Programmieren 2 (INF)
Übungsblatt 08

In diesem Übungsblatt geht es primär um das Arbeiten mit verschachtelten Klassen. Der `FileTreeWalker` vermittelt darüber hinaus weitere Einblicke in den Umgang mit Dateien und Verzeichnissen.

Aufgabe 1

Legen Sie auch für dieses Übungsblatt wieder ein neues Paket an. Legen Sie darin eine Kopie der Klasse `MyDeque` aus dem letzten Übungsblatt an. Statten Sie diese Kopie mit einem Iterator aus, der als verschachtelte Klasse realisiert ist. Welche Art von verschachtelter Klasse bietet sich besonders an? Nutzen Sie alle vorhandenen Möglichkeiten, um die Kapselung zu optimieren.

Aufgabe 2

Werfen Sie einen Blick auf das folgende Programm. Wie funktioniert es? Testen Sie es am Rechner aus!

```
import java.io.File;
import java.io.IOException;

/**
 * Prints recursively all files which are contained in
 * the current directory or in sub-directories.
 * @author Reinhard Schiedermeier, Ruediger Lunde
 */
public class FileTreeWalker {

	public static void main(String[] args) throws IOException {
		new FileTreeWalker().walk(".");
	}

	public void walk(String pathname) throws IOException {
		File[] files = new File(pathname).listFiles();
		if (files != null) {
			for (File file : files)
				if (file.isDirectory())
					walk(file.getCanonicalPath());
				else
					process(file);
		}
	}

	protected void process(File file) throws IOException {
		System.out.printf("%-100s%6d%n", file.getCanonicalPath(),
		    file.length());
	}

}
```

Aufgabe 3

Es sollen nun Varianten des `FileTreeWalker`s für verschiedene Zwecke entwickelt werden. Eine Möglichkeit bestände darin, die gegebene Klasse abzuleiten und dann jeweils die `process`-Methode zu überschreiben. Dem Grundsatz *„favor object composition over class inheritance“* (Gang of Four) folgend gehen wir aber einen anderen Weg.

Legen Sie eine Kopie der gegebenen Klasse unter dem Namen `FlexibleFileTreeWalker` an und definieren Sie *darin* ein Interface `FileProcessor` (warum statisch?). Es verlangt von seinen Implementierungen, die Methode `process(File file)` bereit zu stellen. Jedem `FlexibleFileTreeWalker` kann im Konstruktor ein solcher Dateiprozessor übergeben werden. Beim Gang durch den Dateibaum ruft er dann für jede Datei den übergebenen Prozessor auf.

Aufgabe 4

Legen Sie nun die Klasse `LargeFileFinder` an. Ihre `main`-Methode startet eine Instanz des `FlexibleFileTreeWalker`s, die jetzt aber mit einem speziellen Prozessor ausgerüstet wird. Er ordnet die Dateien nach ihrer Größe bevor er sie ausgibt. Der spezielle `FileProcessor` soll als geschachtelte Klasse realisiert werden (welche Art von Klasse?). Beachten Sie, dass er mindestens ein Attribut und eine weitere Methode (neben `process`) benötigt, um seine Arbeit zu tun. Diese zusätzliche Methode wird dann in der `main`-Methode aufgerufen und stößt die Ausgabe an. Tipp: Das Sortieren lässt sich am einfachsten mit einem `Comparator` bewerkstelligen, der als anonyme Klasse realisiert ist.

Aufgabe 5

Realisieren Sie schließlich ein weiteres Werkzeug `DuplicateFileFinder` zum Auffinden von Duplikaten. Angezeigt werden sollen solche Dateien, die bezüglich Name (`file.getName()`) und Größe übereinstimmen, aber unter unterschiedlichen Pfaden abgelegt sind. Machen Sie dabei Gebrauch von `HashMap`s.