Anmerkungen

- **Es wird noch eine Bonus-Serie geben, bei der Sie noch einen Zusatzpunkt holen können.**
- Die Aufgaben basieren hauptsächlich auf rekursion (Kapitel 11). Aufgabe 7-3 kann man schon vorher beginnen.
- **Abgabe:** Die Abgabe erfolgt online auf ILIAS. Der Quellcode zu den Aufgaben 7-1, 7-2 und 7-3 soll als `*.java` Datei abgegeben werden. Andere Formate werden nicht akzeptiert.
- Quellcode-Dateien, welche wir nicht kompilieren können, werden nicht akzeptiert.
- Arbeit in Zweiergruppen: Geben Sie jeweils nur ein Exemplar der Lösung pro Gruppe ab. Geben Sie in der Quellcode-Datei die **Namen und Matrikelnummern** beider Gruppenmitglieder in den ersten beiden Zeilen als Kommentar an.
- Einzelarbeit: Geben Sie ebenfalls Ihren Namen und Ihre Matrikelnummer in der ersten Zeile der Quellcode-Datei als Kommentar an.

Aufgabe 7-1

1. Schreiben Sie eine **rekursive** Methode `static long fib(int i)`, die die i-te Zahl der Folge

   $$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, \ldots$$

   berechnet. Z.B. soll der Aufruf `fib(7)` den Wert 13 liefern. Die erste Zahl der Folge ist 0 (entsprechend `fib(0)`, die zweite 1 (`fib(1)`, danach ist jede Zahl die Summe ihrer beiden Vorgänger, z.B. `fib(8)` $=$ `fib(7)` $+$ `fib(6)`. Allgemein: `fib(n)` $=$ `fib(n-1)` $+$ `fib(n-2)` (für $n \geq 2$)

   Schreiben Sie dazu eine passende `main`-Methode, die die ersten 50 Zahlen der Folge am Bildschirm ausgibt. Was stellen Sie beim Ausführen des Programms fest?

2. Schreiben Sie eine **rekursive** Methode

   ```
   public static long factorial(int n),
   ```

   welche die Fakultätsfunktion $n! = n \cdot (n-1) \cdot (n-2) \cdot ... \cdot 1$ berechnet. Schreiben Sie dazu eine `main`-Methode, so dass man das Programm wie folgt von der Konsole aus starten kann:

   ```
   $ java Factorial 10
   3628800
   ```

   Woran liegt es, dass dieses Programm effizienter ist als dasjenige aus Teilaufgabe 1? Überlegen Sie sich insbesondere, wieviele rekursive Methodenaufrufe für eine Berechnung nötig sind.

   Hinweis: Die Funktion $n!$ wächst sehr schnell! Der Wert $21!$ übersteigt den Wertebereich von `long` bereits. Java ermöglicht das Rechnen mit sehr grossen natürlichen Zahlen durch die Klasse `BigInteger`.

3. Schreiben Sie ein **nicht-rekursives** Programm mit dem gleichen Output wie das Programm aus Teilaufgabe 1.

4. Gegeben sei folgende Methode:

   ```
   static void iterative(){
      for(int i=0; i<=30; i+=3) System.out.println(i);
   }
   ```

   Schreiben Sie dazu eine äquivalente **rekursive** Methode (ohne Verwendung von Schleifen).

   Überlegen Sie sich, wie *beliebige* Schleifen durch Rekursion eliminiert werden können.

Aufgabe 7-2

Implementieren Sie den rekursiven Mergesort-Algorithmus aus der Vorlesung. Schreiben Sie dazu eine Klasse `MergeSort` mit einer Methode

```
public static void sort(Comparable[] array),
```

die einen Array von `Comparable`-Objekten sortiert. Schreiben Sie dazu eine rekursive Hilfsmethode `mergeSort(...)` und eine (nicht-rekursive) Hilfsmethode `merge(...)`.

Versuchen Sie die vorgegebenen Klassen `SortTest` und `Rectangle` zu verstehen und testen Sie Ihren Algorithmus mit der Klasse `SortTest`.

Hinweis: Ignorieren Sie allfällige Kompilerwarnungen wie "`MergeSort.java uses unchecked or unsafe operations`".

Aufgabe 7-3

Implementieren Sie eine kleine Java-Klasse, welche sich mit dem Mergesort-Algorithmus aus Aufgabe 7-2 sortieren lässt. Das bedeutet, dass Sie das Comparable Interface implementieren müssen. Zum Beispiel können Sie eine Klasse `Person.java` mit den Attributen `age`, `gender` und `height` erstellen. Die Instanzen werden dann nach `height` sortiert. **Wir erwarten von jeder Gruppe eine individuelle Klasse! Das Beispiel soll also nicht benutzt werden.** Schreiben Sie zudem eine passende `toString`-Methode.

Ergänzen Sie die Klasse `SortTest` mit einem Test ihrer Klasse ähnlich wie bei `Rectangle`.