Informationen: Die Beispiele sind bis zur Übung am 17.10. vorzubereiten.

Beispiel 2.1:
Ergänzen Sie die Klasse *Zeitpunkt* aus Beispiel 1.7 um die Methode

```
boolean istVor(Zeitpunkt andererZeitpunkt),
```

die genau dann *true* zurückgibt, wenn der Zeitpunkt, für den die Methode aufgerufen wird, strikt vor *andererZeitpunkt* liegt.

Beispiel 2.2:
Ergänzen Sie die Klasse *Termin* aus Beispiel 1.8 um eine Methode

```
public boolean istVor(Termin andrerTermin),
```

die genau dann *true* zurückgibt, wenn der Termin, für den die Methode aufgerufen wird, vor *andrerTermin* liegt. Verwenden Sie dazu die in Beispiel 2.6 geschriebene Methode.

Beispiel 2.3:
Gegeben sei eine Klasse *Sicherheitstuer* mit einer Methode

```
public boolean oeffne(int code),
```

mit der die Türe geöffnet werden kann, vorausgesetzt der richtige vierstellige Zahlencode wird eingegeben (d.h. als *int* übergeben). In diesem Fall wird die Türe geöffnet und die Methode gibt *true* zurück, andernfalls bleibt die Tür geschlossen und die Methode gibt *false* zurück.

Schreiben Sie eine Klasse *Einbrechen* mit einer statischen Methode

```
public static int oeffneTuer(Sicherheitstuer dieTuer),
```

die die Türe *dieTuer* durch Ausprobieren aller möglichen Codes öffnet und den korrekten Code zurückgibt.

Beispiel 2.4:
Gegeben sei eine Klasse *Zuckerl*, die über die Methoden

```
public String getFarbe()
public String getGeschmack()
```

verfügt, die die Farbe bzw. den Geschmack des Zuckerls als String zurückgeben. Schreiben Sie eine Klasse *Zuckerldose* mit Konstruktor

```
public Zuckerldose(ArrayList<Zuckerl> dieZuckerl),
```

der eine Zuckerldose mit den in der ArrayList *dieZuckerl* vorhandenen Zuckerln anlegt. Implementieren Sie in der Klasse *Zuckerldose* weiters eine Methode

```
public Zuckerl getZuckerl(String farbe, String geschmack),
```

die ein (beliebiges) Zuckerl der Farbe *farbe* und mit Geschmack *geschmack* aus der Zuckerldose entfernt und zurückgibt. Falls es kein entsprechendes Zuckerl gibt, soll die Methode *null* zurückgeben.

Beispiel 2.5:
Schreiben Sie eine Methode

```
boolean sindNachbarnIn(int a, int b, ArrayList<Integer> liste),
```

die genau dann *true* zurückgibt, wenn *a* und *b* in *liste* als Nachbarn vorkommen.

Beispiele:
```
sindNachbarnIn(1,2,{3,2,1,4}) → true
sindNachbarnIn(7,2,{3,2,1,4}) → false
sindNachbarnIn(3,1,{3,2,1,4}) → false
```

Beispiel 2.6:
Gegeben seien die Klassen *Test* und *Reaktor*. Die Klasse *Reaktor* verfügt über eine Methode

```
public boolean fuehreTestDurch(Test einTest),
```

die am Reaktor den Test *einTest* durchführt und *true* zurückgibt, wenn der Reaktor den Test besteht, ansonsten *false*. Mit einer weiteren Methode

```
public void abschalten()
```

kann der Reaktor abgeschaltet werden.

Schreiben Sie eine Klasse *Kernkraftwerk* mit Konstruktor

```
public Kernkraftwerk(ArrayList<Reaktor> dieReaktoren),
```

der ein Kernkraftwerk mit den entsprechenden Reaktoren anlegt.

Implementieren Sie in die Klasse außerdem eine Methode

```
public boolean fuehreTestDurch(Test test),
```

die den Test *test* an allen Reaktoren des Kraftwerks durchführt. Falls ein Reaktor den Test nicht besteht, soll er abgeschaltet werden. Die Methode soll *true* zurückgeben, wenn alle Reaktoren den Test bestanden haben, ansonsten *false*.

Beispiel 2.7:
Schreiben Sie in einer Klasse Ihrer Wahl eine Methode

```
static ArrayList<String> getReversed(ArrayList<String> list),
```

die eine ArrayList zurückgibt, die die Elemente in *list* in umgekehrter Reihenfolge enthält.

Beispiel 2.8:
Versuchen Sie, einzelne nichtstatische Methoden in den Beispielen dieses Übungsblattes statisch zu machen sowie umgekehrt statische Methoden nichtstatisch zu machen. Funktioniert das? Warum (nicht)? Wie unterscheidet sich nun der Methodenaufruf in BlueJ? Warum?

Beispiel 2.9:
Gegeben sei die Klasse *Bestellung* mit der Methode

```
int getPrioritaet(),
```

die die Priorität der Bestellung zurückgibt.

Schreiben Sie eine Klasse *Bestellsystem* mit parameterlosem Konstruktor, der ein Bestellsystem ohne Bestellungen erstellt.

Weiters soll die Klasse *Bestellsystem* über folgende Methoden verfügen:

```
ArrayList<Bestellung> getBestellungen()
```

gibt die Bestellungen absteigend nach Priorität sortiert zurück. (Für Bestellungen mit gleicher Priorität kann die Reihenfolge beliebig sein.)

```
void add(Bestellung b)
```

fügt die Bestellung *b* ins System ein.

```
void erledigeNaechste()
```

löscht die (bzw. eine) Bestellung mit der höchsten Priorität aus dem System.

Hinweis: Am einfachsten ist es, jede Bestellung geordnet ins System einzufügen. Dann müssen die Bestellungen nicht sortiert werden.