Programmierung WS 18
Hausaufgaben - Blatt 6

Julian Giesen (MNR 388487)
Levin Gäher (MNR 395035)
Gruppe 12

HA 2

a)

(1) A(int); explizit − > A(String); explizit− > Object(); implizit
Ausgabe: v1.x: written in A(int)
Begründung: x wird ausgegeben. Da der Konstruktor mit einem int aufgerufen wird, wird x in A(String) auf "written in A(int) gesetzt.
(2)
Reihenfolge: B(int); explizit − > B(String); explizit − > A(String) explizit − > Object(); implizit

Aufruf:System.out.println("v2.x: "+v2.x);
Ausgabe: v2.x: written in B(String)
Begründung: Wegen super() wird die Oberklasse von B aufgerufen. Deswegen wird A(String) aufgerufen.

Aufruf: System.out.println("((B) v2).x: " + ((B) v2).x);
Ausgabe: ((B)v2).x: written in B(int)
Begründung: Das casten von A auf B sorgt dafür, dass das x von B über x von A priorisiert wird.

(3)
Reihenfolge: B(A); explizit − > B(String); explizit − > A(String) − > Object(); implizit

Aufruf: System.out.println("((A)v3).x: "+((A)v3).x);
Ausgabe: ((A)v3).x: written in B(String)
Begründung: v3 wird zu A gecastet. Somit wird die variable x aus A ausgegeben.

Aufruf: System.out.println("v3.x: "+v3.x);
Ausgabe: v3.x: written in B(A) Begründung: v3 ist instanceof B. Somit wird die variable x die in B liegt ausgegeben.
(4)
Reihenfolge: B() explizit − > B(String) explizit − > A(String) explizit − > Object();

Aufruf:System.out.println("((A)v4).x: "+((A)v4).x); Ausgabe: ((A)v4).x: written in B(String)
Begründung: v4 wird zu A gecastet. Somit wird die variable x aus A ausgegeben.

Aufruf: System.out.println("v4.x: "+v4.x);
Ausgabe: v4.x: written in B() Begründung: Da v4 instanceof B ist, wird das x von B ausgegeben.

b)

(1)Signatur: A.f(A)
(2)Signatur: A.f(A)
(3)Signatur: A.f(A)

(4)Signatur: B.f(A)
(5)Signatur: B.f(A)
(6)Signatur: B.f(A)
(7)Signatur: B.f(A)
(8)Signatur: B.f(A)
(9)Signatur: B.f(B)

HA 5

Entry

```
public class Entry {
   private final String name;
   private final Node node;

   public Entry(String Name, Node Node) {
      name = Name;
      node = Node;
   }

   public String getName() {
      return name;
   }

   public Node getNode() {
      return node;
   }

   public File getAsFile() {
      if (node instanceof File)
         return (File) node;
      return null;
   }

   public Directory getAsDirectory() {
      if (node instanceof Directory)
         return (Directory) node;
      return null;
   }

   public Entry createHardlink(String newName) {
      return new Entry(newName, node);
   }
}
```

Node

```
import java.lang.*;

public abstract class Node {

   private long lastModified;

   public Node() {
      touch();
   }

   public long getLastModified() {
      return lastModified;
   }
```

```
   public void touch() {
      lastModified = System.currentTimeMillis();
   }
}
```

Directory

```
public class Directory extends Node {

   private Entry[] children;

   // Constructor

   public static Directory createEmpty() {
      return new Directory();
   }

   public Directory() {
      super();
      children = new Entry[0];
   }

   // General

   public Entry[] getEntries() {
      //return new Entry[children.length];
      return children;
   }

   public boolean containsEntry(String name) {
      return getEntry(name) != null;
   }

   public Entry getEntry(String name) {
      for (int i = 0; i < children.length; i++) {
         if (children[i].getName() == name)
            return children[i];
      }
      return null;
   }

   // Utility

   public void accept(String name, Visitor visitor) {
      visitor.visitDirectory(name, this);
      for (int i = 0; i < children.length; i++) {
         if (children[i].getNode() instanceof File)
            visitor.visitFile(children[i].getName(), (File) children[i].getNode());
         else if (children[i].getNode() instanceof Directory)
            ((Directory) children[i].getNode()).accept(children[i].getName(),
                visitor);
      }
      visitor.visitedDirectory();
   }

   // Creators

   public Entry createDirectory(String name) {
      if (containsEntry(name)) {
         System.out.println("Error: Es existiert bereits einen Eintrag mit dem
             Namen'" + name + "'!");
         return null;
```

```
      }
      Entry dir = new Entry(name, new Directory());
      addEntry(dir);
      return dir;
   }

   public Entry createFile(String name, String content) {
      if (containsEntry(name)) {
         System.out.println("Error: Es existiert bereits einen Eintrag mit dem
             Namen'" + name + "'!");
         return null;
      }
      File f = new File();
      f.writeContent(content);
      Entry file = new Entry(name, f);
      addEntry(file);
      return file;
   }

   public Entry createHardlink(String name, Entry entry) {
      if (containsEntry(name)) {
         System.out.println("Error: Es existiert bereits einen Eintrag mit dem
             Namen'" + name + "'!");
         return null;
      }
      Entry link = new Entry(name, entry.getNode());
      addEntry(link);
      return link;
   }

   // Helpers

   private void addEntry(Entry entry) {
      Entry[] newChildren = new Entry[children.length + 1];
      for (int i = 0; i < children.length; i++)
         newChildren[i] = children[i];
      newChildren[children.length] = entry;
      children = newChildren;
   }
}
```

File

```
public class File extends Node {

   private String content;

   public File() {
      super();
   }

   public String readContent() {
      return content;
   }

   public void writeContent(String Content) {
      content = Content;
      this.touch();
   }

}
```

Visitor

```
public interface Visitor
{
   void visitFile(String name, File file);
   void visitDirectory (String name, Directory directory);
   void visitedDirectory();
}
```

Printer

```
import java.lang.*;

public class Printer implements Visitor {
   public String curPath = "";

   public void visitFile(String name, File file) {
      System.out.println(file.getLastModified() + " " + curPath + name);
      System.out.println("> " + file.readContent());
   }

   public void visitDirectory(String name, Directory directory) {
      curPath = curPath + name + "/";
      System.out.println(directory.getLastModified() + " " + curPath);
   }

   public void visitedDirectory() {
      curPath = curPath.substring(0, curPath.lastIndexOf("/")) + "/";
   }
}
```