technische universität dortmund

fakultät für informatik

Oliver Gasser
Alexander Lochmann

Dortmund, den 09. Januar 2021

Praktikum zur Vorlesung Einführung in die Programmierung WS 20/21

Blatt 7

Es können 12 (+ 11) Punkte erreicht werden.

Grundlage: Include-Guards

Um zu vermeiden, dass Headerdateien mehrfach inkludiert werden, ist es sinnvoll, sogenannte *Include-Guards* zu verwenden (siehe `https://de.wikipedia.org/wiki/Include-Guard`). Beim Erstellen einer Header-Datei (`.h`-Datei) mittels `File` → `New` → `Header-File` oder beim Erstellen einer Klasse mittels `File` → `New` → `Class` werden sie automatisch hinzugefügt.

Grundlage: Namenskonventionen

Um den Quellcode lesbarer zu gestalten, sollten bestimmte Richtlinien zur Namensgebung und Formatierung eingehalten werden. Typischerweise variieren diese Richtlinien von Programmiersprache zu Programmiersprache, und sogar innerhalb derselben Programmiersprache konkurrieren oft verschiedene Denkweisen. Eine weitverbreitete Namenskonvention wurde von der Programmiersprache Java geprägt. Diese beinhaltet folgende Regeln (siehe `https://en.wikipedia.org/wiki/Naming_convention_%28programming%29#Java`):

- Klassennamen sollten mit einem Großbuchstaben beginnen.
- Funktionen, Methoden und Variablen sollten mit einem Kleinbuchstaben beginnen.
- Für die oben genannten Namen gilt: klein weiterschreiben, bei zusammengesetzten Wörtern fangen Teilworte wiederum mit einem Großbuchstaben an (sogenannte *CamelCase*-Schreibweise).
- Konstanten sollten komplett in Großbuchstaben geschrieben werden.
- Verständliche (das heißt ausgeschriebene) Variablennamen benutzen, außer bei Schleifenvariablen, Zählern, oder ähnlichem.

Variablennamen lassen sich in Visual Studio Code komfortabel projektweit ändern, indem erst der zu ändernde Name markiert und dann die Taste *F2* geklickt wird.
Namenskonventionen sind Bestandteil des allgemeinen Programmierstils, der auch die übrige Formatierung des Textes (Einrückungen, etc.) umfasst (siehe `https://de.wikipedia.org/wiki/Programmierstil`). Durch die Tastenkombination `Alt + Shift + F` wird Visual Studio veranlasst, den Quelltext selbstständig zu formatieren.

Aufgabe 1: Elektrofahrzeuge (12 Punkte)

In dieser Aufgabe soll eine Klasse für Elektrofahrzeuge implementiert werden. Legen Sie dazu ein neues Projekt `Aufgabe_7` mit der Quelltextdatei `elektrofahrzeug.cpp` und der zugehörigen Header-Datei `elektrofahrzeug.h` an. Die Header-Datei soll die Klassendefinition enthalten und die Quelltextdatei die zugehörige Implementierung.

HINWEIS: Für diese Aufgabe, sowie weitere Aufgaben in der Übung und im Praktikum ist es notwendig, Visual Studio Code so umzukonfigurieren, dass es mehrere *.cpp*-Dateien übersetzen und zu einem Programm zusammenbinden (*linken*) kann.
Dazu muss die Konfiguration der *Code Runner*-Extension (wie im Abschnitt *Einrichten und Verwenden von Visual Studio Code* der Installationsanleitung) angepasst werden:

- *File* → *Preferences* → *Settings*, dort den Punkt *Extensions* und den Unterpunkt *Run Code configuration* wählen. Bei *Executor Map* auf *Edit in settings.json* klicken.
- Hier in der "cpp"-Zeile die Variable `$filename` entfernen und vor dem zweiten `&&` die Wildcard `*.cpp` einfügen, so dass die Zeile hinterher so aussieht:
  ```
  "cpp": "cd $dir && g++ -Wall -Wextra -std=c++17 -pedantic
  -o $fileNameWithoutExt *.cpp && $dir$fileNameWithoutExt",
  ```

Beachten Sie, dass Sie nach dieser Änderung **jedes entwickelte Programm in einem eigenen Ordner ablegen müssen**, da immer *alle* `.cpp`-Dateien übersetzt und gelinkt werden. Sie können mit Rechtsklick auf den Workspace → *New Folder* einen neuen Ordner anlegen.

EINSCHRÄNKUNG: Bei diesen Aufgaben sollen alle Methoden in einer separaten Quelltextdatei (`.cpp`) und nicht innerhalb der Header-Datei (`.h`) implementiert werden. Generell darf weder in der Deklaration noch in der Implementierung eine `main`-Funktion existieren.

a) Definieren Sie eine Klasse `Elektrofahrzeug` mit den nötigen *privaten* Attributen, um die Eigenschaften in der folgenden Tabelle zu modellieren.

Eigenschaft	Einheit	Wertebereich	Standardwert
Maximale Energie	kWh	reellwertig, größer 0	80
Durchschnittsverbrauch	kWh/100 km	reellwertig, größer 0	12,5
Ladung	kWh	reellwertig, 0 bis Maximale Energie	50
Kilometerstand	km	reellwertig, nicht negativ	30 000
Höchstgeschwindigkeit	km/h	ganzzahlig, größer 0	160

_____ (2)

b) Implementieren Sie einen Konstruktor für die Klasse `Elektrofahrzeug`, der ein *durchschnittliches* Elektrofahrzeug mit den Standardwerten aus der obigen Tabelle erzeugt. Verwenden Sie für das Setzen der Werte die Initialisier-Liste oder setzen Sie die Werte direkt bei der Deklaration der Variablen.

_____ (1)

c) Fügen Sie der Klasse eine Methode `ausgabe` hinzu, welche alle Eigenschaften mit den zugehörigen Attributwerten auf der Konsole ausgibt.

_____ (1)

d) Testen Sie den Konstruktor aus b) und die Ausgabe aus c) in der `main`-Funktion indem Sie ein Elektrofahrzeug mit den Standardwerten erzeugen und die Methode `ausgabe` aufrufen. Erstellen Sie hierfür in dem gleichen Projekt eine neue Source-Datei, beispielsweise `blatt07.cpp`, mit der `main`-Funktion. Inkludieren Sie zu Beginn dieser Datei mittels `#include` die Header-Datei der Elektrofahrzeug-Klasse.

_____ (1)

e) Fügen Sie der Klasse für jedes Attribut eine Methode zum Abfragen des Wertes hinzu.

_____ (2)

f) Implementieren Sie für den Durchschnittsverbrauch und die Höchstgeschwindigkeit je eine Methode zum Setzen des Wertes.
Stellen Sie sicher, dass den Attributen keine ungültigen Werte zugewiesen werden (die Werte müssen > 0 sein). Falls eine Methode zum Setzen mit einem ungültigen Wert aufgerufen wird, soll eine Fehlerausgabe darauf aufmerksam machen und das entsprechende Attribut soll auf denselben Wert wie beim durchschnittlichen Elektrofahrzeug gesetzt werden.

_____ (2)

g) Implementieren Sie für die Ladung und die maximale Energie je eine Methode zum Setzen des Wertes. Für die maximale Energie max und die Ladung ℓ müssen folgende Eigenschaften gelten: $0 \leq \ell, 0 < max$ und $\ell \leq max$. Falls eine Methode zum Setzen mit einem ungültigen Wert aufgerufen wird, so dass $0 > \ell$ bzw. $0 \geq max$, soll eine Fehlerausgabe darauf aufmerksam machen und das entsprechende Attribut soll auf denselben Wert wie beim durchschnittlichen Elektrofahrzeug gesetzt werden.
Zusätzlich muss jeweils getestet werden ob $max < \ell$ gilt und in dem Fall wird $\ell := max$ gesetzt.

_____ (2)

h) Testen Sie alle Setzen-Methoden in der `main`-Funktion, indem Sie jeweils einen ungültigen und einen gültigen Wert übergeben. Testen Sie auch den Fall, in dem die maximale Energie einen kleineren Wert zugewiesen bekommt als die Ladung.

_____ (1)

Aufgabe 2: Bonus: Elektrofahrzeuge (+11 Punkte)

a) Implementieren Sie einen weiteren Konstruktor für die Klasse `Elektrofahrzeug`, durch den alle privaten Attribute direkt gesetzt werden können. Wird ein ungültiger Wert für ein Attribut übergeben, soll analog zu den Methoden zum Setzen verfahren werden.
Tipp: Um Tipparbeit zu sparen, können Sie die entsprechenden Setze-Methoden aufrufen.

_____ (2)

b) Fügen Sie der Klasse eine Methode `prozent` hinzu, welche den prozentualen Ladestatus zurückgibt.

_____ (1)

c) Fügen Sie der Klasse eine Methode `aufladen` hinzu, welche ein Fahrzeug voll auflädt. Die Methode soll die benötigte Ladung (reellwertig, in kWh) zurückgeben.

_____ (1)

d) Fügen Sie der Klasse eine Methode `maximaleReichweite` hinzu, die berechnet und ausgibt, wie weit mit der derzeitigen Ladung und dem Durchschnittsverbrauch noch maximal gefahren werden kann.

_____ (1)

e) Implementieren Sie eine private, statische Methode `verbrauchFuerStrecke`. Diese soll eine Strecke in km und einen Verbrauch in kWh/100km als Parameter erhalten und zurückgeben, wieviel kWh für diese Parameter verbraucht wird.

_____ (1)

f) Fügen Sie der Klasse eine bool'sche Funktion `fahren` hinzu, welche ein Fahrzeug eine gewisse Strecke mit dem Durchschnittsverbrauch fahren lassen soll. Die gewünschte Distanz soll als Parameter an die Funktion übergeben werden. Prüfen Sie zu Beginn, ob das Fahrzeug die Distanz mit dem aktuellen Ladestand schaffen kann, indem Sie die Methode `maximaleReichweite` aufrufen. Wenn die Distanz nicht erreichbar ist, soll das Auto nicht fahren und `false` zurückgegeben werden. Ansonsten soll `true` zurückgegeben und alle Attribute sollen entsprechend angepasst werden.
Tipp: Die Methode `verbrauchFuerStrecke` ist hilfreich.

_____ (2)

g) Erstellen Sie in einem Testprogramm drei `Elektrofahrzeug`-Objekte in einem Array, die wie in der folgenden Tabelle initialisiert werden sollen:

Eigenschaft	durchschnittliches Elektrofahrzeug	Sportwagen	E-Scooter
Maximale Energie	Standardwert	100	10
Durchschnittsverbrauch	Standardwert	20	5
Ladung	Standardwert	30	9
Kilometerstand	Standardwert	15 000	200
Höchstgeschwindigkeit	Standardwert	300	30

Verwenden Sie dabei zur Initialisierung jeden implementierten Konstruktor mindestens ein Mal. Die Objekte sollen nicht in das Array kopiert, sondern direkt im Array erzeugt werden.
Testen Sie nun Ihre Implementierung wie folgt. Geben Sie zu Beginn und jeweils nach den Test-Operationen 3. und 4. die Daten aller Fahrzeuge aus. Verwenden Sie für den gesamten Test die zuvor angelegten `Elektrofahrzeug`-Instanzen in dem Array. Nutzen Sie dabei aus, dass die Fahrzeuge in einem Array stehen, um nicht unnötigen Programmcode zu schreiben bzw. duplizieren.

1. Geben Sie für die Fahrzeuge die prozentuale Ladung aus.
2. Geben Sie für die Fahrzeuge die maximale Reichweite aus.
3. Lassen Sie alle Fahrzeuge 180 km fahren.
4. Laden Sie die Fahrzeuge voll auf.

_____ (3)