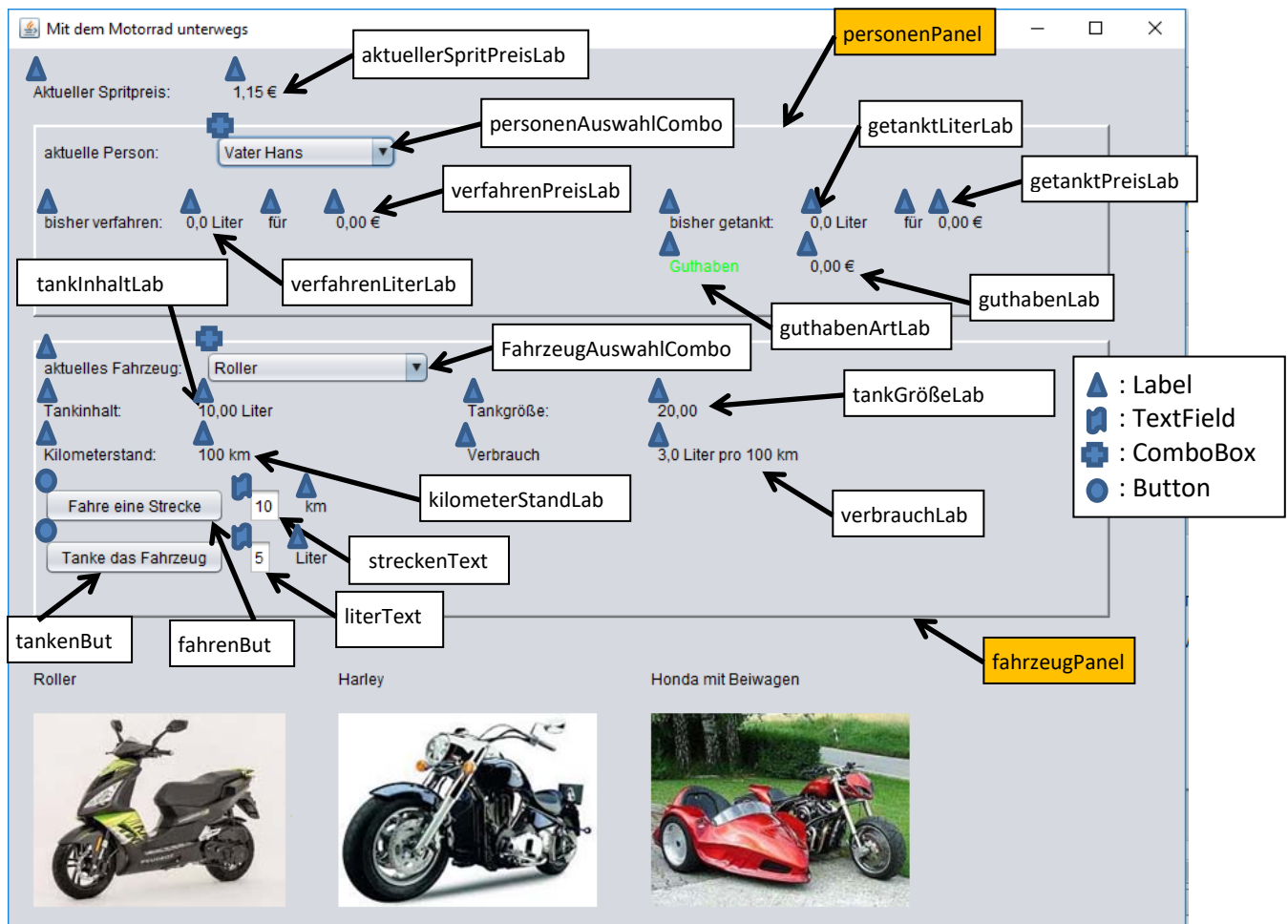


Projekt 2: Unser erstes echtes Java-Programm: Mit dem Motorrad unterwegs

Stellt euch vor, wir kennen eine Familie, in der Vater, Mutter, Sohn Hans und Tochter Gerda begeisterte Motorradfahrer sind. Sie haben insgesamt drei Motorräder: einen Roller, eine Harley und eine Honda, für die es sogar einen Beiwagen gibt. Oft streiten sie sich, wenn mal wieder jemand eine Spritztour gemacht, aber nicht getankt hat. Deshalb soll ein Computerprogramm genutzt werden, indem alle Familienmitglieder eingeben, wie viel sie gefahren sind und wie viel sie getankt haben.

Die Oberfläche des Computerprogramms (genannt „Desktop“) soll so aussehen:

Vollständige Version



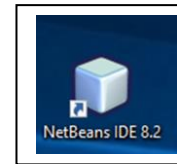
Alle weißen Rechtecke wird man später nicht sehen. Sie geben nur den Namen der Objekte an, die man auf dem Bildschirm sehen soll. Zunächst findest du zwei **Panels** (Klasse JPanel). In diese Panels werden die Objekte, die zueinander passen, einsortiert: Das **personenPanel** oben enthält alle Angaben, die ein Familienmitglied mit den Fahrzeugen gemacht hat. Das **fahrzeugPanel** darunter enthält alle Angaben, die mit einem der drei Fahrzeuge unternommen wurden. Die Panel erscheinen auf dem Bildschirm als Rechteck und helfen dem Benutzer bei der Orientierung.

Innerhalb der Panels sind viele „Labels“ untergebracht: In einem **Label (Klasse JLabel)** kann ein beliebiger Text geschrieben werden. Was geschrieben wird, bestimmt das Computerprogramm. Labels sind z. B. „aktuellerSpritPreisLab“ oder „guthabenLab“. Labels erkennst du auch daran, dass der Name jeweils auf „Lab“ endet. Weiterhin findest du die beiden „Comboboxen“ „personenAuswahlCombo“ und „FahrzeugauswahlCombo“. Hier kann der Benutzer zwischen verschiedenen Personen aus der Familie wählen, die ein Motorrad benutzen wollen. In der anderen **Combobox** (Klasse JComboBox) kann das Fahrzeug ausgewählt werden. Schließlich gibt es noch die beiden **Textfelder** „streckenText“ und „literText“. In einer Textbox (Klasse JTextField) kann der Benutzer die eingetragenen Zahlen selbst nach Belieben verändern. Auf ein Objekt der Klasse JButton kann der Benutzer später klicken und eine Reaktion auslösen.

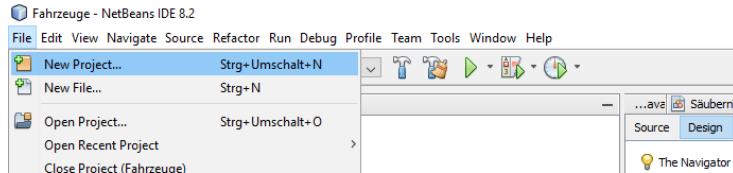
Aufgabe 1: Erstelle dein erstes Java-Programm und erzeuge in diesem die oben dargestellte grafische Benutzeroberfläche, d. h. den Desktop (Aussehen des Bildschirms), den der Benutzer beim Start des Programmes sehen wird. Auf Englisch nennt man das **Grafical User Interface**, kurz „Gui“ und dieses **Gui** müssen wir erzeugen.

Jetzt starten wir zum ersten Mal unsere Java-Umgebung. Diese Java-Umgebung heißt „**Netbeans**“ und wird über das entsprechende Icon auf dem Computer angeklickt. Netbeans erhält viele Komponenten, die dem Programmierer beim Erstellen eines Java-Programms unterstützen.

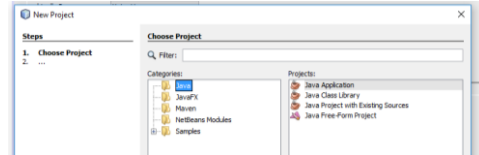
Wähle nach dem Start von Netbeans im Menü „File“ „New Project“.



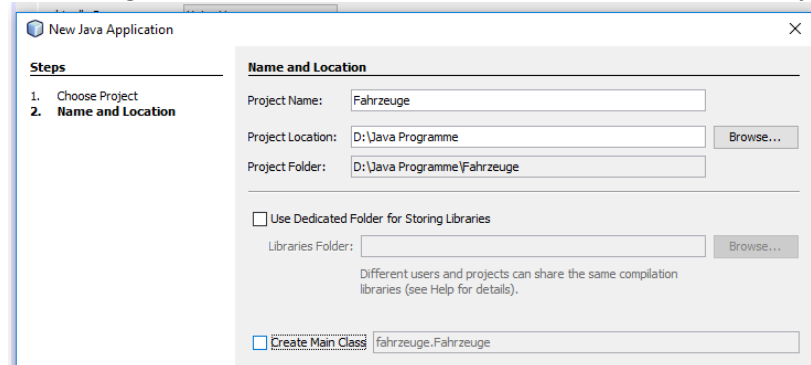
Ein Projekt erzeugen



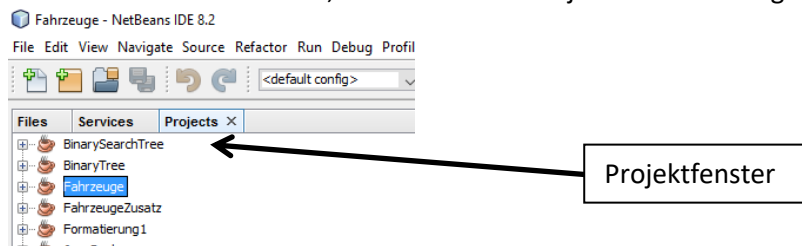
Im erscheinenden Auswahlfenster lässt du die Einstellungen „Java“ und „Java-Application“ ausgewählt und klickst am unteren Rand des Fensters auf „Next“.



Im nächsten Fenster gibst du bei „Project Name“ den Namen „**Fahrzeuge**“ ein, so wollen wir unser erstes Projekt nennen. Bei „Project Location“ muss dein Verzeichnis eingetragen sein, indem du Speicherrecht hast, z. B. „\Mit\2 Hans und Bert“. Der Haken bei „Create Main Class“ muss **gelöscht** werden, wenn wir eine **Gui** in unser Programm einbauen wollen. Klicke auf „Finish“ und eine Projektmappe wird erstellt.

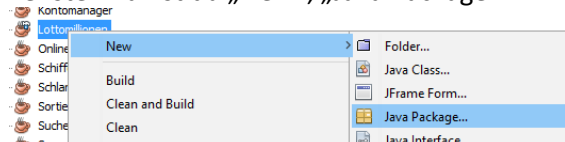


Wenn alles geklappt hat, siehst du auf der linken Seite im „Projekt-Fenster“ dein neues Projekt „Fahrzeuge“, klicke mit der Maus darauf, dann wird dein Projekt blau hinterlegt dargestellt.

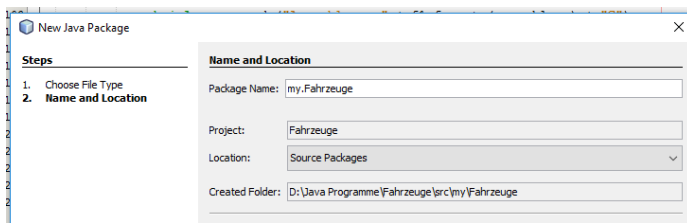


Ein Package erzeugen

Im neuen Projekt „Fahrzeuge“ erzeugen wir als nächstes ein **Package**. Innerhalb eines Package speichert Netbeans alle Dateien ab, die für dein Projekt benötigt werden. Um ein Package zu erzeugen, klickst du mit der rechten Maustaste auf den Namen deines Projektes, bei uns also auf „Fahrzeuge“. Im erscheinenden Fenster wählst du „New“, „Java Package“.

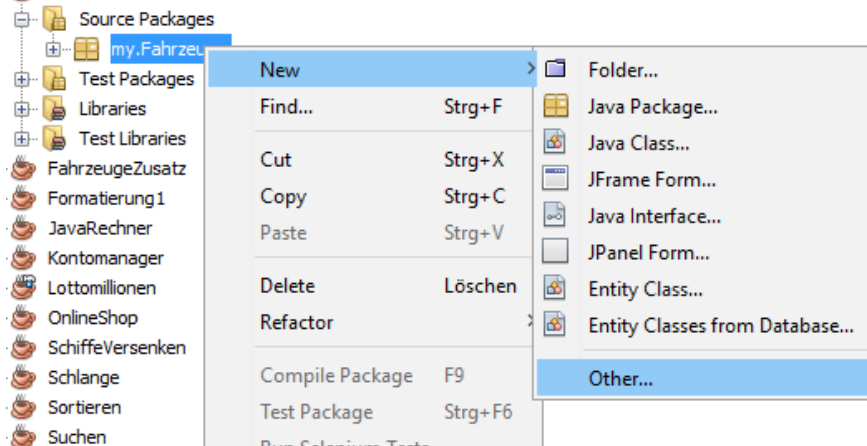


Im zugehörigen Fenster wählen wir als Package Name „**my.Fahrzeuge**“ zum Zeichen dafür, dass dies unser eigenes Package ist. Der Projektname „Fahrzeuge“ wird bereits angezeigt, unter „Created Folder“ kannst du bereits sehen, an welcher Stelle das Package eingerichtet wird.

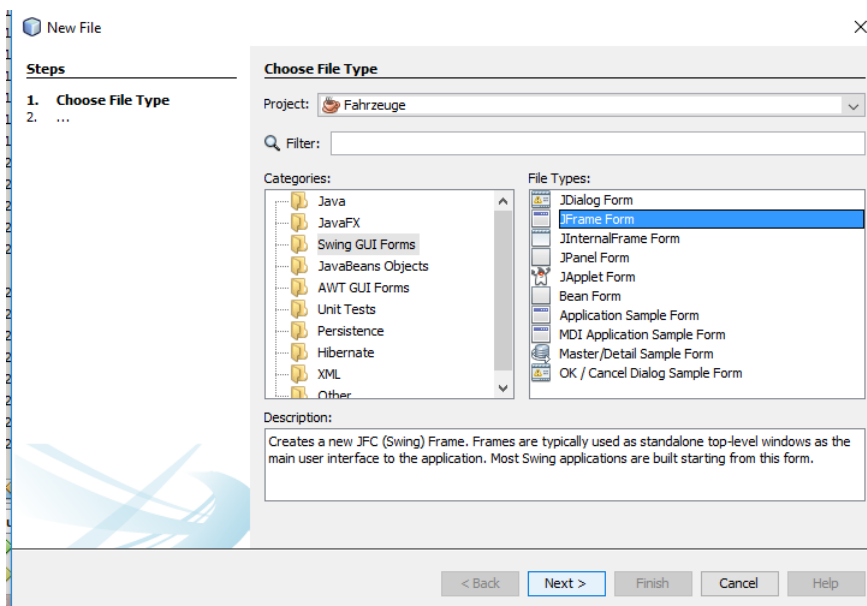


Als nächstes tippen wir zunächst auf das Pluszeichen vor unserem Projekt „Fahrzeuge“, danach auf das Pluszeichen vor „Source Packages“. Weiter geht es mit einem Rechts-Klick auf unser Package „my.Fahrzeuge“, im aufklappenden Fenster wählen wir „New“, „Other“.

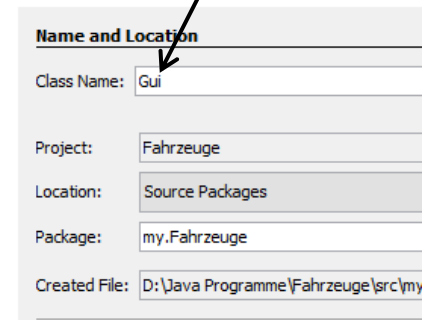
Ein Gui-Klasse erzeugen



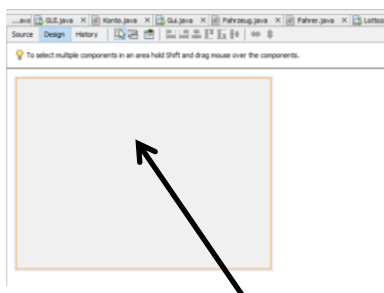
Weiter geht es mit der Auswahl „Swing GUI Forms“ und rechts „JFrame-Form“. Das Frame ist der „Bildschirmrahmen“ in den wir mit den Elementen des Guis unseren Bildschirm für den Benutzer gestalten können. Wenn wir auf „Next“ klicken, wird gleich unsere erste Java-Klasse angelegt.



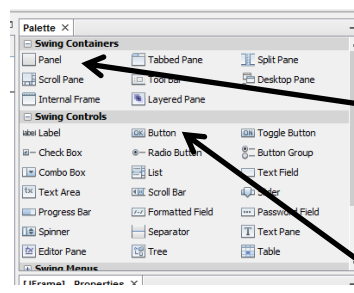
Der neue Name der Klasse **Gui** wird eingetragen:



Im letzten Fenster geben wir der neuen Klasse noch den **Class Name** „Gui“ und schließen das Ganze mit „Finish“ ab. Schließlich siehst du die erste Oberfläche, die wir mit dem **Gui** nun gestalten können:



Ein leeres Frame (Rahmen) als rot umrandetes Rechteck im Designer-Fenster

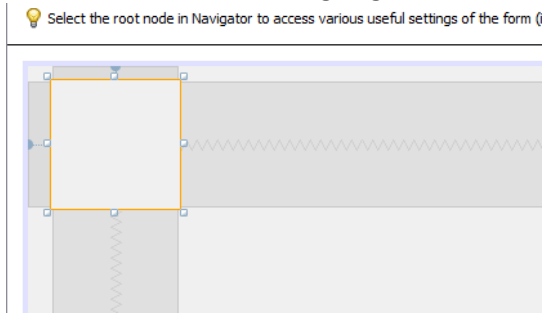


Das Element Panel

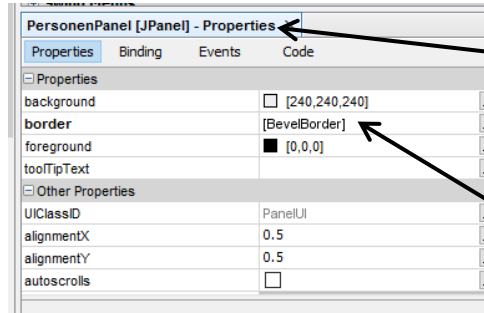
Die verschiedenen Gui-Elemente, die auf dem Frame platziert werden können

Als erstes platzieren wir auf dem leeren Frame zwei Panels, in der wir später die Informationen zu einer Person, die ein Motorrad fährt, bzw. dem Motorrad selbst unterbringen können: Klicke dazu ganz Rechts unter „Swing-Containers“ auf das Wort **Panel**, bewege die Maus auf dein Frame und klicke nochmals an der Stelle, an der das Panel abgelegt werden soll: du erkennst einen roten Rahmen.

Gui-Objekte erzeugen



Das erste Panel erscheint im Designer-Fenster



Das **Propertie-Fenster** enthält die Eigenschaften der Gui-Objekte.

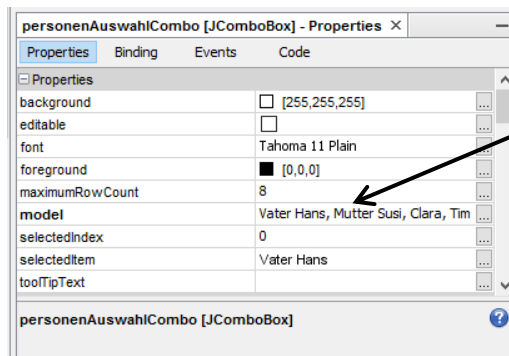
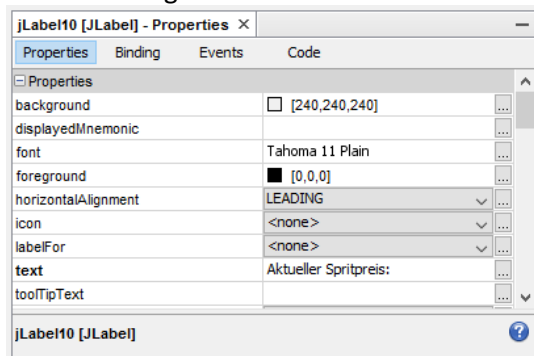
„**Bevelborder**“ wird als Borderstyle ausgewählt

Im Eigenschaftsfenster wird der Border „Bevelborder“ gewählt

Die Größe des Rahmens kannst du verändern, indem du mit der Maus an den verschiedenen Pfeilen ziehst, die das rote Rechteck begrenzen. In dem Moment, indem du an einer der Ecken ziehst, werden die Koordinaten angezeigt. Unser Panel soll in x-Richtung 815 und in y-Richtung 145 Pixel haben. Das Frame ist dazu noch zu klein, vorher kannst du das Frame vergrößern, so dass die beiden Panels untereinander passen. Klicke dazu das Frame an. Damit man die Panels auf dem Bildschirm später erkennen kann, wählst du rechts im Eigenschaftsfenster bei der Eigenschaft (Property) „**border**“ die Option „**BevelBorder**“ aus.

In die Panels platzierst du nun die einzelnen Gui-Elemente, wie du sie auf Seite 1 sehen kannst. Folgende Elemente werden dabei verwendet: JLabel, JComboBox, JTextField (weiß hinterlegt), JButton.

Um die verschiedenen Texte einzugeben, findest du ganz rechts im „**Propertie-Fenster**“ für das aktuelle ausgewählte Gui-Objekt alle **Properties**, also Eigenschaften der Gui-Objekte. Bei einem Label oder einem Textfeld kannst du den Text, der im Label bzw. dem Textfeld erscheint, über die Eigenschaft „text“ eingeben. Die Auswahlmöglichkeiten, die die Combobox später bietet, gibst du in der Eigenschaft „model“ mit Komma voneinander getrennt ein.

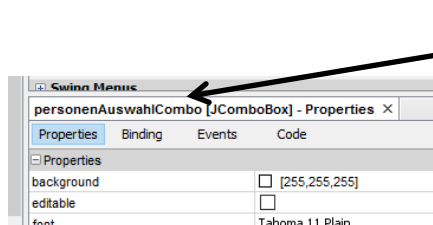
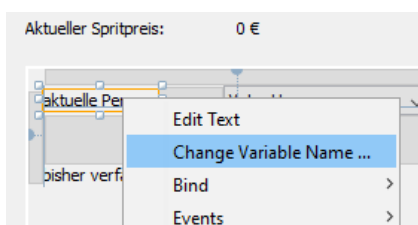


Für Comboboxen kannst du in der Eigenschaft **model** angeben, zwischen welchen Zeilen einträge der Benutzer auswählen kann. Die einzelnen Einträge werden durch Komma getrennt.

Gui-Objekten einen Namen geben


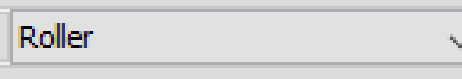
Schließlich geben wir allen Objekten, die wir später im Programm verändern müssen, einen Namen: nach Rechtsklick auf ein Objekt wählen wir im erscheinenden Fenster „Change Variable Name...“ und geben den Namen ein. Bei Objekten in Java benutzen wir folgende Vereinbarung: **Objektnamen** beginnen grundsätzlich mit einem kleinen Buchstaben. Bestehen die Objekte aus mehreren Wörtern, werden alle weiteren Wörter am Anfang groß geschrieben, aber ohne Leerzeichen aneinander gesetzt, also z. B. **personenAuswahlCombo** oder **getanktLiterLab**. Auf diese Weise wird man später alleine an der Schreibweise erkennen können, ob es sich um ein Objekt oder um eine Klasse handelt, denn Klassen werden mit dem ersten Buchstaben großgeschrieben: **JComboBox** ist die Klasse, zu der die beiden Objekte **personenAuswahlCombo** und **getanktLiterLab** gehören. Bei einigen Labels findest du auf S. 8 keinen Namen: hier brauchst du den von Netbeans vorgeschlagenen Namen nicht verändern, da der angezeigte Text niemals verändert werden muss.

Klickt man auf ein Objekt, wird der Objektname oberhalb des Property-Fensters angezeigt:



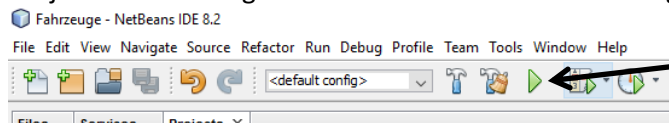
Der Objektname wird oberhalb des Propertie-Fensters angezeigt, in eckigen Klammern dahinter steht der Name

Beispiel: Die Objekte **personenAuswahlCombo** und **fahrzeugAuswahlCombo** sind beides Objekte der Klasse **JComboBox**. Als Instanzen der JComboBox haben sie die typischen Eigenschaften einer Combobox, z. B. die Möglichkeit in der Eigenschaft „**model**“ mehrere Zeilen zur Auswahl anzubieten:

Objekt personenAuswahlCombo: 	Property model: Vater Hans, Mutter Susi, Clara, Tim	Eigenschaften der Klasse JComboBox: <table border="1"> <tr><td>background</td><td><input type="checkbox"/> [255,255,255]</td></tr> <tr><td>editable</td><td><input type="checkbox"/></td></tr> <tr><td>font</td><td>Tahoma 11 Plain</td></tr> <tr><td>foreground</td><td><input type="checkbox"/> [0,0,0]</td></tr> <tr><td>maximumRowCount</td><td>8</td></tr> <tr><td>model</td><td>Vater Hans, Mutter Susi, Clara, Tim</td></tr> <tr><td>selectedIndex</td><td>0</td></tr> <tr><td>selectedItem</td><td>Vater Hans</td></tr> <tr><td>toolTipText</td><td></td></tr> </table>	background	<input type="checkbox"/> [255,255,255]	editable	<input type="checkbox"/>	font	Tahoma 11 Plain	foreground	<input type="checkbox"/> [0,0,0]	maximumRowCount	8	model	Vater Hans, Mutter Susi, Clara, Tim	selectedIndex	0	selectedItem	Vater Hans	toolTipText	
background	<input type="checkbox"/> [255,255,255]																			
editable	<input type="checkbox"/>																			
font	Tahoma 11 Plain																			
foreground	<input type="checkbox"/> [0,0,0]																			
maximumRowCount	8																			
model	Vater Hans, Mutter Susi, Clara, Tim																			
selectedIndex	0																			
selectedItem	Vater Hans																			
toolTipText																				
Objekt fahrzeugAuswahlCombo: 	Property model: Roller, Harley, Honda																			

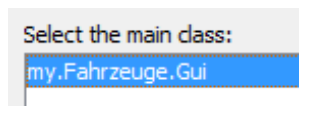
Ein Java-Programm starten

Wir haben bisher die Klassen **JPanel**, **JComboBox**, **JLabel**, **JButton** und **JTextField** innerhalb des Guis verwendet. Es wird Zeit, unser Programm zum ersten Mal zu starten. Wähle in der oberen Icon-Zeile den grünen Pfeil aus. Die gleiche Funktion erreichst du über das Menü unter dem Reiter „Run“ über „Run Projekt“. Netbeans generiert ein ausführbares Java-Programm und startet dieses.



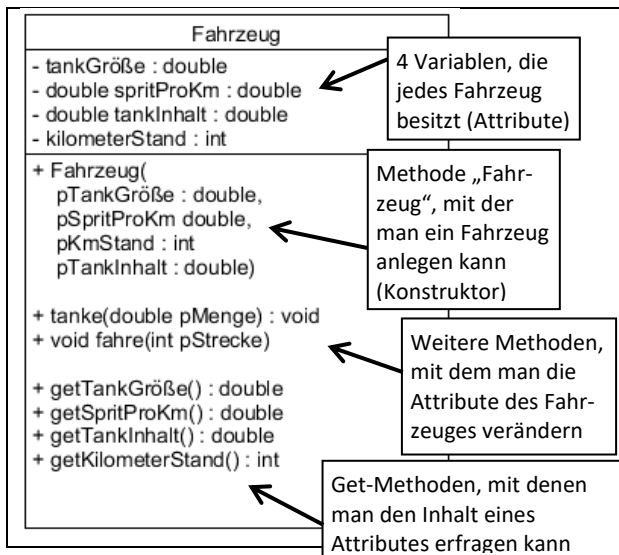
Das Programm wird mit dem Run-Icon gestartet.

Wenn du das Programm zum ersten Mal ausführst, wirst du gefragt, welche Klasse die **main-class** (Hauptklasse) sein soll. Der Vorschlag **my.Fahrzeuge.Gui** ist hier richtig, denn die Klasse **Gui** ist bisher unsere einzige Klasse, die zu Beginn ausgeführt werden soll.



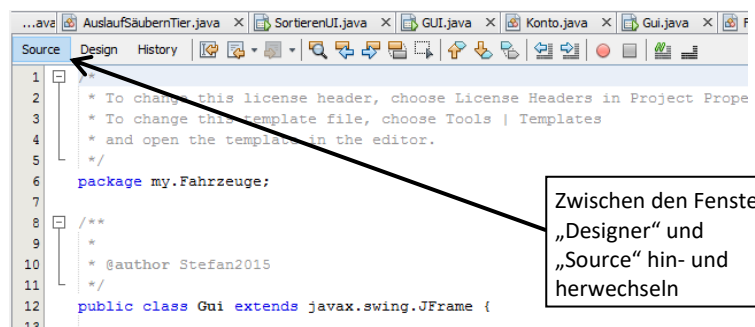
Aufgabe 2: Die Roller lernen fahren – Wir legen die Klasse Fahrzeug an

Wenn du nach dem Programmstart auf die Buttons „Fahre eine Strecke“ oder „Tanke das Fahrzeug“ klickst, passiert garnichts, das werden wir jetzt ändern. Als erstes werden wir die drei Motorräder der Familie anlegen. Die drei Motorräder „harley“, „honda“ und „yamaha“ sind Objekte der Klasse „Fahrzeuge“, denen wir genauso wie dem Rover in Greenfoot Methoden und Eigenschaften zuweisen müssen:



Verschiedene Fahrzeuge unterscheiden sich in vier **Variablen**: Der Tankgroesse (**tankgroesse**), dem Spritverbrauch pro gefahrenem Km (**spritproKM**), dem aktuellen Inhalt des Tankes in Litern (**tankinhalt**) und dem aktuellen Kilometerstand (**kilometerstand**). Mit der besonderen Methode „**Fahrzeug**“ (diese heißt immer genauso wie die Klasse selbst) kann man ein neues Fahrzeug erzeugen, indem man den angegeben Variablen einen Wert zuweist. Diese besondere Methode nennt man auch den **Konstruktor der Klasse**. 6 weitere Methoden (z. B. „**fahre**“ oder „**tanke**“) kann man dazu benutzen, um den Zustand des Fahrzeugs zu verändern, oder den aktuellen Zustand einer der Variablen erfragen.

Um ein neues Fahrzeug anzulegen, wechseln wir in den eigentlichen Programmiercode (Quellcode), der in Netbeans „Source“ (engl. Quelle) genannt wird. Oberhalb des Fensters, indem bisher die Gui-Objekte positioniert wurden, kannst zwischen dem „Designer“ und „Source“ hin- und herwechseln.



Zwischen den Fenstern „Designer“ und „Source“ hin- und herwechseln

Wechsle also in das Source-Fenster. Nach einigen Kommentarzeilen wird angezeigt, dass wir uns innerhalb des Packages `my.Fahrzeuge` befinden. Darunter hat Netbeans bereits die Klasse „Gui“ angelegt, die die Ausgabe unseres Desktops (Bildschirmoberfläche) steuert. Die Zeile `public class Gui extends javax.swing.JFrame` bedeutet, dass Gui eine Unterklasse von JFrame ist, also einer Klasse die den „Bildschirmrahmen“ auf dem Desktop darstellt. Darunter folgt die Methode `public Gui()`. Diese Methode wird beim Start des Programms ausgeführt, wenn eine Instanz der Klasse Gui angelegt wird. Diese Methode ist der **Konstruktor** der Klasse Gui. `initComponents()` initialisiert alle Bildschirmobjekte, die wir im Designer angelegt haben, darum müssen wir uns jetzt nicht mehr kümmern.

```

1  /*
2  * To change this license header, choose License Headers in Project Properties.
3  * To change this template file, choose Tools | Templates
4  * and open the template in the editor.
5  */
6  package my.Fahrzeuge;
7
8  /**
9   *
10  * @author Stefan2015
11  */
12  public class Gui extends javax.swing.JFrame {
13
14      /**
15       * Creates new form Gui
16       */
17      public Gui() {
18          initComponents();
19      }
20
21      /**
22       * This method is called from within the constructor to initialize the form.
23       * WARNING: Do NOT modify this code. The content of this method is always
24       * regenerated by the Form Editor.
25       */
26      @SuppressWarnings("unchecked")
27      Generated Code
45

```

Alle Zeilen, die in grau gedruckt sind, beinhalten Kommentare, die die Lesbarkeit des Programms verbessern sollen. Ein Kommentar wird mit dem Kürzel „/*“ eingeleitet. Weitere Kommentarzeilen beginnen mit „*“. Die letzte Zeile eines Kommentars beginnt mit „*/“.

Hier beginnt die Klasse Gui

Bei der Ausführung eines Java-Programms werden die Kommentarzeilen nicht berücksichtigt.

Direkt hinter der Zeile `initComponents()` fügen wir jetzt drei neue Zeilen ein:

```

roller = new Fahrzeug(20,0.03,100,10);
harley = new Fahrzeug(40,0.05,100,20);
honda = new Fahrzeug(50,0.04,100,15);

```

Beispiel Fahrzeug roller: Die Tankgröße ist 20l, der Spritverbrauch ist 0,03 Liter pro Kilometer (also 3 Liter pro 100 km), der Kilometerstand ist 100km und der aktuelle Tankinhalt beträgt 10 Liter.

Jede Zeile bewirkt, dass ein neues Fahrzeugobjekt angelegt wird: roller ist der Name des ersten Fahrzeugobjektes, Fahrzeug ist die zugehörige Klasse. Es folgen vier Zahlen, die folgende Bedeutung haben sollen: Die Tankgröße ist 20l, der Spritverbrauch ist 0,03 Liter pro Kilometer (also 3 Liter pro 100 km), der Kilometerstand ist 100km und der aktuelle Tankinhalt beträgt 10 Liter. Das Wort „new“ wird immer benötigt, wenn ein neues Objekt angelegt werden soll. Diesen Prozess nennt man **Instanziierung**, d. h. es wird eine Instanz (eine Exemplar) der Klasse Fahrzeug angelegt, dabei wird dem Objekt soviel Speicherplatz zugewiesen, wie es z. B. für alle Feldvariablen usw. benötigt.

Vor der Zeile `public Gui()` fügen wir noch die Zeile ein

`Fahrzeug roller,harley,honda;`

Damit teilen wir Netbeans mit, dass ab sofort die Objekte roller, harley und honda Objekte der Klasse **Fahrzeuge** sein sollen. Werden diese Objekte **vor dem Konstruktor public Fahrzeug()** deklariert, kann man innerhalb der gesamten Klasse darauf zugreifen. Diese Mitteilung nennt man auch **Deklaration**.

Wenn du die Zeilen eingegeben hast, siehst du, dass der Klassenname Fahrzeug jeweils rot unterschlängelt ist. Wenn du mit der Maus oberhalb des Wortes „Fahrzeug“ bist, erhältst du in einem gelben Kasten eine Fehlermeldung, die besagt, dass Netbeans die Klasse „Fahrzeug“ nicht kennt (s. folgende Seite):

```
public class Gui extends javax.swing.JFrame {

    Fahrzeug1 roller;

    Fahrzeug harley,honda

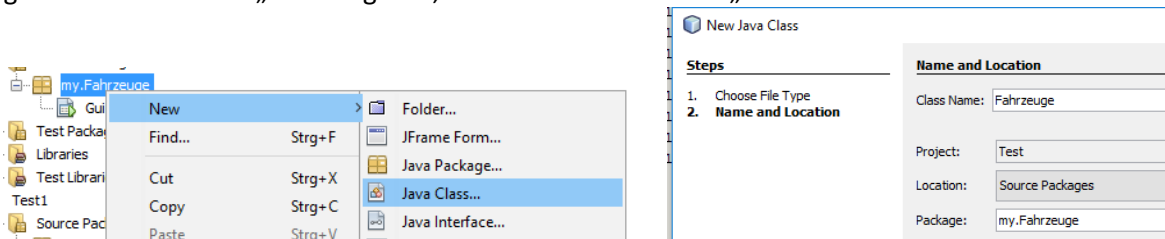
    public Gui () {
        initComponents ();
        roller = new Fahrzeug(20,0.03,100,10);
        harley = new Fahrzeug(40,0.05,100,20);
        honda = new Fahrzeug(50,0.04,100,15);
    }
}
```

cannot find symbol
symbol: class Fahrzeug
location: class Gui

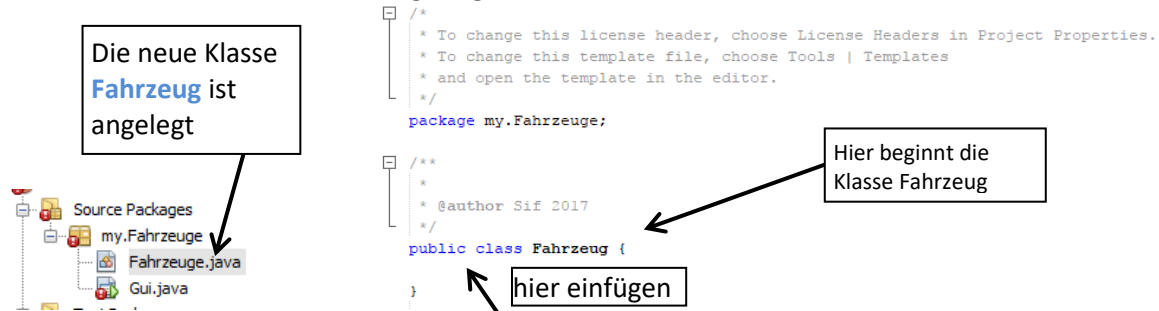
(Alt-Enter shows hints)

Ein weitere Klasse erzeugen

Dies ist nicht verwunderlich, denn bisher haben wir noch gar nicht festgelegt, was eigentlich genau ein Fahrzeug sein soll, welche Variablen und Methoden diese Klasse haben soll. Dies werden wir jetzt nachholen, indem wir zunächst **die neue Klasse „Fahrzeug“ anlegen**. Klicke dazu im Projektfenster (linke Seite) mit der rechten Maustaste auf dein Package „my.Fahrzeuge“. Wähle dann „New“, „Java Class“ und gebe bei Class Name „Fahrzeug“ ein, klicke anschließend auf „Finish“.



Im Projektfenster erkennst du, dass die neue Klasse „Fahrzeuge“ angelegt wurde. Außerdem wird im Source-Fenster die neue Klasse bereits angezeigt:



Die neue Klasse Fahrzeug ist angelegt

Hier beginnt die Klasse Fahrzeug

hier einfügen

Fahrzeuge wird im Projekt-Fenster angezeigt

Die frisch angelegte Klasse Fahrzeuge im Source-Fenster

Ab jetzt kannst du zwischen den beiden Klassen Fahrzeuge und Gui hin- und herwechseln, indem du auf die Klassennamen im **Projektfenster** doppelklickst. Wir bleiben aber zunächst in der Klasse Fahrzeuge:

Direkt **nach der Zeile** „public class Fahrzeuge {“ fügen wir die folgenden Zeilen ein. Achte darauf, dass die Zeilen zwischen die öffnende „{“ und schließende „}“ Klammer der Klasse Fahrzeuge geraten:

In gelben Feldern findest du ab jetzt längere Quelltextpassagen, die du über **Kopieren (Strg + C)** und **Einfügen (Strg + V)** direkt in deinen Quellcode kopieren kannst.

Felder und der Konstruktor

```
private double tankGröße;
private double spritProKm;
private double tankInhalt;
private int kilometerStand;

public Fahrzeug(double pTankGröße, double pSpritProKm, int pKmStand, double pTankInhalt) {
    tankGröße = pTankGröße;
    spritProKm = pSpritProKm;
    kilometerStand = pKmStand;
    tankInhalt = pTankInhalt;
}
```

```
private double tankGröße;
private double spritProKm;
private double tankInhalt;
private int kilometerStand;

public Fahrzeug(double pTankGröße, double pSpritProKm, int pKilometerStand, double pTankInhalt) {
    tankGröße = pTankGröße;
    spritProKm = pSpritProKm;
    kilometerStand = pKilometerStand;
    tankInhalt = pTankInhalt;
}
```

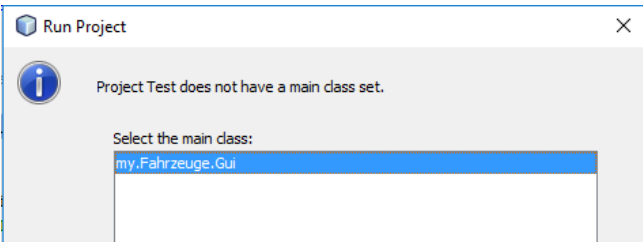
Zunächst werden 4 **Variablen** eingefügt, die wir benötigen, um ein Fahrzeug zu verwalten. Das Wort private bedeutet, dass die Variablen nur innerhalb der Klasse Fahrzeuge benutzt und verändert werden können. Solche Variablen werden auch **Attribute** genannt. In anderen Klassen sind sie jedoch nicht sichtbar: Dies hat den Sinn, dass nur der Programmierer, der die Klasse Fahrzeuge programmiert, das Recht bekommen soll, die Fahrzeugdaten zu verwalten.

Danach kommt der sogenannte „**Konstruktor**“. Er beginnt mit dem Schlüsselwort **public**, hat den selben Namen wie die Klasse (also „**Fahrzeuge**“) und in Klammern folgen alle **Parameter**, die beim Anlegen eines Fahrzeugobjektes mit Zahlen belegt werden sollen. Die Reihenfolge der Parameter entspricht dabei den Zahlen, so wie wir sie beim new-Befehl (S. 7 oben) angelegt haben. Da diese Parameter „**pSpritProKm**“, „**pKmStand**“ usw. nur innerhalb des Konstruktors bekannt sind (sogenannte **lokale Variablen**), werden sie sofort nach der geschweiften Klammer an die **Attribute** tankgroesse, spritgroesse usw. übergeben, die – wie oben gesagt – in der gesamten Klasse **Fahrzeuge** benutzt werden können. Konstruktoren haben eine Besonderheit: Nach dem Wort „**public**“ fehlt die Angabe, ob der Konstruktor eine Variable zurückgeben soll (z. B. void, int, String usw.): Konstruktoren geben **nie** Variablenwerte zurück.

Dieser Prozess noch mal ganz **langsam in Zeitlupe zum Mitdenken**: Durch den Befehl „roller = new Fahrzeug(20,0.03,100,10);“ in der Klasse Gui wird die erste Zahl 20 zunächst an den Parameter pTankgroesse übergeben, anschließend wird die Zahl 20 weiterübergeben an das Attribut **tankGröße**. Innerhalb dieses Attributes könnte nun die Zahl 20 Liter verändert werden, immer bezogen nur auf das Objekt **roller**. Gleiches gilt für die drei weiteren Zahlen bzw. Variablen. Parameter sollten übrigens immer mit dem Buchstaben **p** beginnen, dann erkennt man ganz einfach, dass es sich um einen Parameter handelt.

Wenn du alles richtig gemacht hast, sollten in der Klasse **Gui** alle roten Unterschlängelungen verschwunden sein. Am rechten Bildschirmrand des Source-Fensters kannst du ebenfalls erkennen, ob noch Fehler vorhanden sind: Fehlerzeilen werden durch einen **roten Strich** dargestellt. Wenn du mit der Maus auf einen Strich fährst, erhältst du sogar die passenden Fehlermeldung, die in der betreffenden Zeile aufgetreten ist. Durch Doppelklicken auf einen roten Strich gelangst du ebenfalls direkt an die Stelle, an die der Fehler aufgetreten ist.

Wenn du das Programm startest, wirst du gefragt, welche „**main Class**“ (also „Hauptklasse“) gewählt werden soll. Der Vorschlag **my.Fahrzeuge.Gui** ist korrekt. Es bedeutet, dass bei Programmstart immer die Methode „**main**“ aus der Klasse **Gui** aufgerufen wird. Diese Methode sieht etwas kompliziert aus, sie bewirkt, dass bei Programmstart ein Objekt der Klasse Gui angelegt wird, so dass der Desktop gezeichnet werden kann. Du brauchst dich darum nicht weiter zu kümmern und solltest den Inhalt der Methode main nicht verändern.



Auswahl der Methode main in der Klasse Gui

```

/*
 * public static void main(String args[]) {
 *   /* Set the Nimbus look and feel */
 *   Look and feel setting code (optional)
 *
 *   /* Create and display the form */
 *   java.awt.EventQueue.invokeLater(new Runnable() {
 *       public void run() {
 *           new Gui().setVisible(true);
 *       }
 *   });
 * }
 */

```

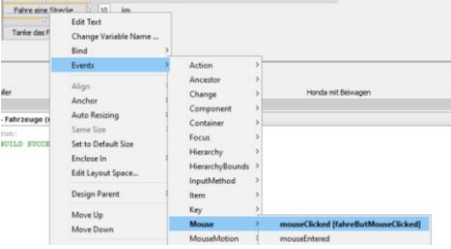
Die Definition der Methode main

Nach dem Programmstart hat sich noch nicht viel verändert. Wir sind jedoch jetzt soweit, dass wir die Methoden Fahren und Tanken programmieren können, die den Roller bewegen:

Aufgabe 3: Die Roller lernen fahren – Wir Programmieren die Methode Fahren

Wenn der Benutzer auf den Button „Fahre“ klickt, soll der Roller die Strecke fahren, die im Textfeld „Streckentext“ angezeigt wird. Dabei muss der Kilometerstand erhöht und der Tankinhalt verringert werden. Um diese Aktion zu realisieren, sind drei Dinge notwendig:

- 1.) In der Klasse Gui muss der Computer reagieren, wenn auf den Button „Fahre“ geklickt wird. Dazu dient eine sogenannte „**Event-Methode**“: Wechsle zur Klasse **Gui**, gehe in den Designer und klicke mit der rechten Maustaste auf den Button „fahreBut“.
Im erscheinenden Fenster klicke auf „Events“, Mouse“, „mouseClicked“. Daraufhin legt Netbeans die Methode „fahreButMouseClicked“ an, die genau dann aufgerufen wird, wenn der Benutzer auf den Button „fahreBut“ klickt. Den Kommentar „add your handling code here“ kannst du löschen und stattdessen den Befehl eingeben, den Roller fahren zulassen:



```

private void fahreButMouseClicked(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
}

```



```
private void fahreButMouseClicked(java.awt.event.MouseEvent evt) {
    int strecke = Integer.parseInt(streckenText.getText());
    roller.fahre (strecke);
}
```

```
int strecke = Integer.parseInt(streckenText.getText());
roller.fahre (strecke);
```

In der ersten Zeile wird zunächst die Streckenlänge ausgelesen, die im Textfeld (jTextField) **streckenText** steht, dies geht mit der Methode **getText()**, die auf das Objekt **streckenText** angewendet wird. Die Methode liefert als Ergebnis einen String, der mit dem Befehl **Integer.parseInt** in eine Integervariable umgewandelt wird. Das Ergebnis speichern wir in der Integer-Variable **strecke**. Anschließend wird die Methode **fahre (strecke)** aufgerufen, die den Roller um die Kilometerzahl fahren lassen soll, die in der Variablen **strecke** steht.

2.) Die Methode **fahre** muss nun in der Klasse **Fahrzeug** eingefügt werden, da alle Daten des Rollers dort abgerufen werden können. Wir wechseln also in die Klasse **Fahrzeug** und geben dort nach dem Konstruktor **Fahrzeug** die folgende Methode ein:

```
public void fahre(int pStrecke) {
    double spritVerbrauch = spritProKm * pStrecke;
    kilometerStand = kilometerStand + pStrecke;
    tankInhalt = tankInhalt - spritVerbrauch;
}
```

```
public void fahre(int pStrecke) {
    double spritVerbrauch = spritProKm * pStrecke;
    kilometerStand = kilometerStand + pStrecke;
    tankInhalt = tankInhalt - spritVerbrauch;
}
```

Wie viele Kilometer gefahren werden sollen, wird zunächst in der Variablen **pStrecke** gespeichert. In der Variable **spritVerbrauch** wird der benötigte Sprit berechnet. Die Variablen **kilometerStand** und **tankInhalt** werden aktualisiert:

3.) Nach dem der Roller gefahren ist, soll der neue Kilometerstand und der neue Tankinhalt mit dem Gui in den Labels **tankInhaltLab** bzw. **kilometerStandLab** angezeigt werden. Dazu muss der Inhalt der beiden Variablen **kilometerStand** und **tankInhalt** aus der Klasse **Fahrzeuge** in der Klasse **Gui** gelesen werden können. Wir fügen dazu in der Klasse **Fahrzeuge** zwei sogenannte **get-Methoden** ein. Mit einer get-Methode kann man den Wert eines Attributes aus der Klasse **Fahrzeug** an die Klasse **Gui** weitergeben:

get-Methoden anlegen

```
public int getKilometerStand() {
    return kilometerStand;
}

public double getTankInhalt() {
    return tankInhalt;
}
```

```
public int getKilometerStand() {
    return kilometerStand;
}

public double getTankInhalt() {
    return tankInhalt;
}
```

Mit dem Befehl **return** wird im ersten Fall der Inhalt der Variable **kilometerStand**, im zweiten Fall der Inhalt der Variablen **tankGröße** zurückgegeben.

In der Klasse **Gui** ergänzen wir jetzt die Methode **fahreMouseClicked** (die ersten beiden Zeilen hast du bereits auf dieser Seite oben geschrieben): Mit der Methode **roller.getTankinhalt()** wird der aktuelle Tankinhalt des Rollers nach dem Fahren ermittelt, **f2.format** bewirkt, dass die Variable vom Typ **double** in einen String mit 2 Dezimalstellen umgewandelt wird. Dazu wird vorher das Dezimalformat **f2** definiert (**#0.00** gibt zwei Stellen hinter dem Komma aus). Mit dem Befehl **tankInhaltLab.setText(...)** wird der formatierte String innerhalb des Labels **tankInhaltLab** geschrieben. Ganz analog wird ebenfalls im Label **kilometerStandLab** der aktuelle Kilometerstand ausgegeben.

Die Methode fahreMouseClicked()

```
private void fahreButMouseClicked(java.awt.event.MouseEvent evt) {
    int strecke = Integer.parseInt(streckenText.getText());
    roller.fahre (strecke);
    DecimalFormat f2 = new DecimalFormat("#0.00");
    DecimalFormat f1 = new DecimalFormat("#0.0");
    tankInhaltLab.setText (f2.format(roller.getTankinhalt()) + " Liter");
    kilometerStandLab.setText (String.valueOf(roller.getKilometerStand()) + " km");
}
```

```
DecimalFormat f1 = new DecimalFormat("#0.0");
DecimalFormat f2 = new DecimalFormat("#0.00");
tankInhaltLab.setText (f2.format(roller.getTankinhalt()) + " Liter");
kilometerStandLab.setText (String.valueOf(roller.getKilometerStand()) + " km");
```

```
import java.text.*;
```

Tipp: 1.) Wenn du die beiden Zeilen, die das Dezimalformat steuern, **vor** dem Konstruktor der Klasse **Gui** unterbringst, kannst du dir die Wiederholung dieser Zeilen in späteren Methoden sparen.
2.) Direkt hinter der ersten package-Zeile muss eingefügt werden: **import java.text.*;** Dadurch bindet Java die benötigten Befehle ein.

Die letzten beiden Zeilen, die die Texte der beiden Labels für Tankinhalt und Kilometerstand setzen, solltest du noch an das Ende des Konstruktors kopieren, damit die Labels zu Beginn des Programms richtig gesetzt werden. Wenn du jetzt dein Programm startest, sollte nach Mausklick auf den Button **fahrenBut** tatsächlich die Fahrt des Rollers simuliert werden, indem sich der Kilometerstand und Tankinhalt des Rollers verändern. Probiere auch aus, was passiert, wenn du eine andere Streckenlänge fährst.

Hurra, der Roller fährt zum ersten Mal!

Nochmal im Überblick, was wir in der letzten Aufgabe in den beiden Klassen Gui und Fahrzeuge gemacht haben:

Klasse Gui:	Klasse Fahrzeug:
<pre>fahreButMouseClicked() { roller.fahre (strecke); }</pre> <p>fahreButMouseClicked ist eine Event-Methode, die Ausgeführt wird, wenn der Benutzer auf den Button klickt.</p> <p>Innerhalb der Eventmethode wird die Methode roller.fahre (strecke) aufgerufen.</p> <p>...</p> <pre>tankInhaltLab.setText (f2.format(roller.getTankInhalt()) + " Liter"); kilometerStandLab.setText (String.valueOf(roller.getKilometerStand()) + " km");</pre> <p>Innerhalb der Eventmethode wird über die Methode getTankInhalt() der veränderte Tankinhalt abgefragt und in das passende Label geschrieben.</p>	<pre>public void fahre(int pStrecke) { double spritVerbrauch = spritProKm * pStrecke; kilometerStand = kilometerStand + pStrecke; tankInhalt = tankInhalt - spritVerbrauch; }</pre> <p>Innerhalb des Objektes Roller werden die Variablen kilometerstand und tankinhalt verändert.</p> <pre>public double getTankInhalt() { return tankInhalt; }</pre> <p>Innerhalb der Methode getTankInhalt wird der aktuelle Tankinhalt zur aufrufenden Methode in der Klasse Gui zurückgegeben.</p>

Tip: Wenn du eine neue Event-Prozedur erzeugst, findest du anschließend im Code häufig einen sehr langen Zeilenbereich, der grau hinterlegt ist. Dies sind die Zeilen, die Netbeans automatisch angelegt hat, als du die einzelnen Gui-Objekte angelegt hast. Da der Text eigentlich nur stört und du diesen auch nicht ohne Weiteres verändern kannst, macht es Sinn, diesen wieder zu verstecken: Dies macht man, indem man auf eines der beiden Minuszeichen am Beginn des grauen Bereiches tippt. Aus dem „-“ wird dabei ein „+“. Durch Klick auf das „+“-Zeichen klappt der Text wieder auf.



In Grau erscheint der Quellcode, der alle Angaben enthält, die du im Gui-Designer gemacht hast

Aufgabe 4: Wenn der Benutzer auf den Button **tankeBut** klickt, soll das Motorrad betankt werden und zwar mit so viel Liatern, wie es im Objekt **literText** angegeben ist. Füge die Methode **tankeButMouseClicked** als Eventmethode des Buttons **fahreBut** ein. Genauso wie auf S. 3 Oben benötigst du eine neue Variable, z. B. **literZahl**, in der die Literzahl gespeichert wird. Danach wird die neue Methode **roller.tanke (literZahl)** ausgeführt. Diese neue Methode musst du jetzt in der Klasse Fahrzeug einfügen. Beim Tanken muss dort lediglich das Attribut **tankinhalt** erhöht werden.

<pre>public void tanke (int pLiter){ }</pre>	<p>Innerhalb der Klasse Fahrzeug wird die neue Methode tanke eingefügt. Sie benötigt den Parameter pLiter, indem der Wert der literZahl aus der Klasse Gui übergeben wird. Du musst noch innerhalb der Methode tanke das Attribut tankinhalt um den Wert von pLiter erhöhen.</p>
---	--

Aufgabe 5: Um wirklich zu verstehen, warum man zwischen den beiden Klassen hinundherspringt, verbessern wir unser Programm jetzt soweit, dass der Benutzer zwischen drei verschiedenen Motorrädern auswählen kann:

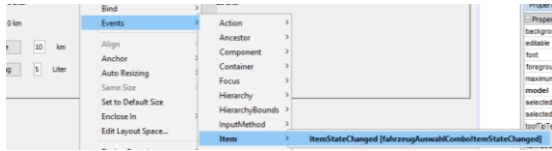
1.) Damit der Benutzer zwischen verschiedenen Motorrädern wechseln kann, muss das Programm in einem neuen Objekt speichern, welches Motorrad gerade ausgewählt ist. Dazu erzeugen wir ein neues Objekt **zweirad** der Klasse **Gui**:

<pre>public class Gui extends javax.swing.JFrame { Fahrzeug roller, harley, honda, zweirad;</pre>	<p>Das neue Fahrzeug „zweirad“</p>
---	------------------------------------

Zwischen mehreren Fahrzeugen wechseln

Ein neuer new-Befehl ist nicht notwendig, da Fahrzeug immer eines der schon existierenden Fahrzeuge roller, harley oder honda sein wird.

2.) Wir erzeugen für die Combobox `fahrzeugAuswahlCombo` die Eventmethode `ComboItemStateChanged`, die immer dann ausgeführt wird, wenn der Benutzer in der Combobox ein anderes Motorrad per Mausklick auswählt. Wir gehen dazu in den Designer der Klasse `Gui`, machen einen Rechtsklick auf das Objekt `fahrzeugAuswahlCombo` und klicken auf „Events“, „Item“, „ItemStateChanged“. Wieder wird die entsprechende Event-Methode von Netbeans angelegt:



Im folgenden Code wird zunächst über die Methode `getSelectedIndex` („ausgewählte Nummer“) bestimmt, der wie viele Eintrag in der Combobox ausgewählt wurde. Diese Nummer wird in der Variable `aktuellesFahrzeug` gespeichert. Mit dem switch-Befehl kann nun je nach Eintrag das passende Fahrzeug gewählt werden. Zum Schluss werden die beiden Labels `tankinhaltLab` und `kilometerStandLab` mit den Werten des neu ausgewählten Fahrzeugs bestückt.

```
private void fahrzeugAuswahlComboItemStateChanged(java.awt.event.ItemEvent evt) {
    DecimalFormat f2 = new DecimalFormat("#0.00");
    int aktuellesFahrzeug = fahrzeugAuswahlCombo.getSelectedIndex();
    switch (aktuellesFahrzeug) {
        case 0:
            zweirad = roller;
            break;
        case 1:
            zweirad = harley;
            break;
        case 2:
            zweirad = honda;
            break;
    }
    tankinhaltLab.setText(f2.format(zweirad.getTankinhalt()) + " Liter");
    kilometerStandLab.setText(String.valueOf(zweirad.getKilometerstand()) + " km");
}
```

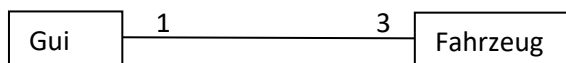
Der Switch-Befehl:

Genauso, wie man beim if-else-Befehl zwischen zwei Werten einer Variable unterscheiden konnte, ermöglicht der switch-Befehl die Unterscheidung zwischen beliebig vielen Werten einer Variablen. Die Werte werden mit dem Wort „case“ benannt. Danach folgt, welche Befehle bei diesem Variablenwert jeweils ausgeführt werden sollen. Mit „break“ werden diese Befehle für einen Variablenwert abgeschlossen.

```
int aktuellesFahrzeug =
fahrzeugAuswahlCombo.getSelectedIndex();
switch (aktuellesFahrzeug){
    case 0:
        zweirad = roller;
        break;
    case 1:
        zweirad = harley;
        break;
    case 2:
        zweirad = honda;
        break;
}
```

3.) In den Methode `fahreButMouseClicked()` und `tankeButMouseClicked()` musst du noch an insgesamt fünf Stellen das Objekt `roller` durch das Objekt `zweirad` ersetzen, damit die Fahrt mit dem jeweils ausgesuchten Fahrzeug geschieht. Außerdem musst du im Konstruktors nach den drei new-Befehlen die Zeile `zweirad = roller;` einfügen, damit wird zu Beginn als `Fahrzeug` der `roller` ausgewählt. Damit kann jetzt mit verschiedenen Motorrädern gefahren werden. Probiere dein neues Programm aus.

Jetzt versuchen wir nocheinmal die wichtigste Frage bei unserem Programm zu klären: **Warum arbeiten wir mit den zwei Klassen Gui und Fahrzeug:** In der Klasse `Gui` können alle Fahrzeuge aufgerufen werden, hier wird die grafische Ausgabe ausgeführt unabhängig davon, welches Fahrzeug gefahren wird. In einem Objekt der Klasse `Fahrzeug` werden für jeweils eines der drei Fahrzeuge alle wichtige Variablenwerte gespeichert und verändert. Wie du siehst, merkt sich Java die Variablenwerte aller Fahrzeuge, auch wenn man zwischen den Fahrzeugen wechselt. Man schreibt diesen Zusammenhang so auf:



Gemeint ist damit, dass von einer Klasse `Gui` 3 verschiedene Fahrzeuge verwaltet werden. Das Ganze geschieht völlig automatisch, der Programmierer muss sich nicht darum kümmern.

Aufgabe 6: Tankgröße und Verbrauch sollen für jedes Motorrad angezeigt werden, außerdem soll verhindert werden, dass man noch weiterfährt, wenn der Tank bereits leer ist.

1. Für das `fahrzeugPanel` muss das Programm noch so erweitert werden, dass die beiden Label `tankgroesseLab` und `verbrauchLab` die Größe des Tankes und den Verbrauch des Motorrades für eine 100 km lange Strecke korrekt anzeigen. Die beiden Variablen `tankinhalt` und `spritProKm` wurden dazu bereits im Konstruktor der Fahrzeuge angelegt. Ergänze zuerst in der Klasse `Fahrzeuge` die beiden get-Methoden `getTankGrossesse()` und `getSpritProKm()`, die die Tankgröße bzw. den Spritverbrauch pro km an die Klasse `Gui` liefern sollen. Am Ende des Konstruktors der Klasse `Gui` „`public Gui()`“ können dann die beiden Label entsprechend mit den Variablenwerten versehen werden. Wenn die Anzeige im Konstruktor funktioniert,

Das Fahrzeugpanel wird fertig

kannst du die betreffenden Zeilen in die Event-methode `fahrzeugAuswahlComboItem-StateChanged` übertragen, denn die Werte ändern sich, wenn ein anderes Motorrad gewählt wird.

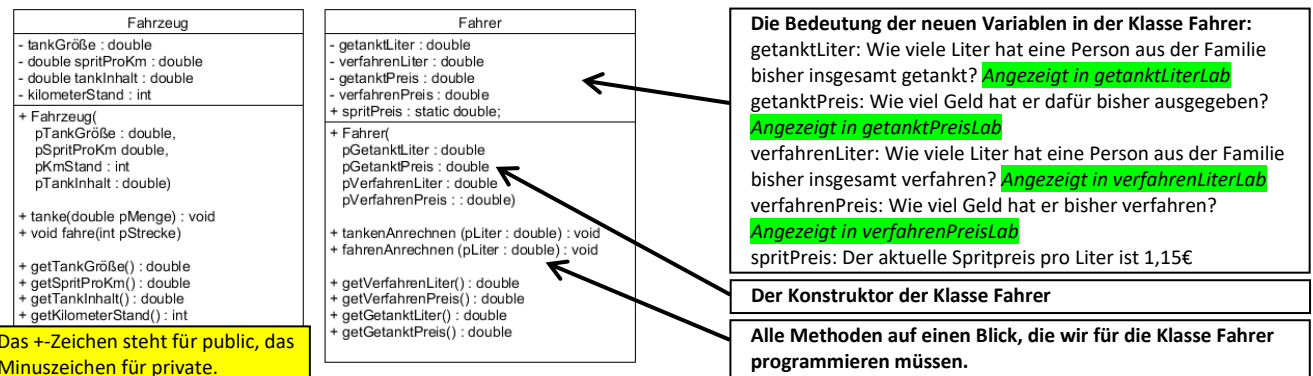
Tipp: Sowohl bei den get-Methoden in der Klasse `Fahrzeug` als auch bei der Programmierung in der Klasse `Gui` kannst du dich orientieren an den Programmzeilen, die die Beschriftung der beiden Labels `tankInhaltLab` und `kilometerStandLab` durchführen.

2. Zur Zeit kann ein Motorrad noch weiterfahren, wenn der Tank bereits leer ist: die Tankanzeige wird dann negativ, was keinen Sinn macht. Du kannst dies verhindern, indem du in der Methode `fahre(pstrecke)` mit einem if-Befehl die beiden Variablen `spritVerbrauch` und `tankInhalt` miteinander vergleichst. Ist der Tank nicht mehr voll genug, soll der Fahre-Befehl einfach ignoriert werden.

Aufgabe 7: Der Motorradmanager berücksichtigt, welche Person gerade fährt

Als letztes soll das Programm nun die einzelnen Label im `personenPanel` beschriften und dabei berücksichtigen, welche Person gerade ein Motorrad fährt. **Im `personenPanel` wird darüber Buch geführt, wieviel Liter eine Person bisher mit allen Fahrzeugen zusammen getankt hat bzw. wieviele Lieter sie bisher insgesamt mit allen Motorrädern verfahren hat.** Dazu legst du gleich die neue Klasse `Fahrer` an: Für jedes der vier Familienmitglieder wird danach ein Objekt dieser Klasse erzeugt, die den Namen der Person trägt, also `vater`, `mutter`, `tochter` und `sohn`. Die folgende Übersicht zeigt, welche Variablen und Methoden benötigt werden. Zum Vergleich ist das **Implementationsdiagramm** für das Fahrzeug angegeben:

Zwischen mehreren Fahrern wechseln



Zur Hilfe zeigen wir dir noch einmal die Struktur der neuen Klasse `Fahrer`, die du anlegen musst:

```
public class Fahrer {
    private double getanktLiter;
    .....

    public Fahrer(double pGetanktLiter, ..... ){
        getanktLiter = pGetanktLiter;
        .....
    }

    public void tankenAnrechnen (double pLiter){
        getanktLiter = ....
    }

    public void fahrenAnrechnen (double pLiter){
        verfahrenLiter = ...
    }

    public double getGetanktLiter(){
        return ...
    }
}
```

Die Variablen werden benannt

Der Datentyp **double** lässt im Gegensatz zum Datentyp **int** zusätzlich Komazahlen zu.

Der Konstruktor wird geschrieben

Die einzelnen Methoden werden geschrieben:
tankenAnrechnen: die Werte in den Variablen `getanktLiter` und `getanktPreis` müssen erhöht werden, wenn `pLiter` getankt wurden
fahrenAnrechnen: die Werte in den Variablen `verfahrenLiter` und `verfahrenPreis` müssen erhöht werden, wenn `pKilometer` gefahren wurden

Bei der Variablen `spritPreis` gibt es eine Besonderheit: Da der Spritpreis für alle Fahrer und Fahrzeuge gleich ist, wird er als „**static**“ definiert: Es gibt diese Variable dann nur ein einziges Mal, die Variable wird nicht für jedes einzelne Fahrzeug getrennt geführt.

Nachdem du die Klasse `Fahrer` angelegt hast, musst du weitere Schritte in der Klasse `Gui` vollenden:

- 1.) Die Objekte `vater`, `mutter`, `tochter`, `sohn` und `person` werden als neue Objekte benannt. „`person`“ steht dabei für die aktuell über die `ComboBox personenAuswahlCombo` ausgewählte Person.
- 2.) Die Objekte werden mit dem `new`-Befehl angelegt. Alle Variablenwerte können dabei zu Beginn 0 sein. Das Objekt `person` wird zu Beginn als `vater` definiert (`person = vater`).

3.) Die Event-Methode `personenAuswahlComboItemStateChanged` wird angelegt: Wenn der Benutzer ein Familienmitglied auswählt, wird mit einem switch-Befehl der Fahrer `person` auf dieses Familienmitglied gesetzt:

```
private void personenAuswahlComboItemStateChanged(java.awt.event.ItemEvent evt) {
    aktuellePerson = personenAuswahlCombo.getSelectedIndex();
    switch (aktuellePerson){
        case 0:
            person = vater;
            break;
        case 1: ....
    }
```

Danach werden die Label `verfahrenLiterLab`, `verfahrenGeldLab`, `getanktLiterLab`, `getanktGeldLab` entsprechend mit den richtigen Variablenwerten versorgt.

4.) In der Methode `tankeButMouseClicked` wird der Befehl `person.tankenAnrechnen (tanken)`; ausgeführt. In der Klasse `Fahrer` werden dabei in der Methode „tankenAnrechnen (double pLiter)“ die Variablen `getanktLiter` und `getanktPreis` gesetzt. **Tipp:** `getanktLiter = getanktLiter +`;

Die neue Methode soll also berechnen, wieviel Liter die Person bisher getankt hat und wieviel Geld sie dazu ausgegeben hat. Anschließend können die Label `getanktLiterLab` und `getanktGeldLab` gesetzt werden.

Ganz analog werden wieder in der Klasse `Gui` in der Methode `fahreButMouseClicked` mit dem Befehl `person.fahrenAnrechnen (strecke * zweirad.getSpritProKm())` die Variablen `verfahrenLiter` und `verfahrenPreis` gesetzt. Dazu muss die Methode `fahrenAnrechnen (double pLiter)` programmiert werden. Die neue Methode soll also berechnen, wieviel Liter die Person bisher verfahren hat und wieviel Geld sie dazu benötigt hat. Anschließend können die Label `verfahrenLiterLab` und `verfahrenGeldLab` gesetzt werden.

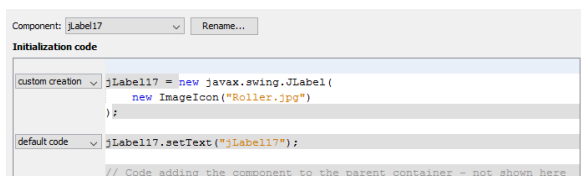
Zusatzaufgabe 1: Verbessere die Methode `tanke` so, dass der Roller nur tankt, wenn noch genug Platz im Tank ist.

Zusatzaufgabe 2: Der Spritpreis soll mit zwei Buttons in Cent-Schritten vergrößert bzw. verkleinert werden können. In der Klasse `Fahrer` wird dazu eine neue Methode `getSpritPreis` notwendig, die den aktuellen Spritpreis zurückgibt. Außerdem muss mit einer Methode `setSpritPreis (double pSpritPreis)` der Spritpreis verändert werden können:

```
public void setSpritPreis(double pSpritPreis){
    spritPreis = pSpritPreis;
}
```

Zusatzaufgabe 3: Das Programm hat noch einen Schönheitsfehler: Wenn zum Tanken nicht mehr genug Platz im Tank ist oder zum Fahren genug Benzin fehlt, verhindert zwar der if-Befehl aus Zusatzaufgabe 1, dass der Tank überläuft bzw. der Tankinhalt negativ wird. Der Person wird jedoch der Tank- bzw. Fahrvorgang immer noch im `personenPanel` angerechnet (Probiere das aus!). Dies kann man so verhindern: In der Klasse `Fahrzeug` werden die beiden neuen Variablen `tatsächlichGetankt` bzw. `tatsächlichGefahren` vom Typ `boolean` angelegt. Die beiden Variablen werden in den Methoden `fahren` bzw. `tanken` gesetzt. In der Klasse `Gui` werden die beiden Variablen mit einer get-Methode abgefragt und nur wenn das Ergebnis `true` ist, werden die weiteren Befehle, mit denen die Variablen für die betreffende Person gesetzt werden, ausgeführt.

Zusatzaufgabe 4: Lege die Bilder der Motorräder an. Die Bilder, die du auf S. 1 sehen kannst, sind jpgs, die du im Lehrerverzeichnis findest. Kopiere diese in dein Projektverzeichnis „Fahrzeuge“. Lege anschließend drei Labels an. Größe und Position kannst du aus der Grafik auf S. 1 abschätzen. Damit in einem Label ein Bild angezeigt wird, muss der Code beim Erzeugen des Labels verändert werden. Klicke dazu mit der rechten Maustaste auf das Label, das den Roller darstellen soll. Wähle in der Auswahlliste den Befehl „Customize Code“. Im unten dargestellten Fenster muss beim Anlegen des Labels die Einstellung „default code“ auf „custom creation“ umgestellt werden, der Benutzer darf dann den Code verändern. Füge im Codefenster daneben schließlich den Befehl „new ImageIcon(“Roller.jpg“)“ ein und verfare mit den andern beiden Bildern genauso. Probiere aus, ob die Bilder erscheinen.



Für die Zusatzaufgaben 4 und 5 sind zwei Import-Befehle in der Klasse `Gui` notwendig:

```
import java.awt.Color;
import javax.swing.ImageIcon;
```


Zusatzaufgabe 5: Im `personenPanel` soll in der untersten Zeile angezeigt werden, wie groß das Guthaben eines Fahrers ist. Das Guthaben ergibt sich aus der Differenz von getankten Litern und verfahren Litern. Üblicherweise wird ein Guthaben grün dargestellt, wenn es positiv ist und rot, wenn es negativ ist. Die Schreibfarbe steuert man mit dem Befehl `guthabenArtLab.setForeground(Color.red)`. Da die Berechnung und Desktopausgabe an mehreren Stellen erfolgen muss, ist es sehr hilfreich, wenn du eine neue Methode (z. B. `public void Guthabenberechnen()`) schreibst, und diese dann im Programm aufrufst.

Zusatzaufgabe 6: Bringe ein viertes Motorrad an den Start.

Wir schauen zurück – wichtige Begriffe, die wir im Programm verwendet haben

Aufgabe 8: Schreibe jeweils dazu Beispiele auf, die im Programm `Motorradmanager` vorkamen. Erkläre jeweils an einem deiner Beispiele, was mit dem jeweiligen Begriff gemeint ist

Klasse:
Attribute:
Konstruktor:
Objekte deklarieren:
Objekte instanzieren (erzeugen):
Methoden, die ein Fahrzeug steuern:
get-Methoden:
set-Methoden:
lokale Variablen:
statische Variablen: