Prof. Dr. Ralf Hinze
Markus Heinrich, M.Sc.

TU Kaiserslautern
Fachbereich Informatik
AG Programmiersprachen

Übungsblatt 2: Programmierpraktikum 2021

Ausgabe: 04. Mai 2021
Abgabe: 11. Mai 2021, 15 Uhr

Aufgabe 1 Dateisystem (36 Punkte)

In dieser Aufgabe soll ein einfaches in-memory Dateisystem modelliert werden. Die Inhalte des Dateisystems existieren also nur im Hauptspeicher d. h. nur während der Ausführung des Programms.

Jedes Element des Dateisystems muss die Schnittstelle `FSObject` implementieren.

```
public interface FSObject {
    /**
     * @return the name of the file system object
     */
    String getName();

    /**
     * @param name the new name of the file system object
     */
    void setName(String name);

    /**
     * @return reference to the parent file system object
     */
    FSObject getParent();

    /**
     * Updates the parent reference
     *
     * @param parent the new parent
     */
    void setParent(FSObject parent);

    /**
     * @return the full path of the object in the file system
     */
    String getPath();

    /**
     * Remove the file system object
     *
     * @throws NotEmpty If the file system object is a non-empty directory
     */
    void remove() throws NotEmpty;
}
```

Wie Sie an den Getter- und Setter-Methoden erkennen können, besitzt ein Dateisystemobjekt einen Namen und ein Elternelement. Die Methode `String getPath()` soll den Pfad des Objekts im Dateisystem als String zurückgeben. Die Methode `remove()` löscht das Element aus dem Dateisystem.

Für unser einfaches Dateisystem brauchen wir Dateisystemobjekte für Dateien (`File` Klasse) und Verzeichnisse (`Directory` Klasse). Ein Verzeichnis enthält eine Liste von Referenzen auf die darin enthaltenen Elemente. Damit lässt sich eine Verzeichnisstruktur, wie in Abbildung 1 dargestellt, modellieren.

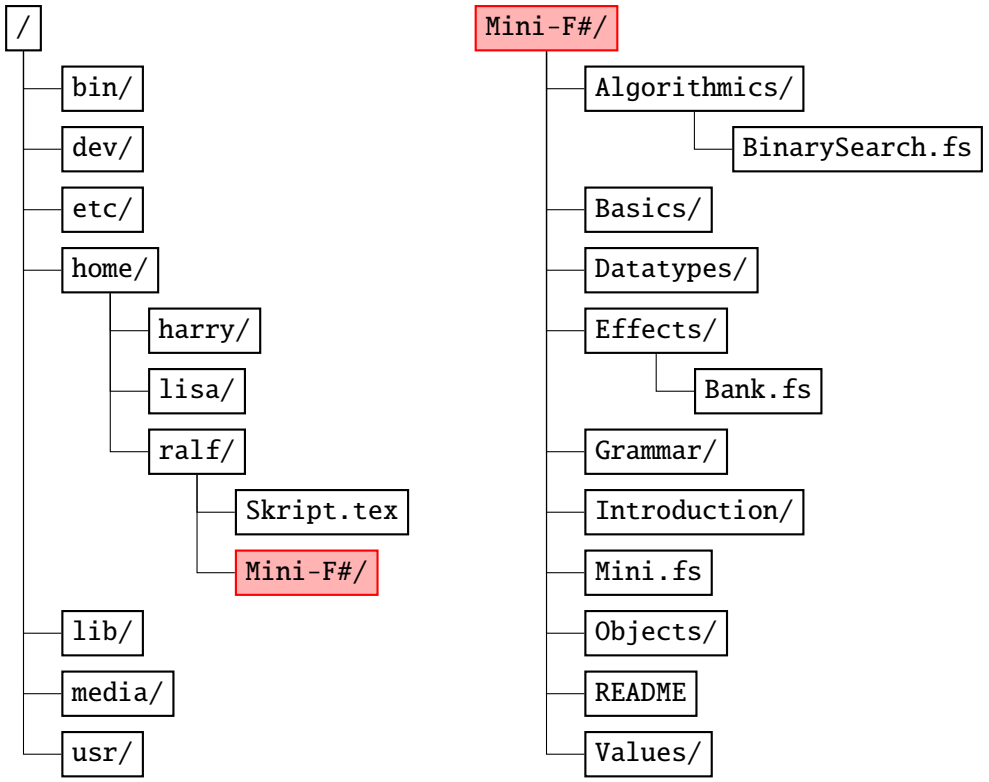

Abbildung 1: Verzeichnisstruktur eines Linux-Dateisystems. Mit unserem HackerFS lassen sich ähnliche Strukturen modellieren.

In der Klasse `HackerFS` führen wir die Funktionalitäten des Dateisystems zusammen. Wir arbeiten damit in-place auf dem Dateisystem. Dazu merken wir uns Referenzen auf das Wurzelverzeichnis und das aktuell ausgewählte Arbeitsverzeichnis (*working directory*). Im aktuell gewählten Arbeitsverzeichnis können dann Operationen auf den Inhalten des Verzeichnisses durchgeführt werden.

Um das Arbeiten mit unserem HackerFS interessanter und angenehmer zu machen, ist in der `Main` Klasse eine minimalistische Shell implementiert, die gängigen Linux Shells nachempfunden ist. Sobald Sie die entsprechenden Methoden in `File.java`, `Directory.java` und `HackerFS.java` implementiert haben, können Sie damit in ihrem Dateisystem navigieren und Operationen auf den Dateisystemobjekten vornehmen.

- `exit` beendet das Programm
- `ls` („list“) listet die Namen der im Arbeitsverzeichnis enthaltenen Objekte auf.
- `ll` („list long“) listet den Inhalt des Arbeitsverzeichnisses mit zusätzlichen Informationen zu den Objekten auf.
- `pwd` („print working directory“) gibt den Pfad des aktuellen Arbeitsverzeichnisses aus.
- `mkdir verzeichnisname` erstellt ein Verzeichnis mit dem Namen `"verzeichnisname"`.
- `rm fsobjekt` löscht eine Datei oder ein leeres Verzeichnis mit dem Namen `"fsobjekt"` im aktuellen Arbeitsverzeichnis.
- `cd` wechselt in das Wurzelverzeichnis
- `cd verzeichnisname` wechselt in das Verzeichnis `"verzeichnisname"` im aktuellen Arbeitsverzeichnis.
- `cd ..` wechselt in das Elternverzeichnis des aktuellen Arbeitsverzeichnisses.
- `cat dateiname` gibt den Inhalt der Datei `"dateiname"` aus.
- `find` gibt den Pfad aller Dateien und Order aus, die im aktuellen Arbeitsverzeichnis enthalten sind.
- `find substring` sucht im aktuellen Arbeitsverzeichnis und dessen Unterordnern nach Dateien und Ordnern, die `"substring"` im Namen enthalten.

Der Einfachheit halber arbeiten wir immer nur im aktuellen Arbeitsverzeichnis. Befehle wie z. B. `cd foo/bar` sind nicht möglich. Stattdessen müssten wir `cd foo` gefolgt von `cd bar` tippen.

Um nicht bei jedem neuen Start des Programms mit einem leeren Dateisystem zu starten, erstellt die Funktion `clone` in der `Main` Klasse ein HackerFS Abbild des Projektverzeichnisses mit einigen der darin enthaltenen Dateien und Ordnern. Den Inhalt der `Main` Klasse können Sie nach Belieben ändern, z. B. können Sie den Aufruf der `clone` Methode entfernen, wenn Sie lieber mit einem leeren Dateisystem starten möchten. Wenn Sie beim Aufruf von `clone` den Pfad ändern möchten, sollten Sie darauf achten, kein Verzeichnis zu clonen, das viele oder große Dateien enthält.

Hinweis: An einigen Stellen möchten Sie evtl. prüfen, ob ein `File` oder ein `Directory` Objekt vorliegt. Einen entsprechenden Typtest können Sie mit **`instanceof`** durchführen (s. dazu auch Folie 101).

Tipp zur Vorgehensweise:
Implementieren Sie die Klassen in der unten genannten Reihenfolge. Die `File` Klasse ist die am wenigsten komplexe, daher empfiehlt es sich mit ihr zu beginnen. Danach sollten Sie die `Directory` Klasse implementieren. Beide Klassen werden in `FSObjectTest` getestet. Erst danach kann die `HackerFS` Klasse sinnvoll implementiert (und mit `HackerFSTest` getestet) werden. Beachten Sie, dass die `clone` Methode, die von der Mini Shell aufgerufen wird, die Methoden `createDirectory`, `enterDirectory`, `leaveDirectory`, `createEmptyFile` und `writeFile` aus `HackerFS` aufruft. Wenn Sie diese noch nicht implementiert haben, aber die Shell trotzdem schon starten möchten, weil Sie evtl. andere Befehle testen möchten, dann sollten Sie den Aufruf der `clone` Methode in der `main` Methode auskommentieren.

Verwenden Sie zur Lösung dieser Aufgabe keine Bibliotheksfunktionen. Davon ausgenommen sind Listen und wenn Sie möchten die StringBuilder Klasse.

a) Implementieren Sie die fehlenden Methoden in der Klasse `File` und definieren Sie geeignete Attribute. Details zur Implementierung entnehmen Sie bitte den Kommentaren in der Vorlage.

   Laden Sie zur Abgabe dieses Aufgabenteils die Datei `File.java` in ExClaim hoch.

b) Implementieren Sie die fehlenden Methoden in der Klasse `Directory` und definieren Sie geeignete Attribute.

   Hinweis: Ein `Directory` Objekt soll eine Liste über darin enthaltene Dateisystemobjekte führen. Definieren Sie ein entsprechendes Attribut und verwenden Sie eine Listenimplementierung aus der Standardbibliothek, z. B. `private final List<FSObject> contents = new ArrayList<>();`. Bei der Implementierung der Klasse `Directory` sollten Sie mit den folgenden Methoden der Liste auskommen:

   - Mit der Methode `boolean isEmpty()` können Sie prüfen, ob eine Liste leer ist.
   - Mit `add(e)` fügen Sie ein `FSObject e` zur Liste hinzu.
   - Mit `remove(e)` entfernen Sie ein `FSObject e` aus der Liste.

   Um zu prüfen, ob ein Dateisystemobjekt mit einem bestimmten Namen in einem Verzeichnis enthalten ist, iterieren Sie über die Inhaltsliste und vergleichen Sie den Namen jedes Objekts mit dem gesuchten Namen.

   Weitere Details zur Implementierung entnehmen Sie bitte den Kommentaren in der Vorlage.

   Laden Sie zur Abgabe dieses Aufgabenteils die Datei `Directory.java` in ExClaim hoch.

c) Implementieren Sie die fehlenden Methoden in der Klasse `HackerFS`. Achten Sie darauf, dass das Wurzelverzeichnis nicht gelöscht werden darf.

   Weitere Details zur Implementierung entnehmen Sie bitte den Kommentaren in der Vorlage.

   Laden Sie zur Abgabe dieses Aufgabenteils die Datei `HackerFS.java` in ExClaim hoch.

d) *Freiwillige Zusatzaufgabe*: Erweitern Sie das Dateisystem und ggf. die Mini-Shell um weitere Funktionalität, z. B. `grep`, `move`, `copy`. In den zusätzlichen Methoden dürfen Sie gerne die Standardbibliothek in vollem Umfang verwenden. Bitte achten Sie darauf, dass die mitgelieferten Testfälle trotz Erweiterung bestanden werden.