Praktikum „SEP: Java-Programmierung“ WS 2018/19

Trie: Typische Fehler

Thomas Lemberger und Martin Spießl
Basierend auf Folien von Matthias Dangl und Karlheinz Friedberger

Stil: Fallunterscheidung

- Tief geschachtelte `if-else`-Kaskade:

```
if (command.startsWith("a")) {
      ...
} else {
    if (command.startsWith("A")) {
        ...
    } else {
        if (command.startsWith("c")) {
            ...
        } else {
            ...
        }
    }
}
```

Stil: Fallunterscheidung

- Besser:

```
switch (command.toLowerCase().charAt(0)) {
   case 'a':
     ...
     break;
   case 'c':
     ...
     break;
   ...
   default:
     ...
     break;
}
```

Stil: Fallunerscheidung

- Auch ok:

```
switch (command) {
  // Fall: new
  case "new":
  case "ne":
  case "n":
    this.helpNew(nameParam, punkteParam);
    break;
  // Fall: add
  case "add":
  case "ad":
  case "a":
    this.helpAdd(nameParam, punkteParam);
  break;
```

Stil: Magic Numbers

- Überall magische Zahl 26:

```
Node[] children = new Node[26];
...
for (int i = 0; i < 26; ++i) {
  ...
}
```

- Besser Konstanten verwenden:

```
private static final int MAX_CHILDREN
    = 'z' - 'a' + 1;
Node[] children = new Node[MAX_CHILDREN];
...
for (int i = 0; i < children.length; ++i) {
  ...
}
```

- Weitere Verbesserung: Iteration mit `for-each`

Beenden der JVM

```
private static void execute(BufferedReader stdin) throws IOException {
  boolean quit = true;
  while (quit) {
    System.out.print("trie> ");
    String input = stdin.readLine();
    if (input == null) {
      System.exit(0);
    }
    ...
```

Beenden der JVM

```
private static void execute(BufferedReader stdin) throws IOException {
  boolean quit = true;
  while (quit) {
    System.out.print("trie> ");
    String input = stdin.readLine();
    if (input == null) {
      System.exit(0);
    }
    ...
```

- `System.exit(0)` beendet ganze JVM
- Führt zu Fehlern bei Wiederverwendung
- Richtiger Mechanismus über `quit` bereits vorhanden, aber hier unbenutzt

Stil: Zeilenbreite und Tabs(1)

- Erlaubte Zeilenbreite: Maximal 80 Zeichen
- Ab und zu mal 82 Zeichen ist noch OK
- Praktomat-Rekord bei Trie:
  `Shell.java: Maximale Zeilenbreite: 486 Zeichen`
  `(letztes Semester: 536)`
- Achtung: CheckStyle zählt Tabulator als 8 Zeichen
- **Tabs sind aber ohnehin verboten**

Stil: Zeilenbreite und Tabs(2)

Analyse der längsten Zeile dieses Jahr:

```
find . -name "*.java" -exec cat  {} \; | \
awk '{print length, $0}'|sort -nr|head -1
```

```
486   private static final String HELPTEXT =
"Willkommen! Folgende Befehle sind zulässig:\nnew:
Erstellt einen brandneuen trie.\nadd <name> <punkte>:
Trägt für <name> die <punkte> in den Trie ein.
\nchange <name> <punkte>: Ändert den Punktestand von
 <name> in <punkte>.\ndelete <name>:Entfernt <name>
aus dem Trie.\npoints <name>: Zeigt den Punktestand
von <name>.\ntrie: Gibt den aktuellen Trie aus.
\nhelp: Gibt diesen Hilfetext aus.\nquit:
Dieses Programm beenden."; //TODO: Helptext
```

Stil: Ungarische Notation (Systems Hungarian)

- Java ist stark und statisch typisiert
- Typinformation im Bezeichner (sogenannte Systems-Hungarian Notation) ist daher
  - im besten Fall redundant
  - im schlimmsten Fall falsch und irreführend
  - immer schlechter Stil
- Maximal verwirrendes Beispiel aus den Abgaben:

```
int charInt;
```

Performance: String-Konkatenation

- Schlecht: Ständig umkopieren

```
String result = "";
...
for (Node child : children) {
  if (child != null) {
    result += child;
  }
}
```

- Besser: (mutable) StringBuilder statt (immutable) String:

```
StringBuilder result = new StringBuilder();
...
for (Node child : children) {
  if (child != null) {
    result.append(child);
  }
}
```

- Noch besser: StringBuilder als Parameter für rekursive Hilfsmethode

Kontrollfluss: Exceptions

- Schlecht: Ablaufsteuerung durch Exceptions

```
StringBuilder result = new StringBuilder();
...
for (Node child : children) {
 try {
   result.append(child.toString());
 } catch (NullPointerException e) {
   // child does not exist
 }
}
```

- Stattdessen: **vorher** auf `null` prüfen (siehe vorherige Folie)

Verletzung der Zuständigkeiten

- Ausgabe direkt im Modell:

```
class Trie {
  ...
  public void add(String key, int value) {
    ...
    if (...) {
      System.out.println("Error! "
          + key + " already exists!");
    }
  }
  ...
}
```

Verletzung der Zuständigkeiten

- Ausgabe direkt im Modell:

```
class Trie {
  ...
  public void add(String key, int value) {
    ...
    if (...) {
      System.out.println("Error! "
          + key + " already exists!");
    }
  }
  ...
}
```

  - Massive Verletzung der Zuständigkeit
  - Geheimnis der `Shell` verletzt
  - Model dadurch weder wiederverwendbar noch austauschbar

Javadoc (1)

```
/**
* im Konstruktor wird der Wurzelknoten erzeugt
*/
public Trie(){
  root = new Node();
}
```

Javadoc (1)

```
/**
* im Konstruktor wird der Wurzelknoten erzeugt
*/
public Trie(){
  root = new Node();
}
```

- Beschreibung ist für den Nutzer der Klasse nicht hilfreich
- Verrät Klassen-Geheimnis!

Javadoc (1)

```
/**
* im Konstruktor wird der Wurzelknoten erzeugt
*/
public Trie(){
  root = new Node();
}
```

- Beschreibung ist für den Nutzer der Klasse nicht hilfreich
- Verrät Klassen-Geheimnis!

```
* @throws "Error! input incorrect!", falls mehr als 3 Wörter...
```

Javadoc (1)

```
/**
* im Konstruktor wird der Wurzelknoten erzeugt
*/
public Trie(){
  root = new Node();
}
```

- Beschreibung ist für den Nutzer der Klasse nicht hilfreich
- Verrät Klassen-Geheimnis!

```
* @throws "Error! input incorrect!", falls mehr als 3 Wörter...
```

- Fehlender Exception-Typ! (`IOException`)

Javadoc (2)

```
/** come on, it's main()!.
*
* @param args I have no idea what is this
* @throws IOException Thomas asked us to do this
*/
public static void main(String[] args) throws IOException
```

Javadoc (2)

```
/** come on, it's main()!.
*
* @param args I have no idea what is this
* @throws IOException Thomas asked us to do this
*/
public static void main(String[] args) throws IOException
```

- Jeder kennt das Prinzip einer `main`-Methode, aber keiner weiß, was diese spezielle `main`-Methode macht!
- Keinen Kommentar in `/**`-Zeile setzen!
- Dokumentation hilft, das eigene Programm zu verstehen und Lücken im Verständnis aufzudecken

Typsicherheit: „Stringly-Typed“

- Java ist eigentlich stark typisiert . . .
- . . . man kann aber PHP in jeder Sprache schreiben . . .

```
public String add(String key, String value) {
  ...
  if (...) {
    return "fail";
  }
  return "success";
}
```

  - Bei uns völlig unzulässig, siehe Veranstaltungstitel

Warum einfach...

Einfach	Kompliziert
`int x = 5;`	`int x = new int[] { 5 }[0];`

Warum einfach...

Einfach	Kompliziert
`int x = 5;`	`int x = new int[] { 5 }[0];`
`char a = 'a';`	`char a = "a".charAt(0);`

Warum einfach...

Einfach	Kompliziert
`int x = 5;`	`int x = new int[] { 5 }[0];`
`char a = 'a';`	`char a = "a".charAt(0);`
`Character.getType(name.charAt(0))`	`(name.charAt(0)-97)<0||`
`!= Character.LOWERCASE_LETTER`	`(name.charAt(0)-97)>26`

Beispiel mit diversen Problemen (1)

```
public void depthFirst(Node curNode) {
  for (int i = 0; i <= ALPHABET_SIZE - 1; i++) {
    if (curNode.childExists(i)) {
      System.out.print("(");
      break;
    }
  }
  for (int i = 0; i <= ALPHABET_SIZE - 1; i++) {
    if (curNode.childExists(i)) {
      System.out.print(curNode.getChild(i));
      depthFirst(curNode.getChild(i));
    }
  }
  for (int i = 0; i <= ALPHABET_SIZE - 1; i++) {
    if (curNode.childExists(i)) {
      System.out.print(")");
      break;
    }
  }
}
```

Probleme:

- Hilfsmethode öffentlich
- Umständlicher Vergleich
- Besser `for-each` statt Indexiteration
- Falsche Klasse (Trie statt Node)
- Ausgabe im Modell

Beispiel mit diversen Problemen (2)

```
public Node(int value)
  throws IllegalArgumentException {
  try {
    if (value < 0) {
      this.value = value;
    }
  } catch (IllegalArgumentException e){
      System.out.println("Error! Value is of wrong form");
    }
}
```

Beispiel mit diversen Problemen (2)

```
public Node(int value)
  throws IllegalArgumentException {
  try {
    if (value < 0) {
      this.value = value;
    }
  } catch (IllegalArgumentException e){
      System.out.println("Error! Value is of wrong form");
    }
}
```

Probleme:

- Mehrere Layout-Fehler (u. a. auch Tabs)
- `throws` in Signatur für Unchecked-Exceptions überflüssig
- `catch` von Unchecked-Exceptions verboten
- Kurios: Die Exception tritt hier gar nicht auf!
- Ausgabe im Modell

Beispiel mit diversen Problemen (3)

```
/**
 * Search for the last node representing the end of a string
 * and change the points.
 *
 * @param key String
 * @param newPoints int
 * @return boolean. True if the search and replacement is successfull
 */

public boolean change(String key, int newPoints) {
```

Probleme:

- Parameterdokumentation nutzlos
- Unnötige Leerzeile zwischen JavaDoc und Signatur
- Besser {`@code true`} statt `True` oder `TRUE`
- Rechtschreibfehler

Beispiel mit diversen Problemen (4)

```
private static final String[] instructions = {
  "new", ": create a new TRIE, the current one will be abandoned. \n"..
private static String prefixMakeUp(String input) {
  String toComp = reverse(input);
  for (int i = 0; 2 * i < instructions.length; i++) {
    for (int j = 0; j < instructions[2 * i].length(); j++) {
      if (toComp.equals(reverse(instructions[2 * i]).substring(j))) {
        return instructions[2 * i];
  }}}
  return "UNKNOWN";
```

Beispiel mit diversen Problemen (4)

```
private static final String[] instructions = {
  "new", ": create a new TRIE, the current one will be abandoned. \n"..
private static String prefixMakeUp(String input) {
  String toComp = reverse(input);
  for (int i = 0; 2 * i < instructions.length; i++) {
    for (int j = 0; j < instructions[2 * i].length(); j++) {
      if (toComp.equals(reverse(instructions[2 * i]).substring(j))) {
        return instructions[2 * i];
  }}}
  return "UNKNOWN";
```

- "Stringly"-Typed (Alternative: `Optional<String>` / `enum`)
- `instructions` enthält key/value-Paare für Hilfe
  $\rightarrow$ besser `Map<K,V>` bzw. Enums benutzen
- Berechnung kompliziert und skaliert schlecht
- `substring(0,j)` statt invertieren
- `reverse` nicht selbst implementieren:
  `new StringBuilder(toComp).reverse().toString()`

Problem: Unnötiger Code bzw. StringBuilder(1)

```
public String toString() {
      String output = root.toString();
      return output;
}
```

```
String out = "";
for (Node child : children)
    if (child != null) out += child.toString();
return out;
```

Problem: Unnötiger Code bzw. StringBuilder(2)

```
public String toString(String input){
  String output = input;
  output = output + "(";
  for(int i = 0; i<child.length; i++){
    if(child[i] != null){
      output =  output + child[i].letter;
      if(child[i].points!=-1){
        output = output + "[" + child[i].points + "]";
        for(int j = 0; j<child.length; j++){
          if(child[i].child[j]!=null){
            output = child[i].toString(output);
        } }
      } else {
        output = child[i].toString(output);
  } } }
  output = output + ")";
  return output;
}
```

Problem: Unnötiger Code bzw. StringBuilder(2)

```
public String toString(String input){
  String output = input;
  output = output + "(";
  for(int i = 0; i<child.length; i++){
    if(child[i] != null){
      output =  output + child[i].letter;
      if(child[i].points!=-1){
        output = output + "[" + child[i].points + "]";
        for(int j = 0; j<child.length; j++){
          if(child[i].child[j]!=null){
            output = child[i].toString(output);
        } }
      } else {
        output = child[i].toString(output);
  } } }
  output = output + ")";
  return output;
}
```

Probleme:

- kein `StringBuilder`
- Verschachtelung bis 5 Level
- Magischer Wert -1
- Tabs statt Spaces
- `toString` überladen!

Problem: Input lesen und darauf reagieren

```
public void runInterface() throws IOException {
  BufferedReader reader =
      new BufferedReader(new InputStreamReader(System.in));
  String line = null;
  String output = null;
  do {
    System.out.println("trie> ");
    line = reader.readLine();
    output = executeCommand(line);
    if (Objects.nonNull(output) && !Objects.equals(output, "quit")) {
      System.out.println(output);
    }
  } while (!Objects.equals(output, "quit"));
}
```