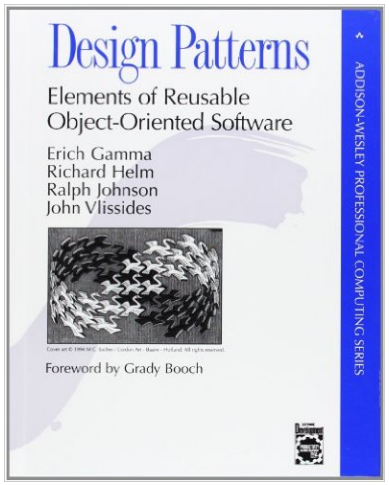
Erbauer – Verwandte Muster

- Abstrakte Fabrik
  - Hauptunterschied zur Fabrik ist die schrittweise Erzeugung
  - Eine Fabrik erzeugt Objekte sofort
- Kompositum
- Ein Kompositum wird häufig durch einen Erbauer erzeugt

Erbauer – Zusammenfassung

- Entkoppelt den Algorithmus zur Erzeugung von Objekten von den Details
- Baut Objekte schrittweise zusammen
- Ermöglicht eine getrennte Wiederverwendung beider Teile
- Vereinfacht die Erweiterbarkeit
  - Unterstützt das Open Closed Principle

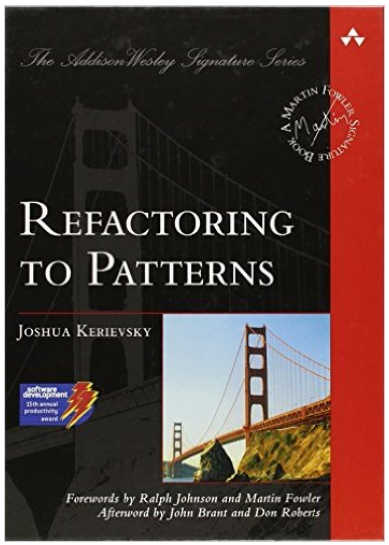
Literatur



- Design Patterns
  - Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
  - Addison-Wesley
  - ISBN: 978-0201633610

Weitere Infos

- Entwurfsmuster auf YouTube
  - John Lindquist erklärt Entwurfsmuster mit StarCraft II
  - `https://www.youtube.com/playlist?list=PL8B19C3040F6381A2`



- Refactoring to Patterns
  - Joshua Kerievsky
  - Addison-Weslay
  - ISBN: 978-0321213358