universität
uulm

1. Klausur zur Vorlesung

Paradigmen der Programmierung

im Sommersemester 2021

Institut für Softwaretechnik und Programmiersprachen
Prof. Dr. Matthias Tichy

15.10.2021

Musterlösung

Aufgabe 1 - OOP

2 + 4 + 8 = 14 Punkte

Betrachten Sie folgenden Sachverhalt:
In einem Beet werden reihenweise Pflanzen eingesät. Reihen werden mit den Großbuchstaben *'A-Z'* bezeichnet. Dies funktioniert, da kein Beet (Patch) länger als 24 Reihen oder kürzer als 3 Reihen ist.

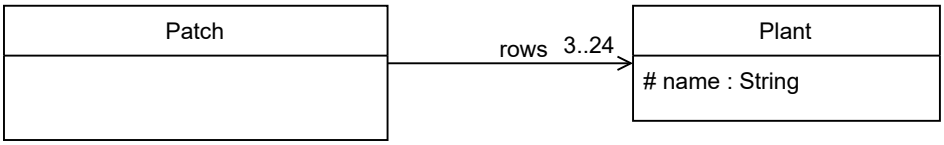


Abb. 1: Klassendiagramm

a) Implementieren Sie die Klasse `Patch` und deren Konstruktor `Patch(int size)`, welcher ein Beet der gegebenen Größe, ohne eingepflanzte Pflanzen, anlegt. Stellen Sie in Ihrer Implementierung sicher, dass alle Eigenschaften aus dem obigen Sachverhalt und Klassendiagramm eingehalten werden und behandeln Sie Verstöße sinnvoll.
Hinweis: Sie dürfen gegebenenfalls Hilfsattribute hinzufügen.

Lösung

```
1  public class Patch {
2      HashMap<Character, Plant> rows = new HashMap<>();
3      int size;

4      public Patch(int size) {
5          if (size < 3 || size > 24) {
6              throw new IllegalArgumentException();
7          }
8          this.size = size;
9      }
10 }
```

b) Implementieren Sie die Methode `addPlantAt` der Klasse `Patch`, welche in einer gegebenen Reihe eine Pflanze pflanzt und `true` zurückliefert wenn dies möglich war. Eine Pflanze kann nicht gepflanzt werden, wenn bereits eine Pflanze in der gegebenen Reihe gepflanzt ist oder die Reihe nicht existiert.

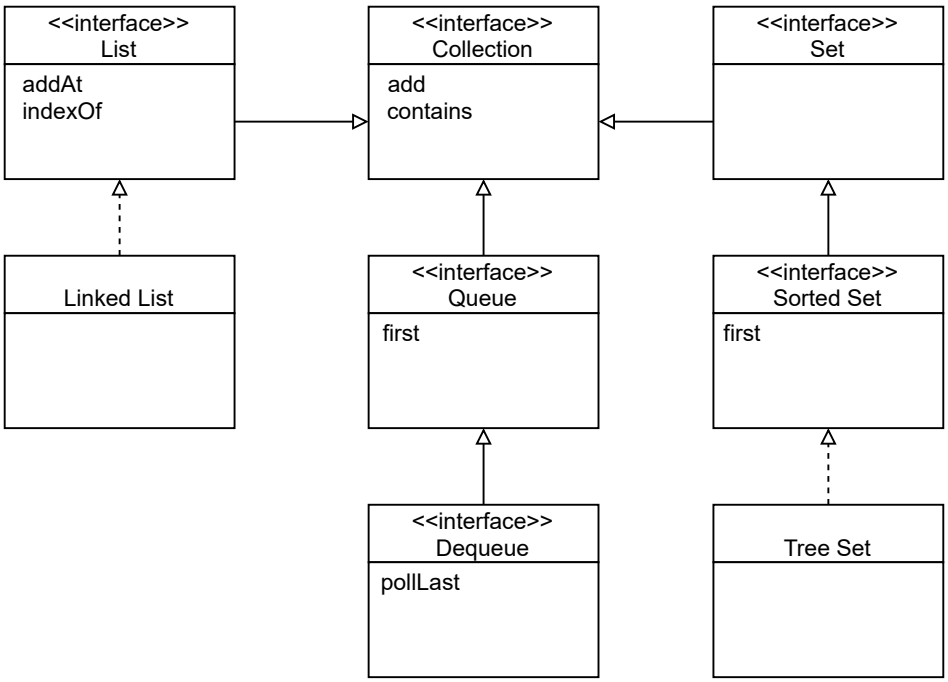
Lösung

```
1  public boolean addPlantAt(char row, Plant plant) {
2      Plant mp = rows.get(row);
3      if (mp == null || row < 'A' || row > 'Z') {
4          return false;
5      }
6      rows.put(row, plant);

7      return true;
8  }
```

c) Ergänzen Sie das folgende Klassendiagramm zur Java Collections-API um Vererbungen und Interface-Implementierungen. Annotieren Sie außerdem Interfaces und Abstrakte Klassen korrekt. Fügen Sie zuletzt die Namen der folgenden Operationen in die Klassen ein, bei denen die jeweilige Operation Sinn ergibt. Es reicht hierbei aus, die Operation in die höchstgelegene Klasse einer Vererbungshierarchie einzutragen.

- *add*: fügt ein übergebenes Element ein
- *first*: liefert das erste Element zurück
- *addAt*: fügt ein übergebenes Element an einer bestimmten Stelle ein
- *contains*: liefert zurück ob ein übergebenes Element existiert
- *indexOf*: liefert die Position eines übergebenen Elements zurück
- *pollLast*: liefert das letzte Element zurück und entfernt es



Aufgabe 2 - JavaIO

7 + 2 + 2 = 11 Punkte

Betrachten Sie folgende Klasse:

```
1 public interface Stack<T> {

2     public void push(T element);

3     public T pop();
4 }

5 public class StackLockedException extends RuntimeException {//...}
```

a) Implementieren Sie eine Klasse `LockedStack<T>` auf Basis der Klasse `Stack<T>`. Verwenden Sie hierfür das **Decorator-Pattern**. Ein `LockedStack` soll mittels der Methoden `lock` und `open` geöffnet und geschlossen werden. Ist der `LockedStack` geschlossen, so soll bei *push* oder *pop* Operationen eine `StackLockedException` geworfen werden. Ein geöffneter `LockedStack` verhält sich wie ein regulärer `Stack`.

— **Lösung** ———————————————————————————

```java
public class LockedStack<T> implements Stack<T> {
    private Stack<T> stack;
    private boolean locked = false;

    public LockedStack(Stack<T> stack) {
        this.stack = stack;
    }

    public void lock() {
        this.locked = true;
    }

    public void open() {
        this.locked = false;
    }

    private void checkLock() {
        if(locked) {
            throw new StackLockedException();
        }
    }

    @Override
    public void push(T element) {
        checkLock();
        stack.push(element);
    }

    @Override
    public T pop() {
        checkLock();
        return stack.pop();
    }
}
```

b) Beschreiben Sie eine Gemeinsamkeit und einen Unterschied zwischen einem `ObjectStream` und einem `DataStream` in ganzen Sätzen.

Lösung

Beide Streams arbeiten auf binären Daten. Größter unterschied, ist, dass ObjectStream beliebige Objekte beinhalten kann während DataStream nur Basidatentypen und Strings kennt.

c) Implementieren Sie die Methode `printIfExist(String... paths)`, welches für jeden übergebenen Pfad, den Pfad ausgibt so wie ein Hinweis dazu ob das Ziel des Pfades ein Ordner ist.
Hinweis: Beachten Sie hierfür auch den JavaNIO Teil des **CheatSheets** am Ende der Klausur.

Lösung

```
1 private record Pair (Path path, boolean exists) {};

2 public static void printIfExist(String... paths) throws IOException {
3     Arrays.stream(paths)
4           .map(path -> Path.of(path))
5           .map(path -> new Pair(path, Files.isDirectory(path)))
6           .forEach(System.out::println);
7 }
```

Aufgabe 3 - XML, JSON, Versionierung **3 + 6 + 3 = 12 Punkte**

Betrachten Sie die folgende *pom.xml* (auch zu finden am Ende der Klausur).

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0">
3      <groupId>de.uulm.sp.pvs</groupId>

4      <build>
5          <pluginManagement>
6              <plugins>
7                  <plugin>
8                      <artifactId>maven-clean-plugin</artifactId>
9                      <version>3.1.0</version>
10                 </plugin>
11                 <plugin>
12                     <artifactId>maven-reports-plugin
13                       </artifactId>
14                     <version>3.0.0</version>
15                 </plugin>
16             </plugins>
17         </pluginManagement>
18     </build>
19 </project>
```

a) Sie sollen mittels eines Java-Programms dieses XML-Dokument verarbeiten. Nennen Sie *2* Ansätze hierfür. Erläutern Sie zudem, in ganzen Sätzen, **2** Punkte die geklärt werden müssten, um sich auf einen der beiden Ansätze festlegen zu können.

Lösung

SAX vs. DOM
verfügbarer Speicherplatz
RO oder RW?
Geschwindigkeit

b) Definieren Sie ein JSON Dokument, welches äquivalent zum gegebenen XML Dokument ist.

Lösung

```
{
    "project": {
        "xmlns": "http://maven.apache.org/POM/4.0.0",
        "groupId": "de.uulm.sp.pvs",
        "build": {
            "pluginManagement": {
                "plugins": [
                    {
                        "artifactId": "maven-clean-plugin",
                        "version": "3.1.0"
                    },
                    {
                        "artifactId": "maven-reports-plugin",
                        "version": "3.0.0"
                    }
                    ]
                }
            }
        }
}
```

c) Nennen und beschreiben Sie **3** Vorteile, die sich durch die Verwendung eines Versionierungsystems wie z.B git oder svn ergeben.

Lösung

einfaches verteiltes Arbeiten
Historie des Projekts
Nachvollziehbarkeit von Änderungen
Branching
...

Aufgabe 4 - ER-Modellierung

5 + 1 + 4 + 2 = 12 Punkte

Betrachten Sie folgenden Sachverhalt (dieser steht in **keinem** Zusammenhang zu dem aus Aufgabe **1** bekannten):

Ein Beet besteht aus zwischen 3 und 24 Reihen. Pro Reihe können zwischen 10 und 30 Pflanzen gepflanzt werden, hierbei ist die Größe der Pflanzen ausschlaggebend. Ein Beet und dessen zugehörige Reihen werden von Personen bewirtschaftet. Hierbei gilt, dass jedes Beet von mindestens einer Person bewirtschaftet werden muss, während eine Person jeweils nur maximal 3 Beete bewirtschaften kann.

a) Modellieren Sie den beschriebenen Sachverhalt als **E-R-Diagramm**. Verwenden Sie UML-Notation für die Angabe von Kardinalitäten.

Lösung

Person $1..*-<$ *bewirtschaftet* $>-0..3$ Beet
Beet $1-<$ *bestehtAus* $>-3..24$ Reihe
Reihe $0..*-<$ *istBepflanztMit* $>-10..30$ Pflanze

b) Nennen Sie **alle** Sachverhalte aus der Beschreibung, die in Ihrem **E-R-Diagramm** nicht modelliert werden können.

Lösung

- Die Anzahl 10..30 abhängig von der Größe.

c) Erstellen Sie zu dem E-R-Diagramm das dazugehörige **relationale Datenbankschema**. Achten Sie hierbei auf sinnvoll gewählte **Primärschlüssel** und stellen Sie sicher, dass das Schema mindestens in 3. Normalform ist.

Lösung

Beet(**BID**)
PersonsInBeet(***PID***,***BID***)
Person(**PID**)
Reihe(**RID**, *BID*)
PflanzeInReihe(***RID***, ***PfID***)
Pflanze(**PfID**, Groesse)

d) Beschreiben Sie welche Eigenschaften eine Relation R besitzen muss um in 3. Normalform zu sein.

Lösung

kein Nichtschlüsselattribut transitiv vom Schlüssel abhängig
kein Nichtschlüsselattribut funktional abhängig von einer echten Teilmenge eines Schlüsselkandidaten
Attributwerte atomar
keine Wiederholungsgruppen

Aufgabe 5 - SQL

0.5 + 3.5 + 4 + 4 + 2 = 14 Punkte

Gegeben seien die folgenden Relationenschemata (auch zu finden auf dem beiliegenden Extrablatt):

Pferd		
PID	Name	**HID**

Auktion	
AID	Datum

Gebot		
PID	**AID**	Preis

Halter	
HID	HName

Primärschlüssel sind unterstrichen, Fremdschlüssel sind **fett** dargestellt. Formulieren Sie, insofern nicht anders spezifiziert, die **SQL-Statements** zur Lösung folgender Teilaufgaben:

a) Löschen Sie die Tabelle *Auktion*.
Hinweis: Dies geschieht nur im Kontext dieser Teilaufgabe und hat keine Auswirkungen auf andere Teilaufgaben.

Lösung

```
DROP TABLE Auktion
```

b) Fügen Sie die Tabelle *Stall* mit den Attributen *Name*, *Capacity* mit sinnvollen Datentypen und einem Primärschlüssel ein.
Hinweis: Dies geschieht nur im Kontext dieser Teilaufgabe und hat keine Auswirkungen auf andere Teilaufgaben.

Lösung

```
CREATE TABLE Stall(
ID INT NOT NULL,
NAME VARCHAR(20) NOT NULL,
CAPACITY INT NOT NULL,
PRIMARY KEY (ID)
);
```

c) Erhöhen Sie den Preis aller Gebote, für das Pferd mit dem Namen *Max* um 100.

Lösung

```
UPDATE Gebot
SET Preis = Preis + 100
WHERE PID IN (
SELECT PID
FROM Pferd
WHERE Name = 'Max'
)
```

d) Geben Sie das höchste Gebot für alle Pferde, die an der Auktion am 31.12.1991 teilgenommen haben und mehr als 3 Gebote erhalten haben, aus.

Lösung

```
SELECT Name, MAX(Preis)
FROM Pferd
NATURAL JOIN Gebot
NATURAL JOIN Auktion
WHERE Datum = '31.12.1991'
HAVING COUNT(Preis) > 3
GROUP BY Name
```

e) Was sind im Kontext von SQL in Java die sogenannten *PreparedStatement*s, wozu dienen Sie, und wie funktionieren Sie?
Hinweis: Sie können Ihre Erklärung durch ein Codebeispiel unterstützen, müssen dies aber nicht.

Lösung

Vorgefertigte SQL Statements in die Parameter übergeben werden.
Verhindern SQL Injektion
Durch das echte parametrisieren wird verhindert, dass zusätzlich übergebene SQL Befehle ausgeführt werden.

Aufgabe 6 - GUI

2 + 5 + 4 + 4 = 15 Punkte

Betrachten Sie folgenden Scene Graph:

```
<BorderPane prefHeight="500.0" prefWidth="700.0"
  xmlns="http://javafx.com/javafx/8.0.111"
  xmlns:fx="http://javafx.com/fxml/1">
    <center>
        <BorderPane>
            <top>
                <ToolBar>
                    <items>
                        <Button background-color="blue">
                            <graphic>
                                <ImageView pickOnBounds="true"
                                  preserveRatio="true">
                                    <image>
                                        <Image url="@/open.png" />
                                    </image>
                                </ImageView>
                            </graphic>
                        </Button>
                    </items>
                </ToolBar>
            </top>
            <center>
                <SplitPane orientation="VERTICAL">
                    <items>
                        <AnchorPane minHeight="0.0" minWidth="0.0">
                            <children>
                                <Label text="no file loaded..."/>
                                <ScrollPane fx:id="imageScrollPane">
                                    <content>
                                        <ImageView fx:id="imageView"/>
                                    </content>
                                </ScrollPane>
                            </children>
                        </AnchorPane>
                    </items>
                </SplitPane>
            </center>
        </BorderPane>
    </center>
</BorderPane>
```

a) Nennen Sie **2** Stellen im SceneGraph die Beispiele für eine Anwendung des Composite-Patterns sind und erklären Sie, in ganzen Sätzen, auf Basis derer, was dieses Pattern ist.

Lösung

z.B Z1 und Z5, wir haben eine BorderPane in einer BorderPane.
Dies Funktioniert da BorderPane ein Composite des Patterns ist uns somit andere Components/Nodes beinhalten kann (somit auch sich selbst). Das Composite Pattern beschreibt eine Gruppe von Objekten, die wie eine einzelne Instanz desselben Objekttyps behandelt werden. Das Pattern ermöglicht die Gleichbehandlung von primitiven Objekten und Behältern und erlaubt somit eine einfache Repräsentation von verschachtelten Strukturen. Das Pattern sorgt für hohe Flexibilität und Erweiterbarkeit erschwert allerdings die Unterscheidung/ unterschiedliche Behandlung einzelner Komponenten.

b) Erklären Sie, in ganzen Sätzen, was das Observer-Pattern ist und wie es funktioniert. Beschreiben Sie zusätzlich **2** Nachteile des Patterns und wieso diese aufgrund des Aufbaus des Patterns existieren.

Lösung

Observer-Pattern beschreibt eine Struktur von 2 Komponenten, Observer und Observable. Observer können ein Observable beobachten und auf Aktionen auf dem Observable reagieren. Dies wird mittels einer notify() Methode realisiert die das Observable bei allen seinen Observern aufruft.
Problem: Notifikationen werden über ein Objekt gehandhabt und sind somit nicht typsicher/ erlauben keine Generizität
Problem: Da es nur die notify() Methode gibt kann keine genauere Fallunterscheidung auf dieser Ebene erfolgen.

c) Erklären Sie, in ganzen Sätzen, wozu Model-View-Controller Architektur dient. Beschreiben Sie zusätzlich die Aufgaben **aller 3** Komponenten.

Lösung

Dient der sauberen Trennung der verschiedenen Aufgaben in einem grafischen Programm.
Model: Stellt Daten dar
View: Verantwortlich für die Darstellung (von Daten)
Controller: Verbindungsstück zwischen Model und View. Empfängt z.B Informationen/Events der View und formt die Daten für die Model-Schicht um.

d) Ergänzen Sie folgenden Code so, dass ein Fenster mit Button angezeigt wird und beim Mausklick des Buttons der Text *Ouch!* auf der Console ausgegeben wird.

Lösung

```
1  int counter = 0;
2  public void start(Stage primaryStage) {
3      var sp = new ScrollPane();
4      var button = new Button();
5      sp.getChildren().add(button);
6      button.setOnMouseClicked(new EventHandler<MouseEvent>() {
7          @Override
8          public void handle(MouseEvent event) {
9              System.out.println("Ouch!");
10         }
11     });
12     primaryStage.setScene(new Scene(sp));
13     primaryStage.show();
14 }
```

Aufgabe 7 - Threads

3 + 7 + 2 = 12 Punkte

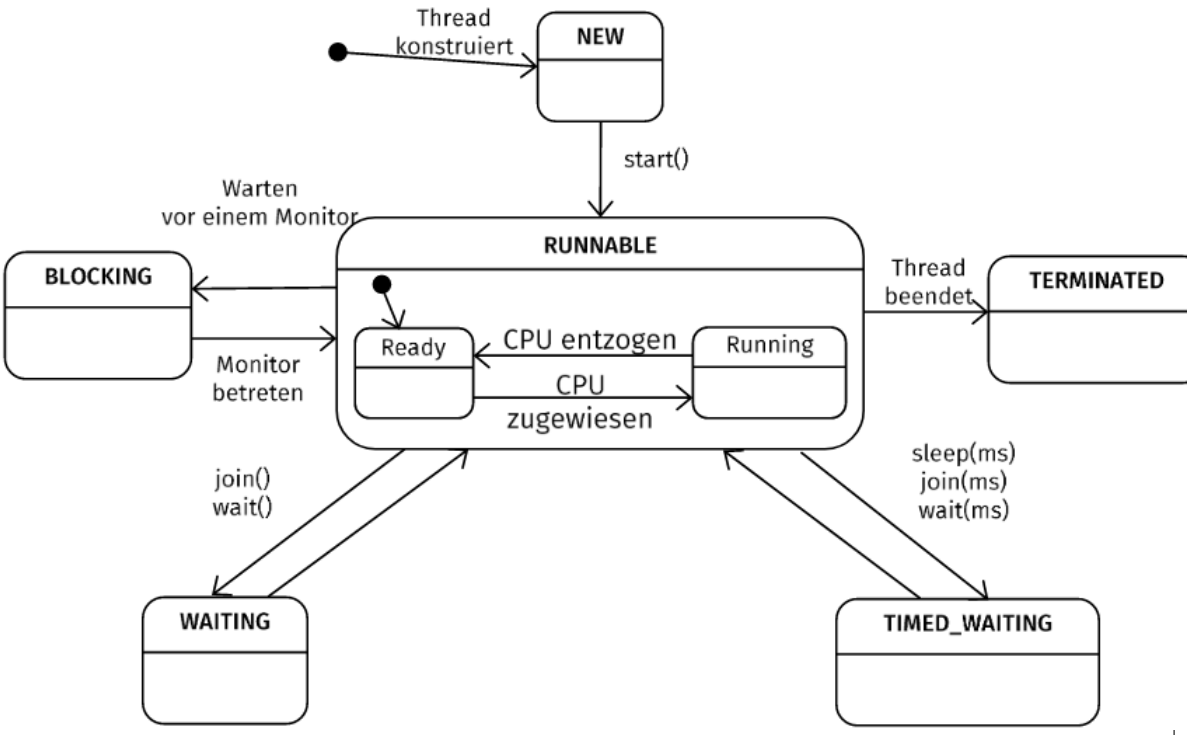
a) Was besagt *Ahmdal's law* und welche Schlussfolgerung kann daraus für die Parallelisierung von Programmen gezogen werden?

Lösung

Ahmdal's law ist ein Modell, dass die theoretische maximale Beschleunigung eines Programms durch Parallelisierung beschreibt.
Es zeigt auf, dass die Beschleunigung eines Programms Asymptotisch ist und somit nicht unendlich hohe Geschwindigkeitsgewinne durch das Erhöhen der involvierten Prozessoren möglich ist.

b) Stellen Sie die verschiedenen Zustände, die ein Thread annehmen kann und wie diese ineinander übergehen, mittels eines (hierarchischen) Zustandsautomaten dar.

Lösung



c) Betrachten Sie folgenden Code:

```
1  public class Locks extends Thread {

2      Object l1;
3      Object l2;

4      public static void main(String[] args) {
5          var aLock = new Object();
6          var anotherLock = new Object();
7          var locks = new Locks(aLock, anotherLock);
8          var blocks = new Locks(locks, aLock);

9          blocks.doOtherStuff();
10         locks.doStuff();
11     }

12     public Locks(Object a, Object b) {
13         l1 = a;
14         l2 = b;
15     }

16     public void doOtherStuff() {
17         synchronized (l1) {
18             synchronized (l2) {
19                 //...
20             }
21         }
22         synchronized (this) {
23             //...
24         }
25     }

26     public synchronized int doStuff() {
27         return 1337;
28     }
29 }
```

Geben Sie die Namen der Objekte in der Reihenfolge an, in der sie durch die Aufrufe in den Zeilen *9* und *10* verwendet werden, um den Programmablauf zu synchronisieren.

Lösung

locks, aLock, blocks, locks

Aufgabe 3 - XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0">
    <groupId>de.uulm.sp.pvs</groupId>

    <build>
        <pluginManagement>
            <plugins>
                <plugin>
                    <artifactId>maven-clean-plugin</artifactId>
                    <version>3.1.0</version>
                </plugin>
                <plugin>
                    <artifactId>maven-reports-plugin
                      </artifactId>
                    <version>3.0.0</version>
                </plugin>
            </plugins>
        </pluginManagement>
    </build>
</project>
```

Aufgabe 5 - SQL

Pferd		
PID	Name	**HID**

Auktion	
AID	Datum

Gebot		
PID	**AID**	Preis

Halter	
HID	HName

JavaNIO

Files

Modifier and Type	Method	Description
static `Stream<Path>`	`find(Path start, int maxDepth, BiPredicate<Path,BasicFileAttributes> matcher, FileVisitOption... options)`	Return a `Stream` that is lazily populated with `Path` by searching for files in a file tree rooted at a given starting file.
static `Object`	`getAttribute(Path path, String attribute, LinkOption... options)`	Reads the value of a file attribute.
static `<V extends FileAttributeView> V`	`getFileAttributeView(Path path, Class<V> type, LinkOption... options)`	Returns a file attribute view of a given type.
static `FileStore`	`getFileStore(Path path)`	Returns the `FileStore` representing the file store where a file is located.
static `FileTime`	`getLastModifiedTime(Path path, LinkOption... options)`	Returns a file's last modified time.
static `UserPrincipal`	`getOwner(Path path, LinkOption... options)`	Returns the owner of a file.
static `Set<PosixFilePermission>`	`getPosixFilePermissions(Path path, LinkOption... options)`	Returns a file's POSIX file permissions.
static boolean	`isDirectory(Path path, LinkOption... options)`	Tests whether a file is a directory.
static boolean	`isExecutable(Path path)`	Tests whether a file is executable.
static boolean	`isHidden(Path path)`	Tells whether or not a file is considered *hidden*.
static boolean	`isReadable(Path path)`	Tests whether a file is readable.
static boolean	`isRegularFile(Path path, LinkOption... options)`	Tests whether a file is a regular file with opaque content.
static boolean	`isSameFile(Path path, Path path2)`	Tests if two paths locate the same file.
static boolean	`isSymbolicLink(Path path)`	Tests whether a file is a symbolic link.
static boolean	`isWritable(Path path)`	Tests whether a file is writable.
static `Stream<String>`	`lines(Path path)`	Read all lines from a file as a `Stream`.
static long	`size(Path path)`	Returns the size of a file (in bytes).
static `Stream<Path>`	`walk(Path start, int maxDepth, FileVisitOption... options)`	Return a `Stream` that is lazily populated with `Path` by walking the file tree rooted at a given starting file.
static `Stream<Path>`	`walk(Path start)`	Return a `Stream` that is lazily populated with `Path` by walking the file tree rooted at a given starting file.
static `Path`	`walkFileTree(Path start, FileVisitor<? super Path> visitor)`	Walks a file tree.

There are two hard things in computer science: cache invalidation, naming things, and off-by-one errors.

JavaNIO

Path

Modifier and Type	Method	Description
FileSystem	getFileSystem()	Returns the file system that created this object.
Path	getName(int index)	Returns a name element of this path as a Path object.
int	getNameCount()	Returns the number of name elements in the path.
Path	getParent()	Returns the *parent path*, or null if this path does not have a parent.
Path	getRoot()	Returns the root component of this path as a Path object, or null if this path does not have a root component.
int	hashCode()	Computes a hash code for this path.
boolean	isAbsolute()	Tells whether or not this path is absolute.
default Iterator<Path>	iterator()	Returns an iterator over the name elements of this path.
Path	normalize()	Returns a path that is this path with redundant name elements eliminated.
static Path	of(String first)	Returns a Path by converting a path string, or a sequence of strings that when joined form a path string.
static Path	of(URI uri)	Returns a Path by converting a URI.
default WatchKey	register(WatchService watcher, WatchEvent.Kind<?>... events)	Registers the file located by this path with a watch service.
WatchKey	register(WatchService watcher, WatchEvent.Kind<?>[] events, WatchEvent.Modifier... modifiers)	Registers the file located by this path with a watch service.
Path	relativize(Path other)	Constructs a relative path between this path and a given path.
default Path	resolve(String other)	Converts a given path string to a Path and resolves it against this Path in exactly the manner specified by the resolve method.
Path	resolve(Path other)	Resolve the given path against this path.
default Path	resolveSibling(String other)	Converts a given path string to a Path and resolves it against this path's parent path in exactly the manner specified by the resolveSibling method.
default Path	resolveSibling(Path other)	Resolves the given path against this path's parent path.
default boolean	startsWith(String other)	Tests if this path starts with a Path, constructed by converting the given path string, in exactly the manner specified by the startsWith(Path) method.
boolean	startsWith(Path other)	Tests if this path starts with the given path.
Path	subpath(int beginIndex, int endIndex)	Returns a relative Path that is a subsequence of the name elements of this path.
Path	toAbsolutePath()	Returns a Path object representing the absolute path of this path.
default File	toFile()	Returns a File object representing this path.
Path	toRealPath(LinkOption... options)	Returns the *real* path of an existing file.
String	toString()	Returns the string representation of this path.
URI	toUri()	Returns a URI to represent this path.

There are two hard things in computer science: cache invalidation, naming things, and off-by-one errors.