

Fragen, Antworten und Kommentare zur aktuellen Vorlesung

Frage: Wir haben Schwierigkeiten ein konkretes Hausarbeitsthema zu finden.

(s. auch FA06) Die Aufgabenstellung ist bewusst offen formuliert, damit sie ein Thema finden, dass sie interessiert. Die Idee ist dann, dieses Framework oder die Bibliothek auszuprobieren und dann eine kleine Aufgabenstellung zu definieren, die sie dann mit OOAD-Mitteln von der Analyse bis zur Implementierung durchziehen. Alternativ entwickeln Sie ein großes Projekt und setzen dann nach der Use Case-Modellierung nur das erste Inkrement um. Generell steht die komplette Durchführung aller Entwicklungsschritte im Mittelpunkt.

Wem das noch zu unkonkret ist, für den ist der Vorschlag sich eine Oberflächentechnologie in Java anzusehen, dann auf einzelne Steuerungselemente wie Knöpfe und Tabellen zu konzentrieren und dann dazu eine Beispielaufgabe zu definieren. Beispiele für Technologien sind: Swing (alt, etabliert, fester Java-Bestandteil), JavaFX (genauer OpenJFX, war mal Teil von Java, muss jetzt selbst ergänzt werden; ist aber in der Java-Version in der KleukerSEU enthalten), Google Web Toolkit (GWT, Web-Applikationen mit Java), Spring (Web-Applikationen, da einen sehr kleinen Einstieg). Der Tipp ist, wenn Sie kein interessantes Framework kennen, etwas Zeit in Google investieren, bei ein zwei Technologien schauen, ob man das Konzept versteht und Hello-World-artiges zum Laufen bekommt und sich dann eine kleine Aufgabenstellung auszudenken.

Eine einfachere Aufgabenstellung ist die Neuimplementierung z. B. von Teilen des Collection-Frameworks von Java und ein Vergleich mit existierenden Implementierungen. Bei solcher einer Aufgabenstellung ist es eher unwahrscheinlich ein sehr gutes aber auch ein sehr schlechtes Ergebnis zu erreichen.

Frage: Es gibt keine Präsentation?

Antwort: Richtig, anders als beim Projektbericht gibt es bei der Hausarbeit keine Präsentation, so dass mehr Zeit zur Erstellung des Textes zur Verfügung steht. Am Ende existiert ein Textdokument, dass der Prof von vorne nach hinten liest und bewertet. Der Arbeitsaufwand ist identisch für alle Module mit Leistungspunkten.

Frage: Wieviele Aufgabenblätter gibt es?

Antwort: 12 bewertete (letzte Blatt hat die Nummer 11, da es 6 und 6b gab).

Frage: Müssen wir Sequenzdiagramme in der Hausarbeit nutzen und sollen wir sie generieren lassen.

Meine Standardantwort bei „müssen“ ist meist: Sie müssen das nicht machen und ich muss ihnen auch keine sehr gute Note geben. Konkret hängt es hier später von der von Ihnen gewählten Aufgabenstellung ab, ob Sequenzdiagramme und wie viele sinnvoll sind. Wenn der Fokus z. B. auf der Kommunikation von GUI-Elementen liegt, gibt es oft ein zentrales Prinzip, so dass ein Diagramm reicht. Da im Sequenzdiagramm nur alle relevanten Klassen sichtbar sein sollen, kann man das Diagramm generieren oder auch von Hand zeichnen. Ein Beispiel ist die Klasse Messages in der Aufgabe 21, die für das Verständnis der MVC-Idee sicherlich nicht relevant ist und weggelassen wird.

Ähnliches gilt für die EinUndAusgabe-Klasse, wobei die eigentliche Ausgabe wieder sehr wichtig ist, aber auch einfach mit einem Pfeil von der View-Klasse zu Extern mit der Ausgabe auf dem Pfeil dargestellt werden könnte.

Beachten Sie, dass meine Vorgaben nur das enthalten können, bei deren vollständiger Abarbeitung ein Ergebnis herauskommt, dass die durchschnittlichen Erwartungen erfüllt. Das entspricht der Basisnote 3.0.

Das ist nebenbei der Grund, warum ich nicht noch detaillierter in die Aufgabenstellung einsteige. Würde ich das machen, wäre es immer schwerer eine (sehr) gute Note zu erreichen. Sehr gute Leistungen entstehen nie durch Pflichterfüllung.

Frage: In Java gibt es die Klassen Observer und Observable, die sind aber deprecated, sollen wir sie nutzen? Hat das was mit dem Flow-API zu tun?

Antwort: Den genauen Grund für „deprecated“ kenne ich nicht. Da es aber üblich ist den Observer-Observable-Ansatz in die eigenen Klassen und Interfaces zu integrieren, spielen diese Klassen in der Umsetzung in der Praxis kaum eine Rolle. Also integrieren und implementieren Sie den Ansatz selbst.

Das Flow-API hat eine gewisse Verwandtschaft, dient aber einen eigen Architekturansatz, wie er z. B. mit dem Framework RxJava umgesetzt werden kann. Wenn Sie sich für interessante Architekturansätze interessieren, ist es sehr empfehlenswert, aber definitiv kein „Muss“.

Generell gibt es immer wieder neue Ansätze, wie Java-Streams, Flow-API oder das Play-Accu-Framework, die gerne gehypt werden. Jeder dieser Ansätze ist sinnvoll, bietet mehrere Vorteile, ist aber kein ultimativer Game-Changer. Man kann die Vorteile nutzen, muss es aber nicht und wird dadurch auch nicht zu schlechteren oder weniger performanten SW-Lösungen kommen.

Zum Clonen hier noch ein kleines weiteres Beispiel, warum eine einfache rekursive Nutzung von clone() nur unter bestimmten Randbedingungen zum Ergebnis führt.

```

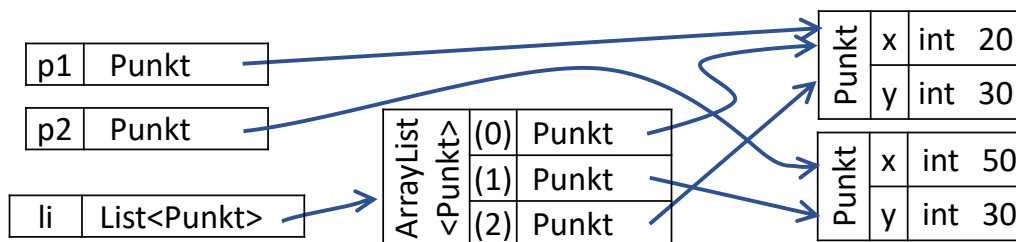
6 public class Beispiel {
7
8     public static List<Punkt> kopieren(List<Punkt> liste){
9         List<Punkt> ergebnis = new ArrayList<>();
10        for(Punkt e:liste) {
11            if (e == null) {
12                ergebnis.add(null);
13            } else {
14                ergebnis.add(e.clone());
15            }
16        }
17        return ergebnis;
18    }
19
20    public static void main(String[] s) {
21        List<Punkt> li = new ArrayList<>();
22        Punkt p1 = new Punkt(20,30);
23        Punkt p2 = new Punkt(50,30);
24        li.add(p1);
25        li.add(p2);
26        li.add(p1);
27        System.out.println("" + (li.get(0) == li.get(2)) );
28        List<Punkt> li2 = kopieren(li);
29        System.out.println("" + (li2.get(0) == li2.get(2)) );
30    }
31 }

```

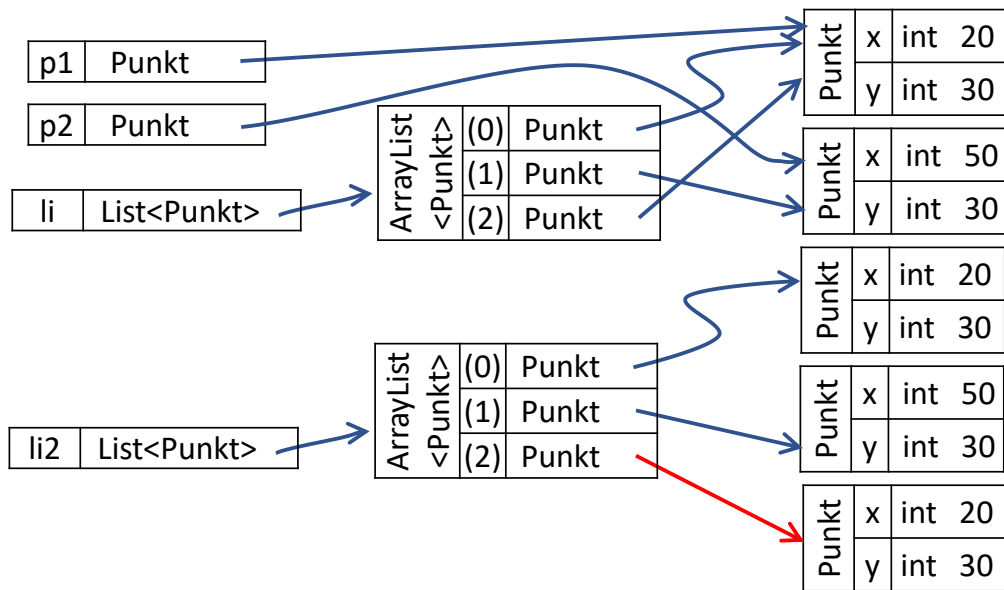
Problems @ Javadoc Console Declaration Coverage

<terminated> Beispiel [Java Application] C:\kleukersSEU\java\bi
true
false

Die Methode kopieren() kopiert einfach jedes existierende Element. In der Liste li befinden sich identische Elemente an den Positionen 0 und 2, wie auch das folgende Objektspeicherdiagramm zeigt.



Beim Erstellen der Kopie stehen dann nur gleiche Objekte an den Positionen 0 und 2.



Als Lösung kann man sich schon geclonte Objekte z. B. in einer zu übergebenden HashMap merken und vor jedem Clonen prüfen, ob das Objekt schon geclonet wurde und ggfls. dieses als Ergebnis liefern. Dies HashMap muss dann bei jedem Clone()-Schritt als Parameter mitgegeben werden. Der XML-Ansatz stellt dies durch Referenzen in XML sicher, wie der folgende Ausschnitt zeigt. Alternativ können hier Binär-Kopien zu schnellen sauberen Clone-Ergebnissen führen.

```
<void property="Liste1">
  <void method="add">
    <object class="entity.Punkt" id="Punkt0">
      <void property="x">
        <int>20</int>
      </void>
      <void property="y">
        <int>30</int>
      </void>
    </object>
  </void>
  <void method="add">
    <object class="entity.Punkt">
      <void property="x">
        <int>50</int>
      </void>
      <void property="y">
        <int>30</int>
      </void>
    </object>
  </void>
  <void method="add">
    <object idref="Punkt0"/>
  </void>
</void>
```

Frage: Wir haben es mit Streams beim Clonen versucht, da ging aber gar nichts.

Achtung, es gibt zumindest zwei Varianten, in denen in Java der Begriff Stream benutzt wird. Die klassische Variante ist auch in den Beispiellösungen zum Clonen enthalten und nutzt Klassen wie

FileInputStream, FileOutputStream oder die 16-Bit-Varianten FileReader und FileWriter. Dies sind alles typische Klassen die eine Dateibearbeitung ermöglichen.

Der Stream-Begriff ist dann in anderer Form bei der Einführung von Lambda-Ausdrücken und der funktionalen Programmierung in Java erneut relevant, z.B.

```
int[] arr= {9,7,3,1};  
Arrays.stream(arr)  
    .forEach(i -> System.out.println(i));
```

Es können u. a. Collections in eine Stream-Verarbeitung gesteckt werden, mit der jedes Objekt des Streams schrittweise in der danach beschriebenen Form verarbeitet wird. Diese Streams können dabei beliebig lang sein und sehr unterschiedliche Quellen haben. Das Thema „Einführung in die reaktive Programmierung in Java anhand eines Beispiels“ ist in diesem Zusammenhang auch für Hausarbeiten wählbar.

Diese kurze Übersicht soll zur Vermutung führen, dass die beiden Stream-Begriffe in der Nutzung direkt auf wesentlich andere Anwendungsszenarien zielen, was ein sinnvoller Basisgedanke ist. In einer abstrakteren Sichtweise stellt sich die Frage, warum dann der gleiche Begriff gewählt wurde. Die Vermutung, dass jede Art von Byte-Stream auch als funktionaler Stream betrachtet werden kann, ist dabei sinnvoll. Es gibt also auch funktionale Möglichkeiten Dateien zu lesen und zu schreiben, was allerdings beim Clone meiner Erfahrung nach nicht hilft (?).

```
public static void main(String[] args) {  
    try(Stream<Path> projekt = Files.list(Path.of("../"))) {  
        projekt.filter(Files::isDirectory)  
            .forEach(System.out::println);  
    } catch (IOException e) {  
        System.out.println("Exception: " + e);  
    }  
}
```

(nachfolgend Beispielantworten der KI, die ich geprüft habe)

skleuker

Warum sind die Klassen Observer und Observable in Java deprecated?

KI

Die Klassen `Observer` und `Observable` in Java sind als veraltet (deprecated) markiert worden, da sie mehrere Designprobleme und Einschränkungen aufweisen:

1. Thread-Sicherheit: Die Implementierung von `Observable` ist nicht thread-sicher, was bedeutet, dass sie in einer Umgebung mit mehreren Threads fehlerhaft arbeiten kann. Entwickler müssen zusätzliche Maßnahmen ergreifen, um sicherzustellen, dass ihre Verwendung in multithreaded Anwendungen korrekt funktioniert.

2. Flexibilität: Das Observer-Pattern in Java hält sich nicht vollständig an die Standard-Entwurfsmusterpraktiken. Zum Beispiel kann ein `Observable` nur eine konkrete Klasse sein, da es keine Schnittstelle gibt. Das schränkt seine Verwendung und Flexibilität ein.

3. Einheitliche Schnittstellen: In modernen Java-Anwendungen ist es üblich, Schnittstellen zu verwenden, um das Observer-Pattern zu implementieren. Java bietet keine `Observable`-Schnittstelle, was zu einer schwerfälligen Implementierung führen kann. Stattdessen gibt es alternative Bibliotheken und Lösungen, wie zum Beispiel das Observer-Pattern über eigene Interfaces zu implementieren, die oft eleganter und flexibler sind.

4. Alternative Ansätze: Es gibt mittlerweile gängigere und robustere Wege, reaktive Programmiermuster zu implementieren, wie zum Beispiel durch die Verwendung von Frameworks wie RxJava oder den in Java 9 eingeführten Flow-API im Rahmen der Reactive Streams.

Aufgrund dieser und anderer Einschränkungen wird Entwicklern empfohlen, auf eigene Implementierungen des Observer-Patterns oder modernere Alternativen zurückzugreifen, um ähnliche Funktionalitäten zu erreichen.

skleuker

Was versteht man unter dem Play-Framework in Java?

KI

Das Play-Framework ist ein Web-Application-Framework, das in Scala und Java entwickelt wurde und das Erstellen von Webanwendungen vereinfacht. Es ist bekannt für seine Skalierbarkeit und die Unterstützung moderner Webstandards. Hier sind einige der Hauptmerkmale und Eigenschaften des Play-Frameworks:

1. Reaktiv und nicht-blockierend: Play ist darauf ausgelegt, reaktive Anwendungen zu unterstützen, die nicht-blockierend und skalierbar sind. Dies macht es ideal für Anwendungen, die hohe Lasten verarbeiten müssen.

2. Concurrency: Play nutzt Akka unter der Haube, um hochkonkurrente und reaktionsschnelle Anwendungen zu ermöglichen.

3. MVC-Architektur: Das Framework folgt dem Model-View-Controller (MVC) Paradigma, was eine klare Trennung zwischen Anwendungslogik, Benutzerschnittstelle und Datenmodellierung fördert.

4. Hot Code Reload: Play bietet die Fähigkeit, Codeänderungen sofort bei der Entwicklung zu sehen, ohne dass ein Neustart des Servers erforderlich ist.

5. Eingebaute Entwicklungswerkzeuge: Es hat leistungsfähige Entwicklungswerkzeuge und unterstützt den Workflows von Continuous Integration und Continuous Deployment.

6. Unterstützung für moderne Webtechnologien: Play bietet Unterstützung für RESTful Webdienste, WebSockets und andere moderne Webtechnologien, die in modernen Anwendungen oft benötigt werden.

7. Einfache Konfiguration: Die Konfiguration von Play-Anwendungen ist im Vergleich zu vielen anderen Java EE Frameworks recht einfach und unkompliziert.

8. Integration mit anderen Technologien: Play integriert sich gut mit anderen Technologien und Tools, wie z.B. Datenbanksystemen, JSON Verarbeitungsbibliotheken und ORMs wie Ebean oder JPA.

Das Play-Framework wird oft von Entwicklern bevorzugt, die Anwendungen mit hohem Durchsatz und Echtzeitanforderungen entwickeln, da es moderne Architekturprinzipien sowie Entwicklerfreundlichkeit durch seine umfangreichen Features und Tools bietet.

skleuker

Ist die Nutzung von Java-Streams schneller als die Nutzung klassischer Schleifen und Alternativen?

KI

Die Nutzung von Java-Streams gegenüber klassischen Schleifen bietet verschiedene Vorteile in Bezug auf Lesbarkeit und Ausdrucksstärke des Codes, jedoch ist die Leistung (Performance) ein differenziertes Thema. Ob Streams schneller sind als klassische Schleifen, hängt von mehreren Faktoren ab:

1. Lesbarkeit und Wartbarkeit: Streams bieten eine funktionale Programmierweise, die häufig zu kürzerem und besser lesbarem Code führt. Dies kann insbesondere bei komplexeren Datenverarbeitungen von Vorteil sein.

2. Performance: In Bezug auf die Ausführungsgeschwindigkeit sind Streams nicht notwendigerweise schneller als klassische Schleifen. Tatsächlich können Streams in einigen Fällen langsamer sein, besonders wenn sie unnötig viele Objekte erzeugen (z.B. durch Boxing und Unboxing von primitiven Typen) oder wenn sie nicht optimal eingesetzt werden.

3. Parallelität: Einer der Vorteile von Streams ist die Möglichkeit, sie einfach zu parallelisieren (d.h. parallele Streams verwenden), was bei großen Datenmengen die Performance steigern kann, da die Verarbeitung über mehrere Threads verteilt wird. Jedoch ist dies nicht immer automatisch schneller, da die Kosten für Thread-Management und Synchronisation eine Rolle spielen.

4. Laufzeitorientierungen: JIT-Compiler können traditionelle Schleifen sehr effizient optimieren. Streams hingegen bieten Potenzial für Laufzeitorientierungen durch die Implementierung in der Streams-API selbst, aber diese Optimierungen hängen von der JVM und dem spezifischen Anwendungsfall ab.

5. Geeignete Anwendungsfälle: Streams sind besonders nützlich, wenn eine deklarative Darstellung der Datenverarbeitung gewünscht ist. Sie sind ideal für Anwendungen, die hauptsächlich mit Sammlungen manipulieren und filtern, transformieren oder aggregieren müssen.

Insgesamt ist es wichtig, sowohl Streams als auch klassische Schleifen auf ihre Eignung für den spezifischen Anwendungsfall zu prüfen. Bei Performance-kritischen Anwendungen ist oft eine Profilierung und Analyse des Codes notwendig, um die beste Wahl zu treffen. In vielen Fällen handelt es sich um einen Trade-off zwischen Klarheit des Codes und maximaler Ausführungsperformance.