TECHNISCHE UNIVERSITÄT DRESDEN

**Fakultät Informatik,** Institut für Technische Informatik, Professur VLSI-Entwurfssysteme, Diagnostik und Architektur

# Implementierung eines Dateisystems für Java-basierte eingebettete Systeme

(Verteidigung Bachelorarbeit)

Dresden, 7. Aug 2012

# Gliederung

- Motivation
- Aufgabenstellung
- Auswahl des Dateisystems
- FUSE
- Blockgerät
- Testfälle
- Ausblick und Zusammenfassung

# Motivation

- Wenig Speicher (RAM: 1 MiB)
- Autonomer Agent ist nicht immer im Netzwerk
- → persistente Datenspeicherung
- → Festplatte und Dateisystem

# Aufgabenstellung

- SHAP – Java Bytecode ausführender Prozessor
- SATA-Controller
- Festplatte
- Gesucht: Dateisystemtreiber (java.io.*)

# Auswahl des Dateisystems

- Kriterien:
  - Operiert auf Festplatten
  - Geringe Komplexität
  - Beachtung der Einschränkungen der Dateisystem- und Dateigröße
  - Dateirechte
  - Verbreitung

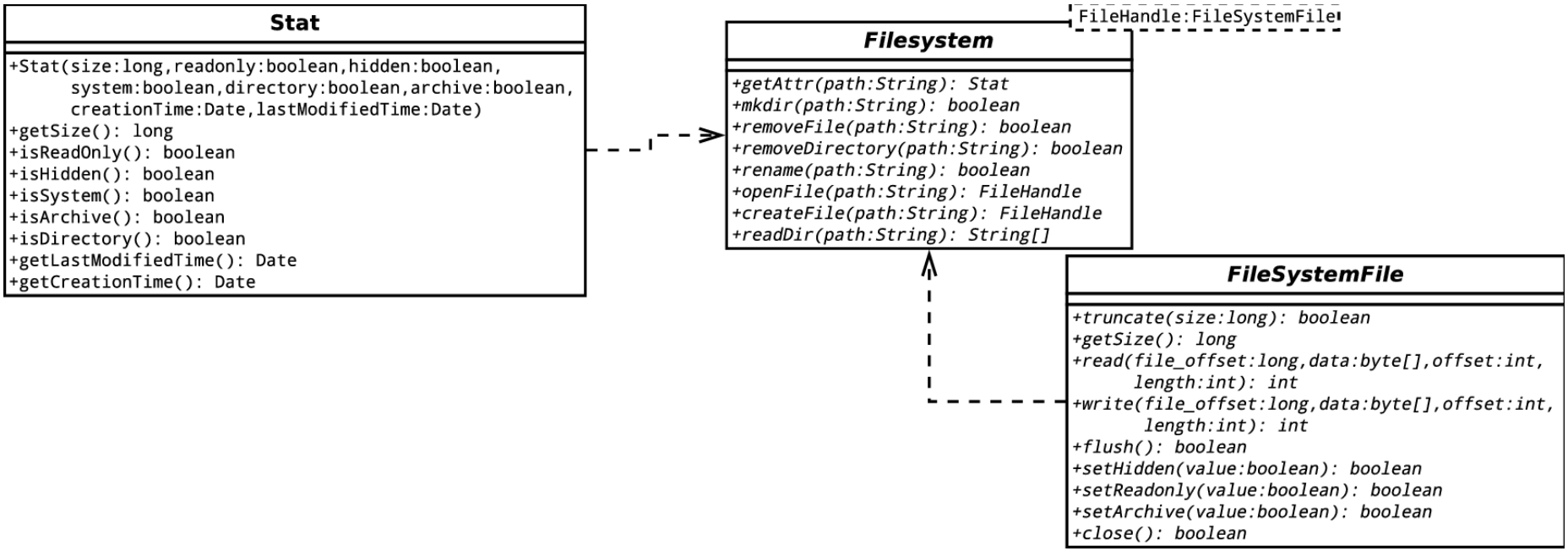
| Dateisystem | Verbreitung | Komplexität | Dateirechte |
|---|---|---|---|
| HFS | MAC | mittel | ja |
| NTFS | Win, Linux, (MAC) | hoch | ja |
| ext2 | (Win), Linux, (MAC) | gering | ja |
| FAT | Win, Linux, MAC | gering | nein |

# java.io.*

- Verzeichnisoperationen
  - → java.io.File
  - Erstellung, Löschung, Existenzprüfung von
    - Ordnern
    - Dateien
  - Auflisten von Verzeichnissen
    - → FileFilter
    - → FileNameFilter
- Dateihandles
  - → FileInputStream
  - → FileOutputStream
  - → RandomAccessFile

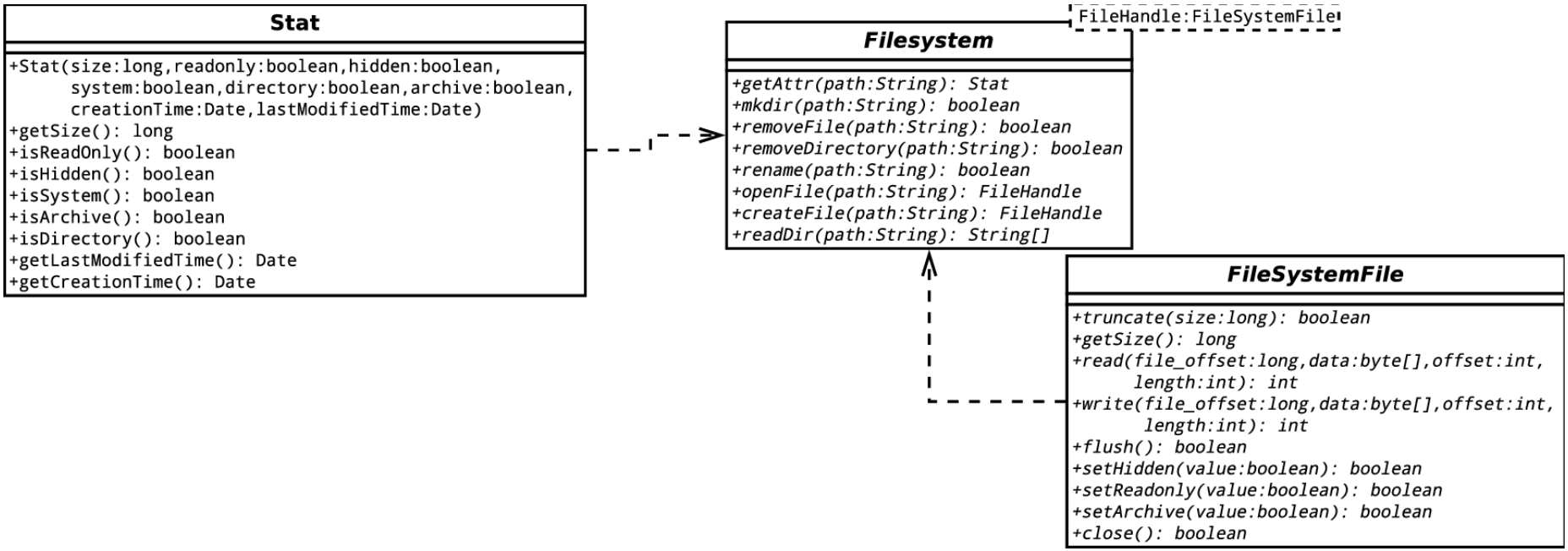
# FUSE

- java.io.* muss auf das FUSE-Interface wrappen
- Dateisysteme implementieren FUSE
- Globaler Bezugspunkt File.rootFS



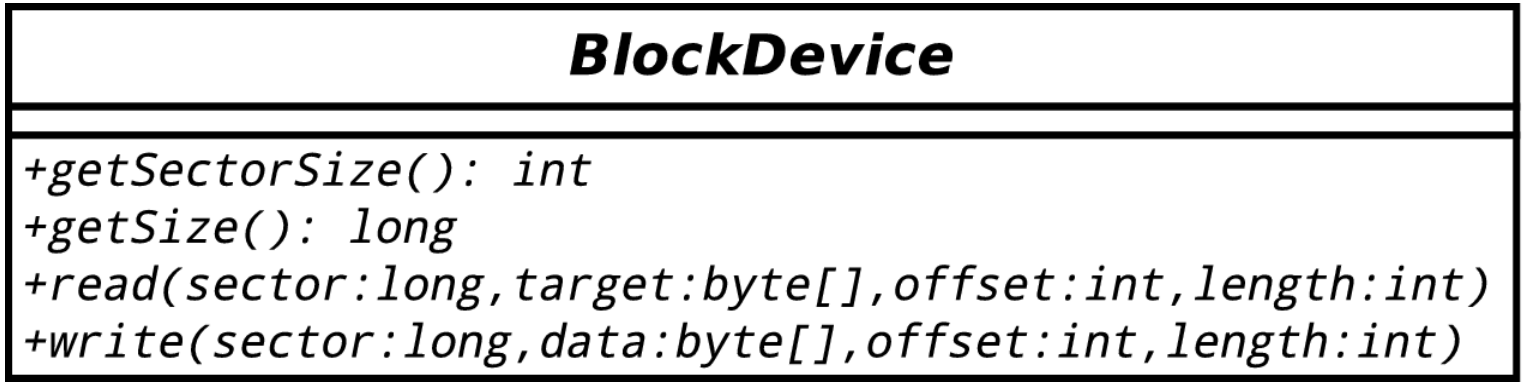
# FUSE

- java.io.* muss auf das FUSE-Interface wrappen
- Dateisysteme implementieren FUSE
- Globaler Bezugspunkt File.rootFS



# Blockgeräte

- SATABlockDevice
- VirtualBlockDevice
- RamBlockDevice
- SubBlockDevice



# Implementierung

- Testumgebung
  - Festplatte mit DOS-Partiton
  - erste Partition mit FAT formatiert
  - Diverse Dateien, Ordner und Unterordner mit
    - Langen Dateinamen
    - Umlauten
    - Verschachtelten Ordnern
    - Großen Dateien
    - Großen Ordnern
- Initialisierung
  - Blockgerät laden (VirtualBlockDevice)
  - Partitionsmanager → Blockgerät
  - Partition → FATFileSystem
  - java.io.File.rootFS = ...

# Leseoperationen

- Verzeichnisse auflisten
  - Directory → startScan(), nextEntry(), nextScan()
  - Fallunterscheidung FAT16 /, FAT32 /, Unterordner
  - DirectoryEntry
  - FileSystem.readdir()
- findFileEntry()
  - stat() / getAttr()
  - fileOpen()
- Hilfsfunktionen
  - readFAT()
  - writeFAT()
  - getNextFreeCluster()
  - getSectorOfCluster()

# Schreiboperationen

- Verzeichniseinträge
  - Erstellen
    - Namen generieren
    - Prüfsumme
    - Laufnummern
    - Platz für Verzeichniseinträge finden
  - Entfernen
    - Clusterkette
    - LFN-Einträge
  - Operationen
    - mkdir()
    - rename()
    - fileCreate()
    - remove()

# Dateihandles

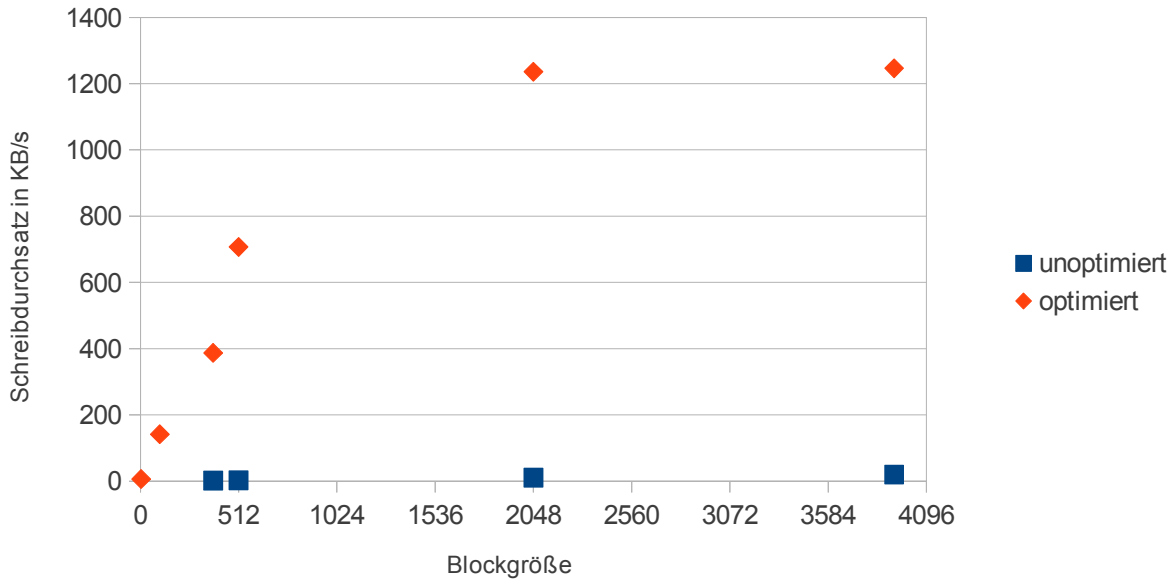
- from: DirectoryEntry
- truncate()
- read()
- write()

# Portierung auf reale Hardware

- Kurzfristig (eine Woche)
- Gregoreanischer Kalender!!!
- Keine Schreib-Operation (→ Testen möglich, aber kein Optimieren)
- Bugs im Linker (Optimierung, statische Methoden der Elternklasse)
- Naiv implementierte read()- und write()-Operation optimiert (Lesezeiger-Cache)

# Benchmark auf dem SHAP

- Große Blöcke: 1,2 MB/s
- 4-Byte-Blöcke: 6,25 KB/s



# Weiteres Optimierungspotenzial

- Sektoren-Caches
  - Jeder Sektor nur ein mal im RAM
  - Pro: FAT schneller, kleine Datenmengen, reRead(), flush()
  - Contra: Daten direkt durchreichen, Speicher
- Caches in den java.io.*-Klassen
  - Pro: kleine Datenmengen schneller, Dateisystemübergreifend
  - Contra: Kooperation des Nutzers bei flush()
- Prefetch
  - Verkettete Liste → nächster Cluster

# Testfälle

- Möglichst jede Methode mit jeder Konstellation testen (Problem: Zustand des Dateisystems)
- Testfall-Klasse arbeitet auf java.io.* (nicht auf der Zwischenschicht)

# Testfälle

- Analyse: Wo finden Programmverzweigungen statt
  - Unterverzeichnis vs / in FAT16 vs / in FAT32
  - Vergrößern vs Verkleinern
  - Einzelsektoren vs Sektorübergreifend vs Clusterübergreifend
- Umfassende Tests vorher/nachher
  - Reichlich exists()-Prüfungen
  - Möglichst redundant den Zustand über alle Kanäle abfragen
    - File.length() nach write abfragen
    - Gleichzeitig Schreibzeiger prüfen
- Realistische Szenarien
  - Datei umkopieren
  - Objekt in RAF schreiben, seek(), auslesen
- → Viele Bugs gefunden, sowohl im Backend als auch im Wrapper

# Perspektiven

- Neue Dateisysteme
  - ext2
  - Präfix-Manager (VFS oder Laufwerksbuchstaben)
  - Pipe zum PC
- Optimierungen

# Ergebnisse

- Entscheidung für FAT
- Implementierung von java.io.File als Wrapper auf eine Zwischenschicht
- Implementierung von FAT als Spezialisierung der FUSE-Klassen
- Umfangreiche Sammlung von Testfällen

# Fragen

- Fragen
- ~~Anregungen~~ Einwürfe
- ~~Vorschläge~~ Kritik

# Quellen


- Filesystem in Userspace. http://fuse.sourceforge.net/
- Microsoft EFI FAT32 File System Specification. http://msdn.microsoft.com/en-us/windows/hardware/gg463080.aspx
- Kapitel Dateisysteme. In: TANENBAUM, Andrew S.: Moderne Betriebssysteme. Pearson Studium, 2002, S. 407ff

»Wissen schafft Brücken.«

| Dateisystem | Windows | Linux | MAC |
|---|---|---|---|
| FAT/VFAT | x | x | x |
| HFS/HFS+ | | | x |
| ext2 | (x) | x | (x) |
| ext3/ext4 | | x | (x) |
| NTFS | x | x | (x) |

Tabelle 3.1: Dateisystemunterstützung etablierter Betriebssysteme

| Eigenschaft | FAT/VFAT[Tan02] | NTFS[Sta03] | ext2[ext12] |
|---|---|---|---|
| Maximale Dateisystemgröße | 4 TiB | 16 EiB | 4 TiB |
| Maximale Dateigröße | 2 GiB | 16 EiB | 2 GiB |
| Journaling | nicht vorhanden | vorhanden | nicht vorhanden |
| Dateirechte | nicht vorhanden | vorhanden | vorhanden |
| Implementierungskomplexität in Codezeilen | 7190 | 29587 | 9561 |

Tabelle 3.2: Dateisysteme

| Blockgröße | unoptimiert | optimiert |
|---|---|---|
| 3928 | 19,83 | 1246,72 |
| 2048 | 10,36 | 1236,17 |
| 512 | 2,6 | 706,85 |
| 379 | 1,92 | 387,12 |
| 101 | | 141,58 |
| 4 | | 6,25 |

| Dateisystem | FAT-Größe in Bits | Maximale Clustergröße | Maximale Dateisystemgröße |
|---|---|---|---|
| FAT12 | 12 | 512 B | 2 MiB |
| FAT12 revidiert | 12 | 4 KiB | 16 MiB |
| FAT16 | 16 | 32 KiB | 2 GiB |
| FAT32 | 32 | 32 KiB | 2 TiB |

Tabelle 2.1: Dateisystemgrößen von FAT

# Umgebung und Einschränkungen

- SHAP auf einem FPGA
- 1 MiB RAM für Programm und Daten
- Maximal 16.000 Elemente im Array
- byte[] nimmt selben Speicherplatz ein wie int[]
- Speichereffizienter SATA-Controller müsste in int[] schreiben
- java.io.File und Anhang bevorzugen byte[] für Lese- und Schreiboperationen
- Verarbeitungsbandbreite beim Umkopieren von int[] nach byte[]: 2 MB/s

# Anforderungen an das Dateisystem

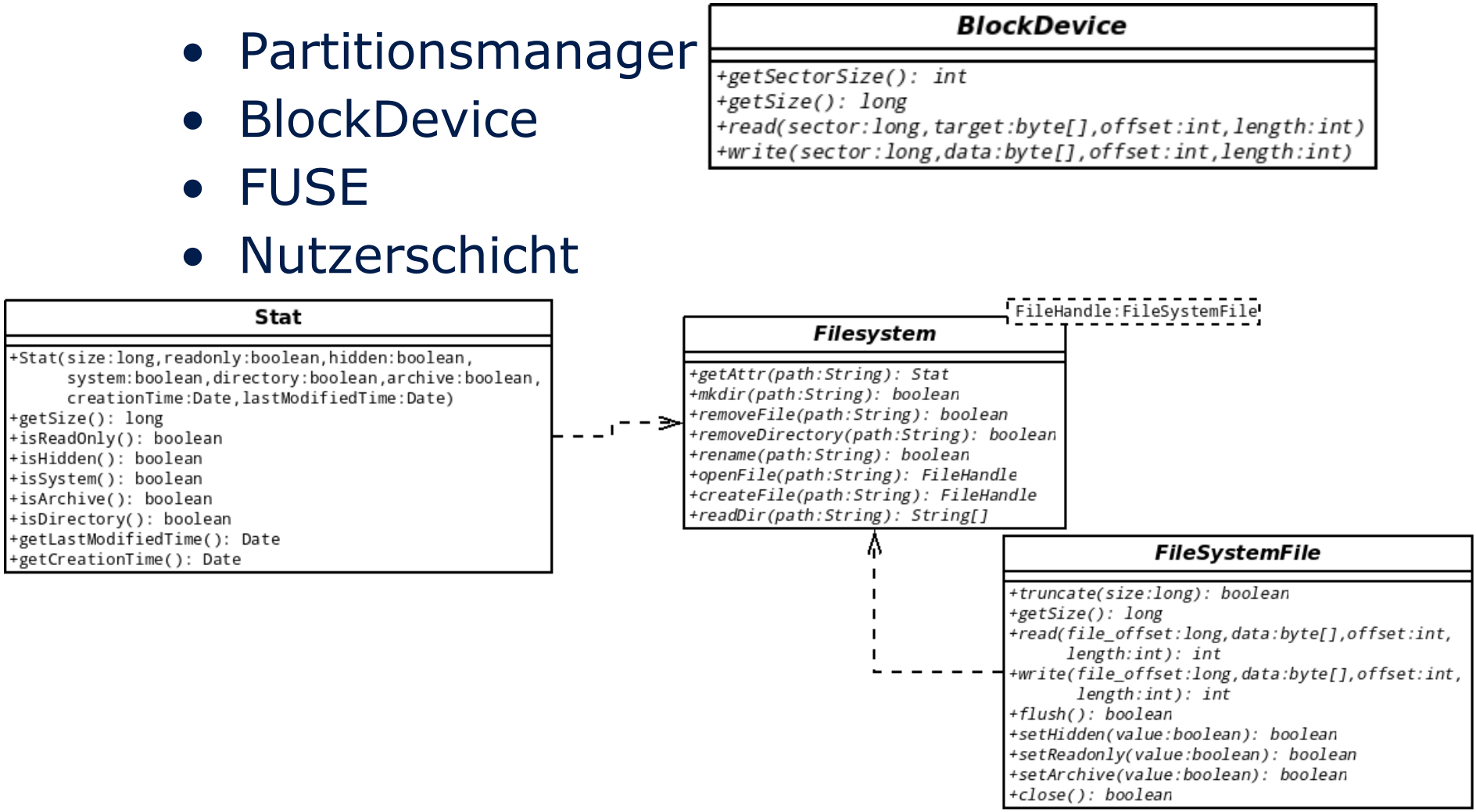
- Sollte mit handelsüblichen Festplatten umgehen
- Festplatte sollte von normalem PC aus schreib- und lesbar sein
- Ziel: java.io.File und Hilfsklassen mit selbem Interface ansprechbar
- → PC-Komfort auf einem Embedded System

# Das java.io.File-Interface

- Benutzerorientiert
- Benötigt „eigentlich“ Betriebssystem
- → Dateisystemunabhängige Zwischenschicht
- FUSE-Interface erfüllt diese Anforderung

# Schichten

- Partitionsmanager
- BlockDevice
- FUSE
- Nutzerschicht



# Anmerkungen zum Entwurf

- Partitionsmanager wird initialisiert
- Aus einer Partition wird das Dateisystem erstellt
- Optional: Darüberschalten eines Laufwerkbuchstaben-Wrappers oder VFS
- Dieses Dateisystem wird File als / zugewiesen
- Anschließend kann man ite.java.io.File ganz normal benutzen

# Details zu FAT

- FAT = File Allocation Table
- Feld von „Next“-Zeigern einer verketteten Liste
- Einfach verkettete Cluster
- Sonderfälle: Frei, Beschädigter Cluster, End of File, End of Clusterchain
- Ein Cluster besteht aus einem oder mehreren Sektoren: Sektorgrößen von 512 Byte bis 32 KiB
- Aufbau: MBR/Reserved, FATs, [Wurzelverzeichnis mit fester Länge], Daten

# Dateien und Ordner in FAT

- Verzeichnis: Liste von 32 Byte großen Einträgen
- Jedem echten Eintrag gehen die LFN-Einträge voraus
- Jeder Eintrag enthält Dateinamenteile und Attribute
- Jeder Datei- und Verzeichniseintrag hat Dateigröße und Zeiger auf den ersten Cluster
- Datei/Verzeichnis ist verkettete Liste von Clustern
- Verzeichnisse haben Einträge „.“ und „..“

# Pro - Contra

- **Negativ**
  - Clustergrößen 32 KiB ↔ SHAP-Arrays
  - Vielfalt der FAT-Implementierungen
  - Komplexität aufeinander aufbauender Erweiterungen und deren Abwärtskompatibilität
- **Positiv**
  - Gute Dokumentation von Microsoft
  - Leicht verständliches Prinzip von FAT
  - Gute Dateisystemtools für Testumgebung
  - Gute Abstraktionsmöglichkeit der Dateisystemproblematik als Klassen