TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK

Lehrstuhl für Sprachen und Beschreibungsstrukturen
Praktikum Grundlagen der Programmierung
Prof. Dr. Helmut Seidl

SS 2008
Aufgabenblatt 8
9. Juni 2008

Abgabe: 17.-19. Juni 2008 beim jeweiligen Tutor

Praktikum Grundlagen der Programmierung

Themen: Polymorphie, Datenstrukturen

Aufgabe 39 (Ü) Generische Klassen

Entwickeln Sie eine generische Klasse `Set<E>`. Ein Objekt der Klasse `Set<E>` soll eine Menge von Elementen vom Typ `E` darstellen. Zur Implementierung können Sie das Interface `List<E>` benutzen.

Die Klasse `Set<E>` muss folgende Methoden bereitstellen:

a) eine Objekt-Methode `public boolean isIn(E elem)`, die testet, ob ein Element mit dem selben Inhalt wie `elem` in dem `Set`-Objekt enthalten ist.
b) eine Objekt-Methode `public void add(E elem)`, die ein Element `elem` dem `Set`-Objekt hinzufügt, falls es nicht schon vorhanden ist.
c) eine Objekt-Methode `public void intersect(Set<E> s)` zur Berechnung der Schnittmenge zwischen dem aktuellen Objekt (für das die Methode aufgerufen wurde) und dem angegebenem Argument. Anschließend soll das aktuelle Objekt nur noch die Elemente enthalten, die in beiden Ursprungsmengen enthalten waren.

Verwenden Sie zum Vergleichen der Elemente den Inhaltsvergleich. Nehmen Sie dabei an, dass die Methode `equals` des Objekts `elem` den Wert `true` zurückliefert, wenn sie mit einem Element gleichen Inhalts aufgerufen wird.

Aufgabe 40 (Ü) Binärer Suchbaum

Ein *Binärbaum* ist ein Baum, in dem alle Knoten nicht mehr als 2 Kinder haben. Ein *binärer Suchbaum* (BSB) ist ein Binärbaum, der folgende Ordnungseigenschaft erfüllt:

Für jeden Knoten des Baums gilt, dass sein eigener Wert nicht kleiner ist als die Werte aller Knoten in seinem linken Teilbaum, aber echt kleiner ist als die Werte aller Knoten in seinem rechten Teilbaum. Implementieren Sie eine Klasse, mit der man generische BSBs repräsentieren kann und

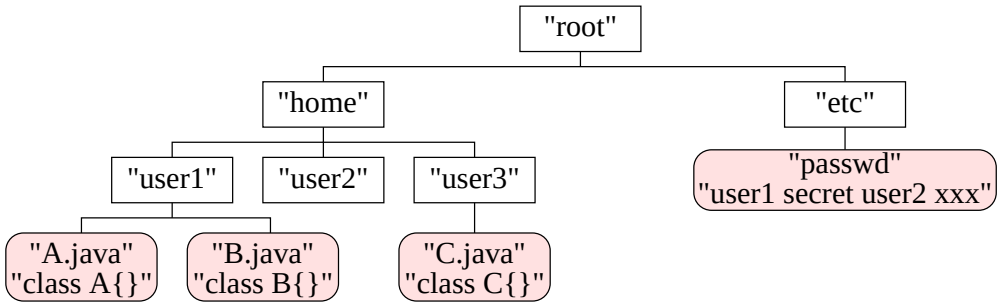
a) die Methoden `BSB getLeft()`, `BSB getRight()`, `T getValue()`, die den rechten bzw. linken Teilbaum und den Wert der Wurzel (`value`) liefern, sowie die Methoden **`void`** `setLeft(BSB b)`, **`void`** `setRight(BSB b)`, **`void`** `setValue(T v)` zum Setzen der entsprechenden Werte,
b) eine `toString()`-Methode zur Darstellung des binären Suchbaums,
c) die Methode **`void`** `insert(T n)`, die - unter Erhalt der Ordnungseigenschaft - einen neuen Knoten mit dem Wert `n` in den BSB einfügt und
d) die Methode **`boolean`** `contains(T n)`, die ausgibt ob die Zahl `n` im BSB enthalten ist.

Testen Sie Ihre Implementierung an Hand eines geeigneten `main`-Programms.

Aufgabe 41 (H) Verzeichnisstrukturen **(16 Punkte)**

Entwickeln Sie ein System von Klassen, mit denen sich Dateien in Verzeichnissen organisieren lassen. Eine Datei besteht aus ihrem Namen vom Typ `String` und ihrem Inhalt, der der Einfachheit halber auch vom Typ `String` ist. Ein Verzeichnis besteht aus seinem Namen vom Typ `String` und aus einer (möglicherweise leeren) Liste von Unterverzeichnissen und Dateien. Zur Implementierung können Sie das Interface `List<E>` benutzen.

Ein Beispiel-Filesystem ist unten abgebildet. Verzeichnisse sind durch Rechtecke und Dateien durch abgerundete graue Rechtecke dargestellt:

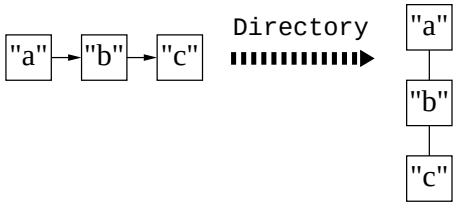


a) Beschreiben Sie Ihre Datenstruktur mit einem geeigneten UML-Klassendiagramm. Führen Sie dazu Klassen `File` und `Directory` ein. Methoden sind nicht verlangt. Alternativ, können Sie auch den Java-Code Ihrer Klassen direkt angeben.

   Implementieren Sie Methoden `public void makeDir(String name)` und `public void makeFile(String name,String inhalt)` der Klasse `Directory`, die ein Verzeichnis bzw. eine Datei mit Inhalt im aktuellen Verzeichnis anlegen. Das aktuelle Verzeichnis ist das `Directory`-Objekt, dessen Methode aufgerufen wurde. Wenn ein Verzeichnis oder eine Datei mit dem selben Namen im aktuellen Verzeichnis existiert, soll nichts gemacht werden.

b) Implementieren Sie eine Methode `public Directory changeDir(List<String> path)` der Klasse `Directory`, die als Argument einen Pfad `path` (als Liste von `String`s) erhält. Wenn `path` zu einem Unterverzeichnis *d* aus dem aktuellen Verzeichnis führt, soll *d* zurückgeliefert werden (z.B. führt `new List<String>("home",new List<String>("user1"))` im "root"-Verzeichnis zum "user1"-Verzeichnis). Sonst soll `null` zurückgeliefert werden.

c) Geben Sie einen Konstruktor `public Directory(List<String> path)` an, der als Argument einen Pfad (als Liste von `String`s) erhält, und ein `Directory` Objekt erzeugt, das ein Verzeichnis darstellt, das nur den angegebenen Pfad enthält.



d) Entwickeln Sie eine Methode `public void mkDirHierarchy(List<String> path)` in der Klasse `Directory`, die als Argument einen Pfad (als Liste von `String`s) erhält. Diese Methode ein Unterverzeichnis im aktuellen Verzeichnis in Ihrem System anlegt, so dass `path` der Pfad vom aktuellen Verzeichnis zum neuen Unterverzeichnis ist. Falls das Unterverzeichnis oder eine Datei mit diesem Namen in dem aktuellen Verzeichnis schon existiert, soll eine Fehlermeldung erzeugt werden.