Name:

Matrikelnummer:

Bearbeitungszeit: 120 min.

Aufgabe 1. Gegeben ist die Klasse `Fahrzeug`, die über den Konstruktor

```
public Fahrzeug(String typ)
```

verfügt, der ein Fahrzeug des angegebenen Typs erzeugt.

Weiters gibt es in `Fahrzeug` folgende Methoden:

```
public double getKmStand()
```

gibt den aktuellen Kilometerstand des Fahrzeugs zurück,

```
public double getTankinhalt()
```

gibt den aktuellen Tankinhalt des Fahrzeugs (in Litern) zurück,

```
public void tanken()
```

betankt das Fahrzeug.

Weiters gibt es Methoden, um das Fahrzeug fahren zu lassen. Diese verändern Kilometerstand und Tankinhalt des Fahrzeugs, werden für die weitere Aufgabenstellung aber nicht benötigt.

Schreiben Sie eine Unterklasse `FahrzeugMitVerbrauchsangabe` von `Fahrzeug` mit dem Konstruktor

```
public FahrzeugMitVerbrauchsangabe(String typ)
```

und der zusätzlichen Methode

```
public double getVerbrauch(),
```

die den durchschnittlichen Treibstoffverbrauch pro 100 Kilometer (in Litern) des Fahrzeugs seit seiner Erzeugung zurückgibt.

Aufgabe 2. Gegeben sind die Klassen `Produkt` und `Palettentest`. Die Klasse `Palettentest` verfügt über einen parameterlosen Konstruktor und die Methode

```
public boolean istPackbar(List<Produkt> produktliste),
```

die genau dann `true` zurück gibt, wenn alle Produkte in der Produktliste auf eine Palette gepackt werden können.

Schreiben Sie die Klasse `Packstation` mit einem parameterlosen Konstruktor und der Methode

```
public int getAnzahlPaletten(List<Produkt> alleProdukte),
```

die die kleinste Anzahl von Paletten zurück gibt, die ausreicht, um alle Produkte in der Liste auf Paletten zu packen. (Sie können davon ausgehen, dass alle Produkte auf Paletten gepackt werden können.)

Aufgabe 3. Schreiben Sie eine Klasse `Knoten`, die Knoten in einem gerichteten Graphen darstellt. Der parameterlose Konstruktor von `Knoten` erzeugt einen Knoten, von dem aus kein anderer Knoten erreichbar ist. Die Methode

```
public List<Knoten> getNachfolger()
```

liefert eine Liste jener Knoten zurück, die vom aktuellen Knoten aus direkt erreichbar sind. Die Methode

```
public boolean addNachfolger(Knoten nachfolger)
```

fügt dem aktuellen Knoten einen weiteren direkten Nachfolger hinzu, allerdings nur dann, wenn dadurch kein Kreis entsteht. Ein Kreis ist eine Folge von direkten Nachfolgern $K_1 \to K_2 \to \cdots \to K_n \to K_1$, sodass der letzte Nachfolger gleich dem ersten Knoten der Folge ist. Bei erfolgreichem Hinzufügen eines Nachfolgers liefert die Methode `true` zurück, ansonsten `false`.

Hinweis: Ein Knoten K_2 kann nicht als direkter Nachfolger von K_1 hinzugefügt werden, wenn von K_2 aus über (mehrere) Nachfolger der Knoten K_1 erreicht werden kann.

Beispiel: Haben die Knoten U, V, W, X, Y, Z die in der Tabelle bzw. in der Abbildung angegebenen direkten Nachfolger, dann kann den Knoten U, V, W, X oder Y der Knoten U nicht als direkter Nachfolger hinzugefügt werden, da dadurch ein Kreis entstehen würde. Sehr wohl kann aber dem Knoten Z der Knoten U als direkter Nachfolger hinzugefügt werden.

Knoten	U	V	W	X	Y	Z
Nachfolger	V	W, Y	X	$-$	X	Y