Project Report
Group twitchplayseth

Java and C# in depth, Spring 2014

Leonhard Helminger
Marc Gähwiler
Philipp Gamper

April 6, 2014

1 Introduction

This document describes the design and implementation of the *Personal Virtual File System* of group *twitchplayseth*. The project is part of the course *Java and C# in depth* at ETH Zurich. The following sections describe each project phase, listing the requirements that were implemented and the design decisions taken. The last section describes a use case of using the *Personal Virtual File System*.

2 VFS Core

The *VFS Core* provides a basic interface that allows to create and remove a *VFS container file* and perform certain operations on an existing *VFS container file* that are required to fulfill all requirements mentioned in the next section.

The main API classes are

VDisk
: Provides an interface to create a new *VFS container file*, delete an existing *VFS container file* or perform certain operations on an existing *VFS container file*.

VDirectory
: Represents a directory in the *VFS*. Provides an interface to get the

entries of the directory, add and remove entries and copy or delete the whole directory including it's entries.

VFile

Represents a file in the *VFS*. Provides an interface to write and read from the file and to copy or delete the file.

2.1 Requirements

In this section all requirements, that were implemented in the first part, are listed. For each requirement there is a short description of the requirement and how it is implemented in the project.

The virtual disk must be stored in a single file in the working directory in the host file system
The idea behind the virtual disk is to store a whole file system in a single file in an existing file system. This requirement was implemented was implemented in `VDisk`.

***VFS* must support the creation of a new disk with the specified maximum size at the specified location in the host file system** A new *VFS* can be created using the `VDisk` class. To create a new *VFS* it is necessary to format the *VFS file* using the `format` method of `VDisk` before it is used for the first time.

***VFS* must support several virtual disks in the host file system**
Once a *VFS* file has been created and formatted, it can be opened by using the constructor of `VDisk`. The implementation allows to create and open an unlimited amount of *VFS files* which each contain their own *VFS*. There is no limitation of how many *VFS files* are opened in parallel at runtime (excluding system limitation like amount of available RAM or physical diskspace).

***VFS* must support disposing of the virtual disk**
A previously created *VFS file* can be deleted with the `discard` method of `VDisk`. It simply removes the *VFS file* from the host's filesystem.

***VFS* must support creating/deleting/renaming directories and files**
All necessary interfaces that handle file/directory creation, deletion and renaming are implemented as methods of `VDisk`, namely `touch/mkdir`, `delete` and `move`.

VFS **must support navigation: listing of files and folders, and going to a location expressed by a concrete path**

VFS **must support moving/copying directories and files, including hierarchy**

VFS **must support importing files and directories from the host file system**

VFS **must support exporting files and directories to the host file system**

VFS **must support querying of free/occupied space in the virtual disk**

2.2 Design

Give an overview of the design of this part and describe in general terms how the implementation works. You can mention design patterns used, class diagrams, definition of custom file formats, network protocols, or anything else that helps understand the implementation.

2.2.1 General Design

2.2.2 VFS File Format

2.2.3 Design Patterns

3 Quick Start Guide

[optional: This part has to be completed by April 8th.]

If you have a command line interface for your VFS, describe here the commands available (e.g. ls, copy, import).