**Informationen:** Die Beispiele sind bis 5.12. 2019, 8 Uhr abzugeben bzw. zur Übung vorzubereiten.

**Beispiel 9.1:**
Gegeben sei eine Klasse *Spielkarte*, die das Interface *Comparable<Spielkarte>* implementiert. Schreiben Sie eine Klasse *KartenspielZuZweit* mit Konstruktor

```
KartenspielZuZweit(TreeSet<Spielkarte> hand1,
                           TreeSet<Spielkarte> hand2),
```

der den beiden Spielern ihre Karten zuteilt und festlegt, dass der erste Spieler ausspielt (d.h. beginnt). Sie können davon ausgehen, dass beide Spieler gleich viele Karten zugeteilt bekommen.

Schreiben Sie weiters eine Methode

```
boolean stich(),
```

die einen Stich gemäß den folgenden Regeln simuliert: Der ausspielende Spieler spielt zunächst seine höchste Karte aus. Falls der andere Spieler die ausgespielte Karte stechen kann (d.h. eine höhere Karte hat), so soll er die ausgespielte Karte mit der niedrigstmöglichen Karte stechen. Falls der andere Spieler keine höhere Karte als die ausgespielte hat, soll er seine niedrigste Karte zugeben. Die Methode soll *true* zurückgeben, wenn der ausspielende Spieler den Stich macht, ansonsten *false*. Außerdem soll die Methode festlegen, dass der Spieler, der den Stich macht, als nächstes ausspielt.

**Beispiel 9.2:** ***[Median.java]***
Schreiben Sie eine Klasse *Median* mit einer Methode

```
int getMittleresElement(int[] a),
```

die ein Element *x* des Arrays *a* zurückgibt, sodass zumindest die Hälfte der Elemente in *a* kleiner gleich *x* ist, und zumindest die Hälfte der Elemente in *a* größer gleich *x* ist.

**Beispiel 9.3:**
Gegeben sei eine Klasse *Dominostein*, die über die Methode

```
public int[] getPunkte()
```

verfügt, die die beiden Punkteanzahlen des Dominosteins in einem Array der Größe 2 zurückgibt.

Schreiben Sie eine Klasse *Dominospiel* mit einer Methode

```
public boolean istKorrekt(ArrayList<Dominostein> dieSteine),
```

die genau dann *true* zurückgibt, wenn *dieSteine* aus richtig aneinandergelegten Dominosteinen besteht, d.h. für jeden Stein ist eine der beiden Punkteanzahlen identisch mit einer Punkteanzahl des vorhergehenden Steins, die andere Punkteanzahl ist identisch mit einer Punkteanzahl des nachfolgenden Steins. (Sie können davon ausgehen, dass die Dominosteine in *dieSteine* paarweise verschieden sind, wobei ein Stein *{j,k}* derselbe ist wie *{k,j}*. )

*Beispiel:* Die Listen mit den Dominosteinen *{{4,3},{4,6},{1,6}}* und *{{3,4},{3,2},{4,2},{4,4}}* sind richtig aneinandergelegt, nicht aber die Liste *{{3,4},{3,2},{3,1}}*.

**Beispiel 9.4:** Schreiben Sie die Klasse *Termin* aus Beispiel 6.1 wie folgt um:

- Es soll einen Konstruktor

  ```
  Termin(int jahr, int monat, int tag, String text)
  ```

  geben, der für das angegebene Datum einen Termin mit dem angegebenen Text erzeugt.

- Die Methode *toString()* aus *Object* soll so überschrieben werden, dass sie Jahr, Monat, Tag und Text des Termins in einem String zurückgibt.

- Implementieren Sie das Interface *Comparable* in *Termin* so, dass es in der weiteren Aufgabenstellung nützlich ist.

Schreiben Sie weiters eine Klasse *Terminverwaltung* mit einem parameterlosen Konstruktor, der eine leere Terminverwaltung ohne Termine erstellt. Weiters soll es folgende Methoden geben:

```
void add(Termin einTermin)
```

fügt einen Termin zur Terminverwaltung hinzu. Wenn es in der Terminverwaltung schon einen Termin mit dem gleichen Datum gibt, wird dieser Termin durch den neuen Termin überschrieben.

```
Termin getNaechstenTermin()
```

gibt den frühesten Termin in der Terminverwaltung zurück und löscht diesen aus der Terminverwaltung. Falls die Terminverwaltung keine Termine enthält, gibt die Methode *null* zurück. Verwalten Sie zur Lösung der Aufgabe die Termin in einem *TreeSet*.

**Beispiel 9.5:** Schreiben Sie eine Methode

```
public static int getMaxLengthAscSeq(int[] liste),
```

die die Länge der längsten (streng) aufsteigenden Folge von aufeinanderfolgenden Werten in *liste* zurückgibt.

*Beispiele:*
```
getMaxLengthAscSeq({1,2,1,2,1,2,1} = 2
getMaxLengthAscSeq({1,5,3,4,8,9,4,3,5}) = 4
getMaxLengthAscSeq({9,8,7,6,5,4,3,2,1}) = 1
getMaxLengthAscSeq({0,0,1,1,2,2}) = 2
```

Überlegen Sie sich eine gute Hilfsmethode, um Mehrfachschleifen zu vermeiden.

**Beispiel 9.6:** Schreiben Sie eine Klasse *WordGenerator* mit einer Methode

```
static void generateWordsOfLengthN(char[] alphabet, int n),
```

die alle Wörter der Länge *n* bestehend aus Zeichen in *alphabet* generiert und per *System.out.println* ausgibt.

*Hinweis:* Legen Sie eine konkrete Reihenfolge fest, in der die Wörter generiert werden sollen. Anschließend überlegen Sie sich, wie ausgehend vom zuletzt generierten Wort das nächste erzeugt werden kann.