Praktikum 01: Einführung in Algorithmen

Einleitung Tests und Abgabe

Sie finden im Java-Code-Rahmen der Aufgabenstellung jeweils einen passenden Test je Aufgabe. Für die erste Aufgabe heisst der Test z.B. ADS1_1_test, für die zweite Aufgabe ADS1_2_test, usw. Diese Tests korrespondieren mit den Tests, welche in Moodle beim Hochladen der Praktika durchgeführt werden. Nach erfolgreicher Abgabe auf Moodle erhalten Sie einen Punkt je erfolgreich abgegebener Aufgabe.

Aufgabe 1: Kleinstes gemeinsames Vielfaches (kgV)

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

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

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.

Verwenden Sie den zur Verfügung gestellten Test, um Ihre Methode zu überprüfen. Geben Sie danach Ihre Lösung auf Moodle ab.

Hinweise:

```
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:

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

Aufgabe 2: Stack

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: Überlegen Sie sich, welche Reaktion auf Stackunterlauf sinnvoll ist.

Welche jeweiligen Vor- und Nachteile haben die beiden Varianten und was muss man bei der Null-Wert Lösung zusätzlich beachten?

Gerüst:

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

Aufgabe 3: Klammertester

Arithmetische Ausdrücke und Quelltexte (z. B. Java, LaTeX, etc.) können oft sehr viele ineinander geschachtelte 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.

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. Hinweis: implementieren Sie eine Methode private String getNextBracket, die Ihnen die nächste Klammer retourniert und alle nicht interessierenden Zeichen überliest.

Hinweise:

- Beispiel für korrekte Klammerung: [(3 +3)· 35 >+3]· {3 +2}
- Beispiel für falsche Klammerung, erste { wird durch) geschlossen: [({3 +3)· 35} +3]· {3 +2}

Gerüst:

```
public class BracketServer implements CommandExecutor {
   ...
}
```

Aufgabe 4: XML Wellformed Tester

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 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 in den Strings 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.

Gerüst:

```
public class WellformedXmlServer implements CommandExecutor {
   ...
}
```