Informationen: Die Beispiele sind bis 31.10. um 8 Uhr abzugeben bzw. bis zur Übung vorzubereiten.

Beispiel 4.1:
Schreiben Sie eine Klasse *CountedClass*, die einen parameterlosen Konstruktor und eine Methode

```
static int getNumberOfInstances()
```

enthält, die die Anzahl der bisher erzeugten Objekte dieser Klasse zurückgibt.

Beispiel 4.2: Schreiben Sie eine Klasse *Person* mit Konstruktor

```
Person(String name, Person partner),
```

der eine Person mit entsprechendem Namen und Partner anlegt. Beachten Sie, dass die angelegte Person als Partner in der Person *partner* vermerkt wird. Schreiben Sie entsprechende *get*-Methoden, die Name bzw. Partner der Person zurückgeben.

Beispiel 4.3:
Schreiben Sie in einer Klasse Ihrer Wahl eine Methode

```
static int wuerfelnBisDoppelSechs(),
```

die solange würfelt, d.h. zufällig eine Zahl von 1 bis 6 erzeugt, bis zweimal hintereinander 6 gewürfelt wurde. Die Methode soll die Anzahl der nötigen Versuche zurückgeben.

Beispiel 4.4: ***[RandomNumberTest.java]***
Implementieren Sie eine Klasse *RandomNumberTest* mit den Methoden

- `int[] getRandomNumbers(int min, int max, int size),`
  die ein Array der Größe *size* mit Zufallszahlen >=*min* und <=*max* zurückgibt,
- `int getDice(),`
  die zufällige Werte von 1 bis einschließlich 6 zurückgibt,
- `String getAnswer(),`
  die zufällig eine der Zeichenketten "ja", "nein" oder "vielleicht" zurückgibt.

Alle möglichen Ergebnisse sollen jeweils gleich wahrscheinlich sein.

Beispiel 4.5: ***[Urne.java]***
Schreiben Sie eine Klasse *Urne*, deren Objekte Urnen darstellen, aus denen zufällig rote und blaue Bälle gezogen werden können. Es gibt einen Konstruktor

```
Urne(int anzahlRot, int anzahlBlau),
```

der eine Urne erzeugt, die zu Beginn *anzahlRot* viele rote und *anzahlBlau* viele blaue Bälle enthält.
Die Methode

```
String nextBall()
```

gibt einen roten oder einen blauen Ball zurück, dargestellt durch die Strings "rot" bzw. "blau". Dieser Ball wird zufällig aus jenen Bällen ausgewählt, die sich noch in der Urne befinden. Der ausgewählte Ball soll aus der Urne entfernt werden. Falls sich in der Urne keine Bälle mehr befinden, soll die Methode den Wert *null* zurückgeben.

Beispiel 4.6:
Schreiben Sie in einer Klasse Ihrer Wahl eine statische Methode

```
static ArrayList<Integer> combineLists(
           ArrayList<Integer> list1, ArrayList<Integer> list2),
```

die eine ArrayList zurückgibt, in der beginnend mit dem ersten Element von *list1* zunächst abwechselnd je ein Element von *list1* und *list2* steht. Am Ende der Liste stehen dann die Elemente der längeren der beiden Listen. Für *list1={1,2,3}* und *list2={9,8,7,6,5}* soll das Ergebnis etwa *{1,9,2,8,3,7,6,5}* sein.

Beispiel 4.7:
Gegeben seien die Klassen *Produkt* und *Anlage*. Die Klasse *Produkt* verfügt über eine Methode

```
public boolean hatPrioritaet(),
```

die genau dann *true* zurückgibt, wenn das Produkt Priorität hat.

Die Klasse *Anlage* verfügt über eine Methode

```
public int gibBearbeitungszeit(Produkt p),
```

die die Bearbeitungszeit des Produktes *p* auf der Anlage zurückgibt.

Schreiben Sie in einer Klasse Ihrer Wahl eine Methode

```
ArrayList<Produkt> waehleProdukte(
      ArrayList<Produkt> dieProdukte, int zeit, Anlage a),
```

die Produkte aus *dieProdukte* zur Bearbeitung auf Anlage *a* auswählt und in einer ArrayList zurückgibt, die innerhalb von *zeit* Zeiteinheiten bearbeitet werden können. Dabei sollen zunächst möglichst viele Produkte mit Priorität gewählt werden, anschließend sollen für die gegebenenfalls noch verbleibende Zeit möglichst viele Produkte ohne Priorität gewählt werden.

Beispiel 4.8:
Gegeben seien zwei Klassen *Lieferant* und *Artikel*. Die Klasse *Lieferant* verfügt über folgende Methoden:

```
public void bestelle(Artikel derArtikel)
```

bestellt den Artikel *derArtikel* beim Lieferanten bestellt.

```
public boolean istLieferbar(Artikel derArtikel)
```

gibt *true* zurück, wenn der Lieferant *derArtikel* liefern kann, ansonsten *false*.

Schreiben Sie eine Klasse *ProduktFertigung* mit Konstruktor

```
public ProduktFertigung(String typ,
                        ArrayList<Artikel> dieArtikel),
```

der eine Produktfertigung für das Produkt *typ* anlegt, für dessen Fertigung die in *dieArtikel* angeführten Artikel benötigt werden. Implementieren Sie in die Klasse *ProduktFertigung* außerdem eine Methode

```
public boolean bestelle(ArrayList<Lieferant> dieLieferanten),
```

die alle für die Fertigung nötigen Artikel bei einem Lieferanten aus *dieLieferanten* bestellt, der alle Artikel liefern kann. In diesem Fall soll die Methode *true* zurückgeben. Kann kein Lieferant alle Artikel liefern, so soll nichts bestellt und *false* zurückgegeben werden.