# Algorithmen und Datenstrukturen

## Einführung in Algorithmen

### Aufgabe 1 – Kleinstes gemeinsames Vielfaches (kgV) – [1 Punkt]

Wenn man den ggT bestimmt hat, kann man das kleinste gemeinsame Vielfache (kgV) unter Anwendung folgender Formel bestimmen.

$$\mathrm{ggT}(m,n) \cdot \mathrm{kgV}(m,n) = |m \cdot n|$$

Aufgabe: Implementieren Sie die Methode **`int kgv(int a, int b)`**, die den kgV zweier int-Werte bestimmt. Die Testfälle sollen korrekt durchlaufen werden können. Die Methode soll zusätzlich mit beliebigen Werten von Ihrem CommandExecutor aufgerufen werden können. Die Klasse muss im "ch.zhaw.ads" Package liegen.

### Hinweis:

```
public String execute(String s) {
      String[] numbers = s.split("[ ,]+");
      int a = Integer.parseInt(numbers[0]);
      int b = Integer.parseInt(numbers[1]);
      return Integer.toString(kgv(a,b));
}
```

oder

```
public String execute(String s) {
      Scanner scanner = new Scanner(new ByteArrayInputStream(s.getBytes()));
      int a = scanner.nextInt();
      int b = scanner.nextInt();
      return Integer.toString(kgv(a,b));
}
```

### Gerüst:

```
package ch.zhaw.ads;

public class KgvServer implements CommandExecutor {
      public int kgv(int a, int b)...
}
```

### Abgabe

Praktikum: ADS1.1

Filename: KgvServer.java

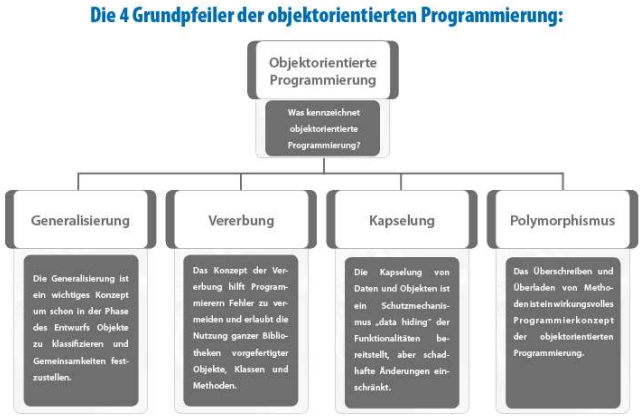
### Aufgabe 2 - Stack [1 Punkte]

Implementieren Sie einen Stack, der beliebige Typen speichern kann (als Objekte). Ihr Stack muss dabei folgende Methoden implementieren bzw. das vorgegebene Interface implementieren.

- push
- pop
- peek
- isEmpty
- isFull

## Aufgabe

Im Stack wird ein sog. ADT implementiert. Überlegen Sie sich, welche "Grundpfeiler" der OOP sich durch einen ADT umsetzen lassen (keine Abgabe).




Implementieren Sie den Stack und überprüfen Sie die korrekte Implementation Ihrer Klasse mit dem *StackTest*.

## Gerüst:

```
public class ListStack implements Stack {
      ...
}
```

## Abgabe

Praktikum: ADS1.2

Filename: ListStack.java

## Aufgabe 3 - Klammertester [4 Punkte]

Arithmetische Ausdrücke und Quelltexte (z. B. Java, LaTeX, etc.) können oft sehr viele ineinandergeschachtelte Klammern enthalten. In dieser Aufgabe soll geprüft werden, ob alle Klammern korrekt geschlossen werden. Dabei sollen die folgenden Klammern erkannt werden: ( ), [ ], { }.

Die Klammerung ist korrekt, wenn jede öffnende Klammer durch eine Klammer vom selben Typ geschlossen wird.

Beispiel für korrekte Klammerung:

[ (3 +3)· 35 >+3]· {3 +2}

Beispiel für falsche Klammerung:

[({3 +3)· 35} +3]· {3 +2}

(Erste { wird durch ) geschlossen.)

a) Implementieren Sie die Methode `boolean checkBrackets (String arg)`, die als Eingabe einen Text erhält und ausgibt, ob die Klammerung korrekt ist.

Verwenden Sie Ihren Stack aus der vorhergehenden Aufgabe, um die geöffneten Klammern zu verwalten. Führen Sie die UnitTests aus.

Hinweis: implementieren Sie eine private Methode `char nextBracket()`, die Ihnen die nächste Klammer retourniert und alle nicht interessierenden Zeichen überliest.

b) Zusatzaufgabe (wird später gestellt) überprüft, ob Sie in der ADS Vorlesung noch etwas lernen können.

### Gerüst:

```
public class BracketServer implements CommandExecutor {
    public boolean checkBrackets(String command) {
    ...
}
```

### Abgabe

Praktikum: ADS1.3

Filename: BracketServer.java

## Aufgabe 4 - XML Wellformed Tester [4 Punkte]

Ein XML Dokument wird als *wohlgeformt* (wellformed) bezeichnet, wenn u.a. alle öffnenden Tags korrekt mit einem korrespondierenden schliessenden abgeschlossen werden. Siehe auch https://en.wikipedia.org/wiki/Well-formed_document

Schreiben Sie ein Programm, das dies überprüft. Implementieren Sie dafür eine Methode `boolean checkWellformed(String arg)`. Als Test nehmen Sie einfach beliebige XML Dateien, die sich entweder auf Ihrer Maschine befinden oder aus dem Internet. In der ExBox können Sie dann diese Datei mittels File→Open Ihrem Server als String übergeben.

Hinweise:

- Implementieren Sie eine private Methode `String getNextToken()`, die Ihnen das nächste Token retourniert und alle nicht interessierenden Teile überliest. Um die Tokens zu erkennen, können Sie String Operationen oder - falls Sie diese schon kennen - Regex Operationen verwenden.
- Die Tokens, die gleich wieder geschlossen werden wie z.B. <b/> können - als einfache Lösung - auch überlesen werden.
- XML Elemente können auch Attribute enthalten: <a attr = "foo" >. Diese müssen ebenfalls überlesen werden.

### Gerüst:

```
public class WellformedXmlServer implements CommandExecutor {
    public boolean checkWellformed (String command) {
    ...
}
```

### Abgabe

Praktikum: ADS1.4

Filename: WellformedXmlServer.java