

2 Das Projekt Fahrrad

Bemerkung:

Die Idee, als erste selbstständig programmierte Klasse ein Fahrrad zu modellieren, stammt von Peter Brichzin, Gymnasium Ottobrunn. Ich habe einige seiner Ideen übernommen und überarbeitet, so dass diese an meinen Unterrichtsstil angepasst wurden.

Nun wirst du dich etwas gründlicher mit dem Quelltext einer Klasse beschäftigen. Du wirst nun die wichtigsten Bausteine einer Klasse kennen lernen: Datenfelder, Methoden und Konstruktoren.

2.1 Erstellung eines Modells

Übung 2.1:

Betrachte das Fahrrad in Abbildung 2.1 und überlege, wie du es beschreiben würdest.



Abbildung 2.1: Fahrrad

Eine Möglichkeit der Beschreibung wäre:

Typ:	Kinderfahrrad
Rahmenfarbe:	rot
Sattelfarbe:	blau
Radgröße:	24“
Gangschaltung:	5-Gang
Marke:	Epple

Mit dieser Beschreibung des Fahrrads hast du zwei Dinge gleichzeitig gemacht:

1. Du hast eine Klasse *RAD* erstellt.
2. Du hast mit Hilfe dieser Klasse ein Objekt, z. B. *felixRad*, erzeugt.

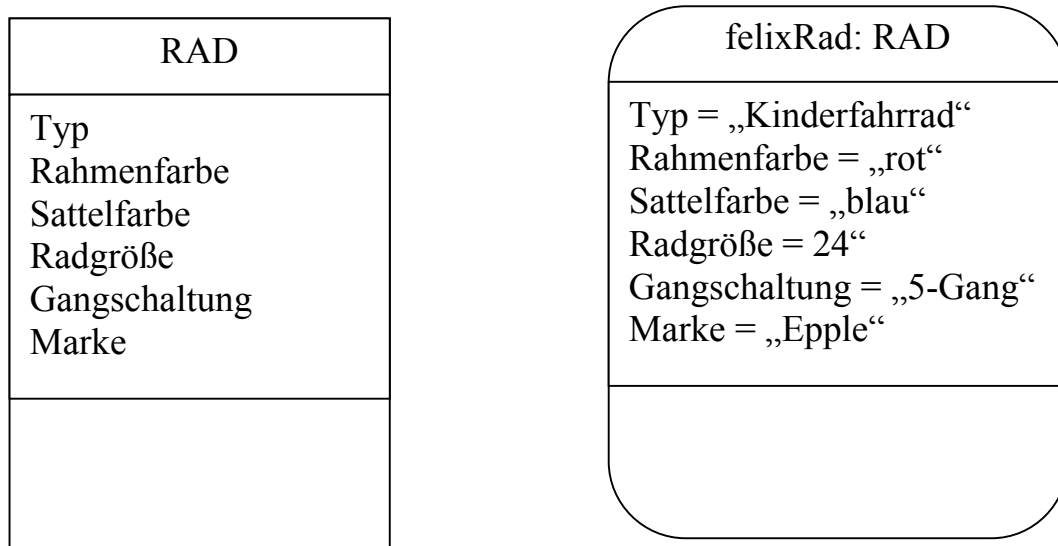


Abbildung 2.2: Die Klasse *RAD* und das Objekt *felixRad*

Eine Klasse dient als Fabrik oder Bauplan für beliebig viele Objekte. Abbildung 2.2 zeigt, dass mit Hilfe der Klasse (Bauplan) *RAD* das Objekt *felixRad* erzeugt wurde. Nach demselben Muster kannst du auch Objekte *ninaRad*, *veraRad*, usw. erzeugen. All diese Objekte besitzen dieselben Attribute (Eigenschaften), unterscheiden sich allerdings in den Werten, die diese Attribute annehmen.

Im Klassenbauplan sind also alle Attribute angelegt. Werte erhalten erst die Attribute der Objekte.

2.2 Implementierung der Klasse Rad

2.2.1 Datenfelder beschreiben die Eigenschaften (Attribute)

Nun wirst du die Klasse *Rad* in **BlueJ** implementieren. Damit diese nicht zu kompliziert wird, verwendest du zunächst nur zwei Attribute *farbe* und *aktuellerGang*.



Abbildung 2.3: Modell eines Rads

Abbildung 2.3 zeigt das einfache Modell der Klasse *Rad*. Anhand der Abbildung 2.4 und der Übung 2.2 kannst du nun dieses Modell in **BlueJ** implementieren.

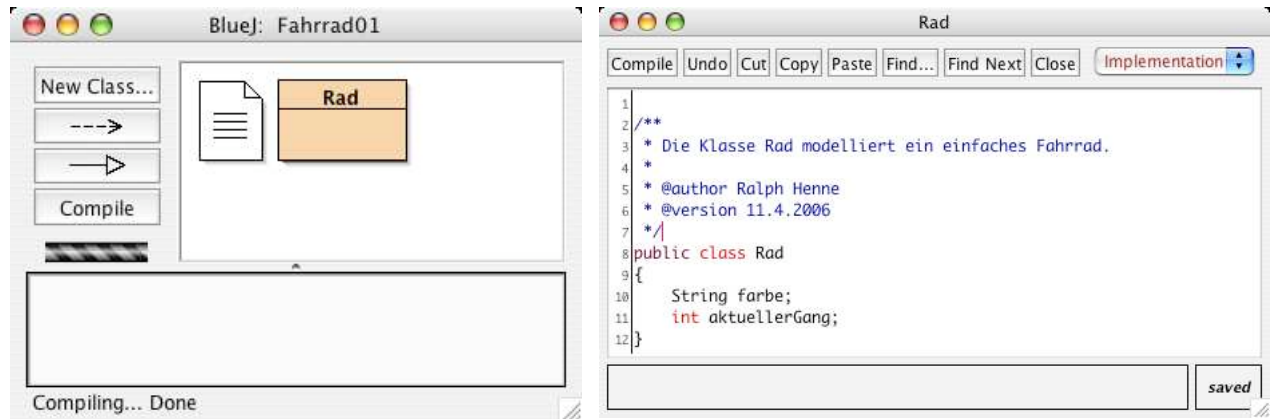


Abbildung 2.4: Klassendiagramm und Quelltext der Klasse *Rad*

Übung 2.2:

- Erstelle in **BlueJ** ein Projekt *Fahrrad01*.
- Mit *New Class* erstelle nun innerhalb dieses Projekts eine Klasse *Rad*.
- Ersetze im Editorfenster den gesamten vorgeschlagenen Quelltext durch den in Abbildung 2.4 dargestellten.
- Kompiliere die Klasse *Rad*.
- Erzeuge mit *new Rad()* die drei Fahrräder *felixRad*, *veraRad* und *ninaRad*.
- Untersuche mit Hilfe des Inspektors das *felixRad*.



Abbildung 2.5: Objektleiste und Inspektor

Der Inspektor des Objekts *felixRad* zeigt die aktuellen Werte in den Datenfeldern *farbe* und *aktuellerGang*. Da du bis jetzt noch keine speziellen Werte diesen Datenfeldern zugewiesen hast, werden die so genannten Standardwerte gezeigt, in diesem Fall also die Werte *null* und *0*.

Du wirst dich jetzt natürlich fragen, warum der Inspektor zwei verschiedene Standardwerte *null* und *0* zeigt. Java verwendet zur Darstellung von Daten verschiedenen Datentypen. In der Klasse *Fahrrad* hast du die beiden Datentypen *String* und *int* verwendet. Der Datentyp *String* wird für eine Zeichenkette verwendet, der Datentyp *int* für eine ganze Zahl.

Grundsätzlich werden Datentypen in zwei Kategorien geteilt:

Primitive Typen: Der Wert eines primitiven Typs wird direkt in das Datenfeld geschrieben. Abbildung 2.5 zeigt den primitiven Typ *int*. Weitere primitive Typen sind *boolean*, *char*, *short*, *long*, *float*, *double*.

Objekttypen: In das Feld eines Objekttyps wird nicht sein Wert, sondern ein Link (Referenz) auf ein Objekt geschrieben. Abbildung 2.5 zeigt den Objekttyp *String*, der hier mit dem „ungültigen“ Link *null* initialisiert wurde. (Beachte die Großschreibung von *String*!)

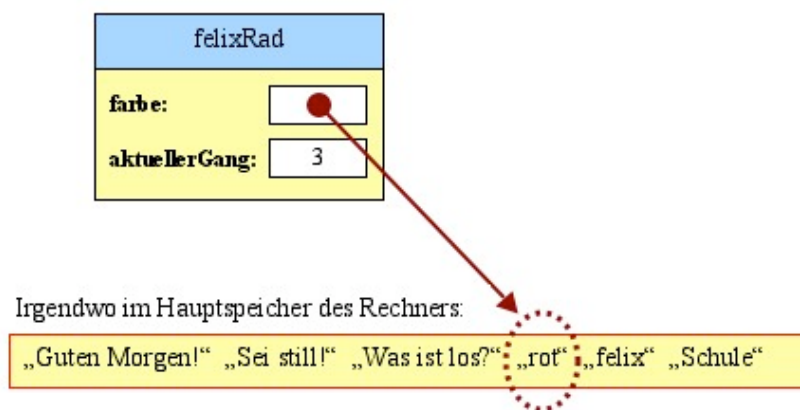


Abbildung 2.6: Referenz auf das Objekt *rot* vom Typ *String*

2.2.2 Konstruktoren initialisieren Objekte

Wie Abbildung 2.6 darstellt, sollten in den Datenfeldern *farbe* und *aktuellerGang* die Werte „rot“ und 3 stehen. Der Inspektor des Objekts *felixRad* zeigt jedoch hier nach der Erzeugung von *felixRad* die Standardwerte *null* und 0.

Java bietet einen Mechanismus, der dafür sorgt, dass die Datenfelder von Objekten bereits während deren Erzeugung mit sinnvollen Werten belegt werden. Hierzu verwendet man den so genannten *Konstruktor*. Der Konstruktor wird einmalig bei der Erzeugung des Objekts aufgerufen und setzt dieses Objekt in einen vernünftigen Anfangszustand.

```
public class Rad
{
    String farbe;
    int aktuellerGang;

    public Rad()
    {
        farbe = "rot";
    }
}
```

```
    aktuellerGang = 3;
}
}
```

Abbildung 2.7: Ein möglicher Konstruktor in der Klasse *Rad*

Der Konstruktor wird nur einmal aufgerufen, nämlich wenn mit `new Rad()` ein Objekt erzeugt wird. Die entsprechenden Datenfelder werden nun mit den Werten *rot* bzw. *3* belegt

Übung 2.3:

Implementiere den in [Abbildung 2.7](#) dargestellten Quelltext in deiner Klasse *Rad*. Erzeuge anschließend ein Objekt *felixRad* und untersuche die Attributwerte mit Hilfe des Inspektors. Vergleiche dein Ergebnis mit der [Abbildung 2.8](#).

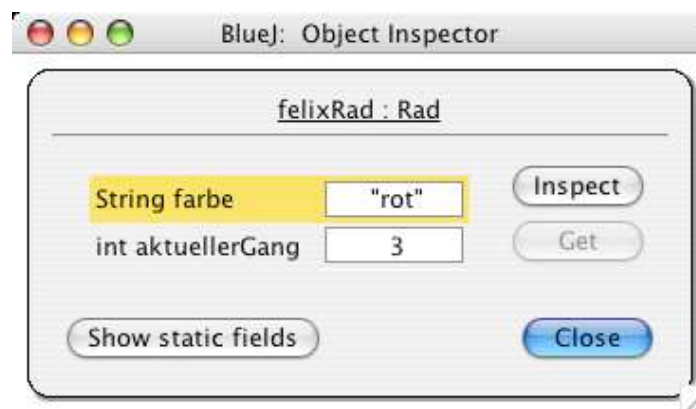


Abbildung 2.8: Inspektor des Objekts *felixRad*

Übung 2.4:

Erzeuge nun zwei weitere Objekte *ninaRad* und *veraRad*. Untersuche auch diese Objekte mit Hilfe ihrer Inspektoren. Formuliere, welches Problem aufgetreten ist.

Du wirst festgestellt haben, dass nun alle drei Räder dieselbe Farbe und denselben aktuellen Gang besitzen. Es ist aber sehr unwahrscheinlich, dass Felix, Nina und Vera drei gleiche Fahrräder besitzen. Du musst also nun den Konstruktor so verändern, so dass die Fahrräder unterschiedliche Farben haben.

```
public class Rad
{
    String farbe;
    int aktuellerGang;

    /** Konstruktoren */
    public Rad()

```

```

{
    farbe = "rot";
    aktuellerGang = 3;
}

public Rad(String farbeH, int aktuellerGangH)
{
    farbe = farbeH;
    aktuellerGang = aktuellerGangH;
}
}

```

Abbildung 2.9: Die Klasse *Rad* erhält einen weiteren Konstruktor

Übung 2.5:

Implementiere in der Klasse *Rad* den weiteren Konstruktor wie in Abbildung 2.9 dargestellt. Erzeuge anschließend das Objekt *veraRad* mit den Werten „blau“ und 2. Vergleiche dazu die Abbildung 2.10.

Erzeuge auch ein Objekt mit dem ursprünglichen Konstruktor. Formuliere schriftlich die Unterschiede zwischen diesen beiden Konstruktoren.

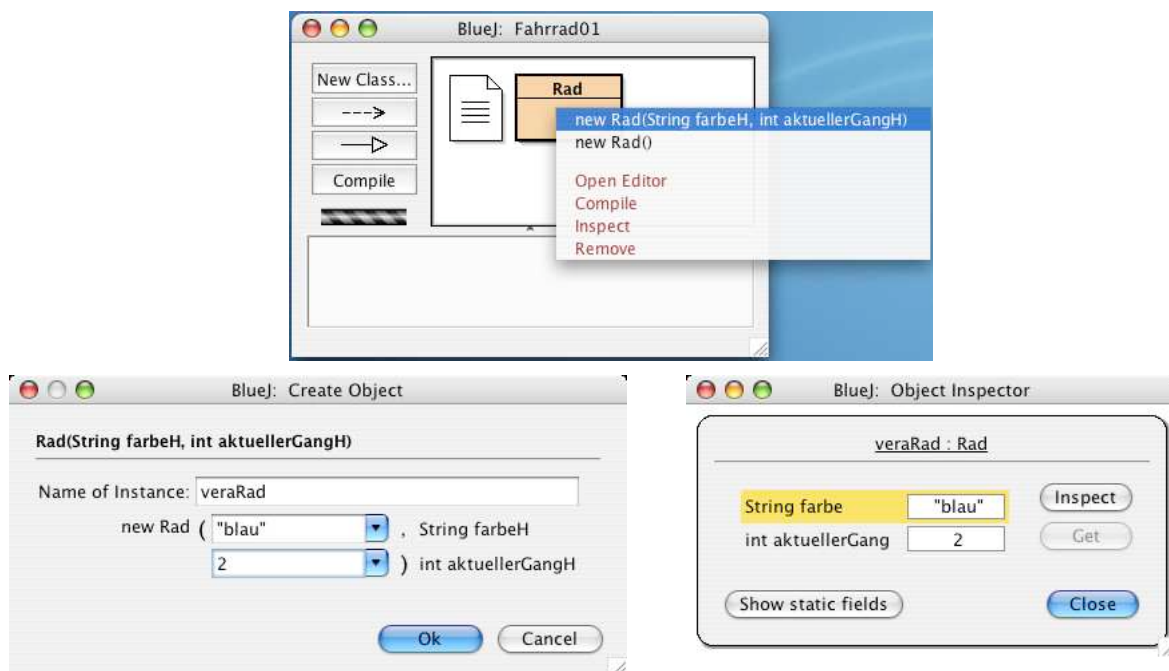


Abbildung 2.10: Erzeugung des Objekts *veraRad*

Bemerkung:

Ich bezeichne im Konstruktor die formalen Parameter mit dem Buchstaben „H“ für **H**ilfsparameter. Somit entfällt die Suche nach einem weiteren geeigneten Parameternamen. In der Literatur werden auch häufig Bezeichnungen wie *neuFarbe* verwendet.

Eine weitere Möglichkeit ist die Verwendung des Schlüsselworts **this**. Jedoch hat eine Einführung dieses Schlüsselworts zu diesem Zeitpunkt die Schüler nur verwirrt.

2.2.3 Methoden beschreiben die Fähigkeiten

Nina, Vera und Felix unternehmen nun eine kleine Radtour. Da die Fahrt durch hügeliges Gelände geht, müssen sie öfters den Gang wechseln. Die meisten Gangschaltungen zeigen in einem kleinen Fenster den aktuellen Gang an.

In diesem Kapitel wirst du die oben beschriebene Situation simulieren. Dazu musst du allerdings deine Klasse *Rad* erweitern. Bis jetzt enthält das Modell nur die Attribute (Datenfelder), nun fügst du geeignete Fähigkeiten (Methoden) hinzu.

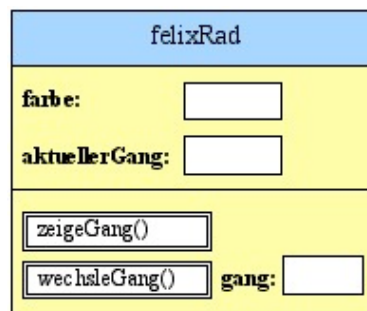


Abbildung 2.11: Das Modell der Klasse *Rad* enthält nun auch Methoden

Übung 2.6:

Die Abbildung 2.11 deutet an, dass die beiden Methoden *zeigeGang()* und *wechseleGang()* verschiedene Aufgaben zu erledigen haben. Formuliere mögliche Unterschiede zwischen diesen beiden Methoden.

Die Methoden lassen sich in zwei Arten klassifizieren:

Sondierende Methoden liefern Objekt-Informationen

Sie liefern einen Attributwert zurück, z.B. eine Zahl, einen String, o. Ä.
Sie sollen das Objekt **nicht** verändern.

Verändernde Methoden verändern das Objekt

Sie verändern Attributwerte eines Objekts.
Sie werden i. A. nicht verwendet, um Informationen über das Objekt zu liefern.

Übung 2.7:

Welchen der beiden Arten lassen sich die Methoden *zeigeGang()* und *wechsleGang()* zuordnen. Begründe deine Entscheidung.

Dieser Unterschied zeigt sich auch in der Implementierung dieser beiden Methoden:

```
public int zeigeGang()
{
    return aktuellerGang;
}

public void wechsleGang(int gang)
{
    aktuellerGang = gang;
}
```

Abbildung 2.12: Die Methoden *zeigeGang()* und *wechsleGang()* in der Klasse *Rad*

Die Methode *zeigeGang()* zeigt den Wert des Datenfeldes *aktuellerGang*. Sie liefert also einen Rückgabewert mit dem Datentyp *int*. Dieser Datentyp muss im Methodenkopf `public int zeigeGang()` beschrieben werden. Den Wert des Datenfeldes *aktuellerGang* liefert die *return*-Anweisung.

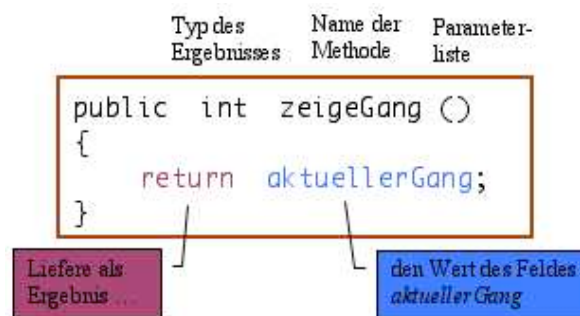


Abbildung 2.13: Sondierende Methode *zeigeGang()*

Die Methode *wechsleGang()* besitzt keinen Rückgabewert. Auch dies wird im Methodenkopf `public void wechsleGang(int gang)` mit dem Schlüsselwort *void* (engl.: Leerstelle, Fehlstelle) dargestellt. Allerdings ändert diese Methode den Wert des Datenfeldes *aktuellerGang*. Der neue Wert wird der Methode als Parameter `int gang` übergeben. Nun kann der Parameterwert in das Datenfeld *aktuellerGang* eingelesen werden.

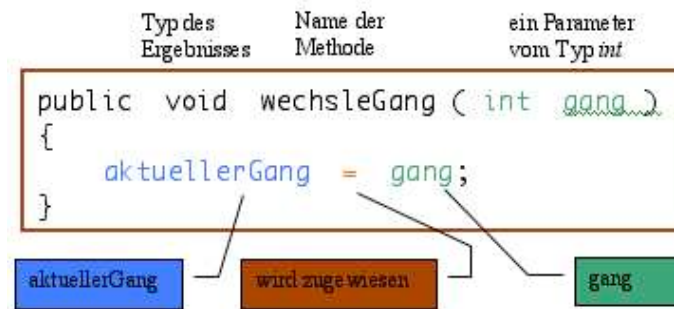


Abbildung 2.14: Verändernde Methode *wechsleGang()*

Übung 2.8:

Implementiere in der Klasse *Rad* die beiden Methoden *zeigeGang()* und *wechsleGang()* wie in Abbildung 2.12 dargestellt. Erzeuge anschließend ein Objekt *felixRad*. Teste nun die beiden Methoden. Vergleiche dazu die Abbildung 2.15.

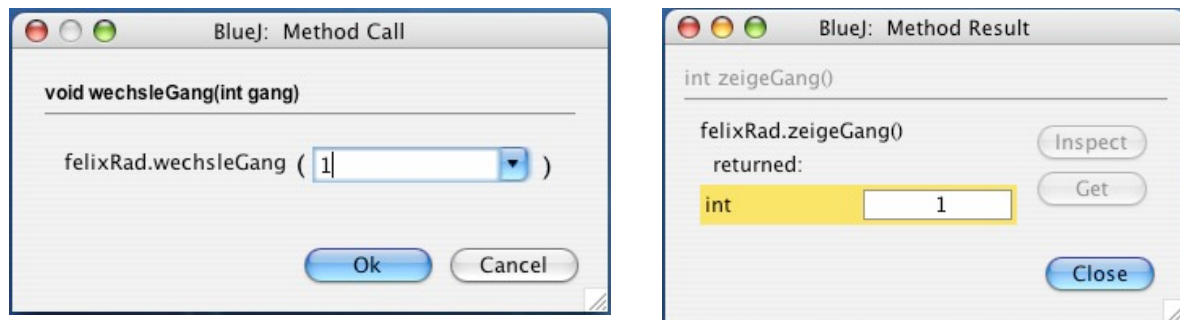


Abbildung 2.15: Felix hat in den 1.Gang geschaltet

2.2.4 Dokumentation

Du solltest dir angewöhnen, während der Programmierung deinen Quelltext zu dokumentieren. Java besitzt verschiedene Möglichkeiten der Dokumentation. In Abbildung 2.4 wurde eine Art bereits beschrieben:

```
/** ..... Erläuterungen ..... */
```

Als Minstdokumentation solltest du auf diese Weise jede Methode kurz erklären. Abbildung 2.16 zeigt nun den dokumentierten Quelltext der Klasse *Rad*. Verwende auch die von **BlueJ** automatisch durchgeführten Einrückungen, die die Lesbarkeit des Quelltextes vereinfachen.

```
/**
 * Die Klasse Rad modelliert ein einfaches Fahrrad.
 *
 * @author Ralph Henne
 * @version 11.4.2006
```

```
*/
public class rad
{
    String farbe;
    int aktuellerGang;

    /** Konstruktoren */
    public Rad()
    {
        farbe = "rot";
        aktuellerGang = 3;
    }

    public Rad(String farbeH, int aktuellerGangH)
    {
        farbe = farbeH;
        aktuellerGang = aktuellerGangH;
    }

    /** Liefert den aktuellen Gang. */
    public int zeigeGang()
    {
        return aktuellerGang;
    }

    /** Wechselt in einen neuen Gang. */
    public void wechsleGang(int gang)
    {
        aktuellerGang = gang;
    }
}
```

Abbildung 2.16: Dokumentierter Quelltext der Klasse *Rad*

Übung 2.9:

Dokumentiere die Klasse *Rad* nach dem Muster aus [Abbildung 2.16](#).

2.2.5 Weitere Übungen zur Klasse *Fahrrad*

Du wirst nun weitere Übungen mit dem Fahrrad durchführen. Bei diesen Experimenten wird immer etwas schief laufen, das ist ganz normal so. Bei großen Veränderungen solltest du dir aber immer ein paar Sicherheiten einbauen. Ich habe mir angewöhnt, von funktionierenden Projekten Sicherheitskopien anzulegen, auf die ich im Notfall wieder zurückgreifen kann. Bei mir heißen dann solche Kopien *Fahrrad01*, *Fahrrad02*, *Fahrrad03*, usw.

Übung 2.10

Speichere dein Projekt *Fahrrad01* unter dem Namen *Fahrrad02*.

Felix möchte nun sein Fahrrad mit einer anderen Farbe streichen. Implementiere Methoden, um die Farbe des Fahrrads zu ändern und diese auch anzuzeigen. (vgl. Projekt *Fahrrad02*)

Übung 2.11:

Felix ärgert die beiden Mädchen und schraubt die Ventile aus den Reifen. Aber er wird dabei erwischt und muss zur Strafe die Fahrradschläuche wieder aufpumpen. Entwerfe geeignete Datenfelder und Methoden. (vgl. Projekt *Fahrrad03*)

Übung 2.12:

Auf einer Radtour tauschen die beiden Mädchen ihre Fahrräder. Implementiere geeignete Datenfelder und Methoden. (Es soll nicht der Austauschvorgang simuliert werden, sondern nur, dass Nina mit Veras Rad fährt u. u.) (vgl. Projekt *Fahrrad04*)

2.3 Verbesserungen der Klasse Rad

In diesem Kapitel wirst du die Klasse *Rad* verbessern, das bedeutet, dass du dein bisheriges Modell besser an die Realität anpassen wirst.

Übung 2.13:

Speichere dein Projekt *Fahrrad01* unter dem Namen *Fahrrad05*. Erzeuge ein Objekt *felixRad* und rufe dessen Methode *wechsleGang()* auf. Gib als Eingabewert die Zahl -3 ein. Überprüfe nun im Inspektor, in welchen Gang Felix gerade geschaltet hat.

Selbstverständlich machen negative Gänge keinen Sinn. Deshalb dürfen auch keine Eingabewerte angenommen werden, die kleiner als Eins sind, da dies der kleinste Gang ist.

Java bietet für solche Probleme Kontrollstrukturen. Eine der wichtigsten Kontrollstrukturen ist die *if*-Anweisung, die zur Lösung des obigen Problems verwendet werden kann:

Übung 2.14:

In den Fachbüchern findet man für die *if*-Anweisungen die unten aufgeführte Beschreibung oder eine ähnliche. Versuche zu verstehen, was hier gemeint sein

könnte. Wie würde eine Übersetzung ins Deutsche lauten? Finde Beispiele, wie man diese *if*-Anweisung verwenden könnte.

***if*-Anweisung**

```
if (Bedingung) {  
    Anweisungen1;  
}  
else {  
    Anweisungen2;  
}
```

Wertet die Bedingung aus. Ist sie wahr, so werden die Anweisungen1 ausgeführt; ist sie falsch, werden die Anweisungen2 ausgeführt.
Der **else**-Teil ist optional. Ist er nicht vorhanden, so führt eine falsche Bedingung dazu, dass die Ausführung mit der Anweisung nach der **if**-Anweisung weitermacht.

Das Problem der negativen Gänge könnte nun mit Hilfe der obigen Vorschrift folgendermaßen formuliert werden:

```
wenn (gang ist kleiner Eins) {  
    setze aktuellerGang auf 1;  
}  
sonst {  
    setze aktuellerGang auf gang;  
}
```

Dies ist zwar kein schönes Deutsch, aber der Computer kann mit solchen Anweisungen etwas anfangen. In Java lautet die Methode *wechsleGang()* also nun:

```
/** Wechselt in einen neuen Gang. */  
public void wechsleGang(int gang)  
{  
    if (gang < 1) {  
        aktuellerGang = 1;  
    }  
    else {  
        aktuellerGang = gang;  
    }  
}
```

Abbildung 2.17: Erweiterter Quelltext der Methode *wechsleGang()*

Übung 2.15:

Implementiere in dem Projekt *Fahrrad05* die in Abbildung 2.17 dargestellte Methode *wechsleGang()*. Überprüfe nun, ob du in einen Gang kleiner Eins schalten kannst!

Übung 2.16: (für Experten)

Das Fahrrad *felixRad* hat eine 5-Gangschaltung, das *ninaRad* eine 7-Gangschaltung und *veraRad* eine 3-Gangschaltung.

- a) Entwerfe einen geeigneten Konstruktor.
- b) Implementiere die Methode *wechsleGang()*, so dass die drei Kinder nur in die jeweils erlaubten Gänge schalten können.
- c) Teste deine Klasse!