# Übung 1 Fibonacci-Zahlen

1. Die Fibonacci-Reihe ist eine unendliche Folge von ganzen positiven Zahlen (siehe  
   <http://de.wikipedia.org/wiki/Fibonacci-Folge>) sie beginnt mit f0 = 0 & f1 = 1.

Jede weitere Zahl ergiebt sich aus der Summe ihrer beiden Vorgänger, also: f2 = f1 + f0.

1. Erzeugt nun eine Klassemit einer Methode fibonacci() in welcher ihr für eine gegebene ganze positive Zahl die zugehörige Fibonaccizahl ausgiebt (z.B. fibonacci(1) gibt zurück: 1).
2. Versucht die Methode fibonacci() sowohl iterativ als auch rekursiv(siehe Beispiel) umzusetzen.

# Übung 2 Züge

1. Erstellt fuer alle Klassen jeweils ein Klassendiagramm, bevor ihr sie implementiert.
2. Schreiben Sie Klassen, die Eisenbahnzüge repräsentieren. Ein Eisenbahnzug besteht aus einer Lokomotive und einer beliebigen Anzahl Wagen, möglicherweise auch überhaupt keinen.
3. Lokomotiven und Wagen haben die folgenden Eigenschaften(alle ganzzahlig), welche im Konstruktor gesetzt werden sollen:
   1. Lokomotive:  
      • Länge (in Meter)  
      • Typ (irgendeine Zahl)
   2. Wagen:  
      • Länge (in Meter)  
      • Passagierkapazität (Anzahl Personen)
4. Definieren Sie die Klassen Lokomotive und Wagen, jeweils mit den angegebenen Eigenschaften und sinnvollen Methoden. Die oben genannten Eigenschaften sind unveränderlich.
5. Das interessante Problem ist das Zusammenstellen eines Zuges aus den Einzelteilen. Der erste Wagen hängt direkt an der Lokomotive. Geben Sie der Klasse Lokomotive deshalb eine  
   Objektvariable ersterWagen vom Typ Wagen, dazu eine Getter- und eine Setter-Methode. An jedem Wagen hängt der jeweils nächste Wagen oder gar nichts beim letzten Wagen. Definieren Sie in der Klasse Wagen eine Objektvariable naechsterWagen des gleichen Typs Wagen, wieder mit Gettern und Settern. Diese Objektvariable speichert ein anderes Objekt derselben Klasse oder null beim letzten Wagen.
6. Definieren Sie schließlich eine Klasse Zug, die den ganzen Zug repräsentiert. Ein Zug-Objekt kennt "seine" Lokomotive (Instanzvariable \_lokomotive), aber nicht die Wagen. Diese können aber, einer nach dem anderen, auf dem Weg über die Lokomotive erreicht werden.
7. Die Klasse Zug bietet die folgenden Methoden (ergänzen Sie sinnvolle Parameter und  
   Rückgabewerte):  
   • Konstruktor: Der Zug-Konstruktor erwartet eine Lokomotive und baut einen ziemlich kurzen Zug, der nur aus der Lokomotive, noch ohne Wagen besteht.  
   • wagenHinzufuegen: Hängt für diesen Zug einen gegebenen Wagen an das Ende an. • erstenWagenEntfernen: Hängt den ersten Wagen aus diesem Zug aus und liefert den  
   ausgehängten Wagen als Ergebnis zurück. Die restlichen Wagen rücken nach vorne. Falls es  
   keinen Wagen gibt, ist das Ergebnis null.  
   • zugAnhaengen: Akzeptiert als Parameter einen anderen Zug und hängt alle Wagen des  
   anderen Zuges in der gleichen Reihenfolge an diesen Zug an. Im anderen Zug bleibt nur die  
   Lokomotive zurück. Nutzen Sie für diese Methode geschickt die vorher definierten Methoden.  
   • getWagenAnzahl: Liefert die Anzahl der Wagen in diesem Zug (ohne Lokomotive).  
   • getKapazitaet: Liefert die gesamte Passagierkapazität dieses Zuges, das heißt die  
   Summe der Passagierkapazitäten aller Wagen.  
   • getLaenge: Liefert die Gesamtlänge dieses Zuges, d. h. die Summe der Länge der  
   Lokomotive und aller Wagen.  
   • info: Gibt eine Beschreibung dieses Zuges mit allen Bestandteilen (Typ der Lok, Anzahl  
   Wagen, Gesamtlänge, gesamte Passagierkapazität sowie für jeden Wagen Seriennummer,  
   Wagenlänge und Passagierkapazität ) auf der Konsole aus.

Beispiel rekursiver Methodenaufruf:

public int zweierpotenz(int exponent) {

return (exponent == 0) ? 1 : zweierpotenz(exponent-1)\*2;

}