ulm university universität
uulm

1. Klausur zur Veranstaltung

Programmierung von Systemen

im Sommersemester 2021

Prof. Dr. Matthias Tichy, Stefan Höppner

16.08.2021

Musterlösung

Aufgabe 1 - OOP

3 + 4 + 4 + 3 = 14 Punkte

Betrachten Sie folgende Klassen (auch zu finden am Ende der Klausur) und die zugehörige Systembeschreibung:

```java
package exam01.ex01.design.other;
public enum BookType {
    ADVENTURE, ROMAN, FANTASY;
}
```

```java
package exam01.ex01.design;
public class Library {
    private <typeA> booksByType;
}
```

```java
package exam01.ex01.design.other;
public abstract class AbstractBook {
    protected String title;
    protected <typeB> pages;
}
```

```java
package exam01.ex01.design.other;
public class Page {
    private int number;
    private String content;
}
```

Ein Bekannter bittet Sie darum, Software für seine private Büchersammlung weiter zu entwickeln, die er selbst nicht fertig stellen kann, da ihm das nötige Know-How fehlt. Er wünscht sich, dass mit der Software all seine Bücher komplett digitalisiert werden können, er sich aber immer noch wie in seiner heimischen Bibliothek fühlt. Dafür sollen sich digitale Bücher verhalten wie echte Bücher und weiterhin *geordnet* nach Genre (BookType) und schnell auffindbar in der Bibliothek stehen. Ihr Bekannter versichert ihnen zudem, dass keine zwei Bücher in seiner Sammlung den selben Namen tragen und dass dies auch niemals vorkommen darf.

a) Welche Variablen sind in welcher Klasse sichtbar? Füllen Sie die Nachfolgende Tabelle mit *Ja/Nein* aus. *Nicht ausgefüllte Felder werden als falsch gewertet.*

Lösung

Variable / sichtbar in Klasse	`booksByType`	`content`	`number`	`pages`	`title`
`AbstractBook`	Nein	Nein	Nein	Ja	Ja
`Library`	Ja	Nein	Nein	Nein	Nein
`Page`	Nein	Ja	Ja	Ja	Ja

b) Wählen Sie für `typeA` und `typeB` sinnvolle Datentypen aus. Geben Sie den vollständigen Typen jeweils an und argumentieren Sie **in ganzen Sätzen**, warum die gewählten Datenstrukturen eine sinnvolle Wahl sind.

Lösung

Hier sind viele Entscheidungen möglich z.B:
typeA: HashMap/Map¡BookType, Collection¡AbstractBook», da wir in der Library die Bücher nach Typ ordnen wollen und diese Struktur einen schnellen Zugriff ermöglicht.
typeB: Stack¡Page¿: so verhalten sich Bücher. Man muss eine Seite umblättern um zur nächsten zu kommen.

c) Implementieren Sie, auf Basis ihrer gewählten Datenstrukturen, die Methode `getBookByName(String name)` der Klasse `Library`, welche das Buch mit dem angegebenen Namen aus der Menge aller Bücher zurückgibt. Beschreiben Sie zudem etwaige Änderung(en), welche an der aktuellen Klassenstruktur vorgenommen werden müssten, um diese Implementierung zu ermöglichen.
Hinweis: Beachten Sie hierfür auch den JavaOOP Teil des **CheatSheets** am Ende der Klausur.

Lösung

```
1 // es muss ein getter für title in AbstractBook hinzugefügt werden.
2 public Optional<AbstractBook> findBookByTitle(String title) {
3     return books.values().stream()
4     .flatMap(bookCollection -> bookCollection
5                 .stream().filter($ -> $.getTitle() == title))
6     .findFirst();
7 }
```

d) Implementieren Sie, auf Basis ihrer gewählten Datenstrukturen, die Methode `insertBook(AbstractBook book)` der Klasse `Library`, welche ein neues Buch in der Bibliothek abspeichert. Sollte ein Buch mit gleichem Namen bereits existieren, so soll dieses überschrieben werden. Beschreiben Sie zudem etwaige Änderung(en), welche an der aktuellen Klassenstruktur vorgenommen werden müssten, um diese Implementierung zu ermöglichen.
Hinweis: Beachten Sie hierfür auch den JavaOOP Teil des **CheatSheets** am Ende der Klausur.

Lösung

```java
// es muss ein getter für title in AbstractBook hinzugefügt werden.
public void insertBook(AbstractBook book, BookType type) {
    var bookTitle = book.getTitle();
    var filteredBooks = books.get(type).stream()
      .filter(aBook -> aBook.getTitle() != bookTitle)
      .collect(Collectors.toList());
    filteredBooks.add(book);
    books.put(type, filteredBooks);
}
```

Aufgabe 2 - JavaIO

5 + 3 + 5 = 13 Punkte

a) Sie erhalten nun die Aufgabe eine große Menge an Daten, die in den Klassen aus **Aufgabe 1** gespeichert sind, zu übertragen. Dabei werden Ihnen die folgenden Designentscheidungen vorgelegt:

- Gepufferter Datenstrom vs. Ungepufferter Datenstrom
- Bytestream vs. Objectstream vs. Datastream

Geben Sie an, welche Varianten Sie wählen würden und erläutern Sie **in ganzen Sätzen** Ihre Beweggründe. Gehen Sie dabei auch auf Einschränkungen (z.B. für die Leseseite) ein, die durch Ihre Wahl entstehen.

Lösung

Je nach Begründung kann hier nichts alzu falsch sein. Bsps.:
Buffered Bytestream: Bytream damit Daten unabhängig von Format übertragen werden können, Gegenseite muss dann aber wissen in welcher Reihenfolge Daten übertragen wurden um richtig zurück zu wandeln. Buffered hier weil dann immer ein Block geladen und umgewandelt werden kann während der nächste in den Puffer lädt.
Buffered Datastream: Daten in Klassen gehalten ergo vermutlich primitive Daten. Datastream hat dafür Methoden, andere Seite muss die nur in gleicher Reihenfolge wieder lesen kann sie verarbeiten wie sie will. Buffered damit jeder Read möglichst schnell aus Puffer kommt während der nächste nachlädt.

b) Erklären Sie mit Hilfe eines selbst gewählten Code-Beispiels das **Decorator-Pattern** und welchen Vorteil dieses im Kontext von IO-Streams bietet.

Lösung

Ein Entwurfsmuster, das es erlaubt, einem individuellen Objekt dynamisch Verhalten hinzuzufügen, ohne das Verhalten anderer Objekte derselben Klasse zu beeinflussen. Das Decorator-Pattern ist oft nützlich, um das Single-Responsibility-Prinzip einzuhalten, da es die Aufteilung der Funktionalität zwischen Klassen mit eindeutigen Aufgabenbereichen erlaubt. Die Verwendung von Decorators kann effizienter sein als die Verwendung von Subklassen, da das Verhalten eines Objekts erweitert werden kann, ohne ein völlig neues Objekt zu definieren. Bei IO-Streams erlaubt dies ein kaskadieren der Konstruktoren um einem Stream verschiedene Eigenschaften zu geben.

c) Implementieren Sie die Methode `listFiles(String dir)`, welche alle Dateinamen ausgibt, die unter einem angegebenen Pfad liegen. Dabei sollen immer **nur** die Namen von Dateien ausgegeben werden, nicht die von Ordnern. Die Methode soll stattdessen *rekursiv* auch alle Dateinamen in gefundenen Ordnern ausgeben.
Hinweis: Beachten Sie hierfür auch den JavaNIO Teil des **CheatSheets** am Ende der Klausur.
Ein `Stream` kann in eine Liste mittels `.collect(Collectors.toList())` umgewandelt werden sollte dies notwendig sein.

Lösung

```java
public static void listFilesUsingFilesList(String dir) throws IOException {
    Files.walk(Path.of(dir))
      .filter(Files::isRegularFile)
      .map(Path::getFileName)
      .forEach(System.out::println);
}
```

Aufgabe 3 - XML und Build-Automatisierung **2 + 5 + 3 = 10 Punkte**

Betrachten Sie die folgende *pom.xml*.

```
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      <artifactId>Klausur</artifactId>

5      <dependencies>
6          <dependency>
7              <groupId>org.java-websocket</groupId>
8              <artifactId>Java-WebSocket</artifactId>
9              <version />
10         <dependency />
11         <dependency>
12             <groupId>junit</groupId>
13             <artifactId>junit</artifactId>
14             <version>4.11</version>
15             <scope>test</scope>
16         </dependency>
17     </ dependencies>
18     <build>
19         <pluginManagement>
20         <plugins>
21             <plugin>
22                 <artifactId>maven-compiler-plugin</artifactId>
23                 <version>3.8.0</wersion>
24             </plugin>
25         <plugins>
26         <pluginManagement>
27     </build>
28 </project>
```

a) Das XML Dokument ist nicht wohlgeformt. Geben Sie die Zeilennummern **vier** vorkommender Fehler an und erklären Sie, wie diese korrigiert werden können.

Lösung

Z2: keine Anführungszeichen um Wert des Attributs herum.
Z7: Schließender Tag fehlt.
Z23: öffnender und schließender Tag sollten gleich heißen.
Z26: sollte schließender Tag sein.

b) Beschreiben Sie **in ganzen Sätzen** für die Zeilen **Z3-4**, **Z5-17**, **Z6-10**, **Z11-16**, **Z20-25** in der *pom.xml*, was diese für das zugrundeliegende Maven-Projekt bedeuten.

Lösung

Z3-4: Definieren Metadaten für Projekt, groupId und artifactId.
Z5-17: Definieren die Externen Libraries die im Projekt verwendet werden können.
Z6-10: Definiert, dass das Java-WebSocket Paket im Projekt eingebunden wird.
Z11-16: Definiert, dass das JUnit Paket nur für Tests eingebunden wird.
Z20-25: Definiert Plugins die während der Build-Phase des Projekts verwendet werden können.

c) Nennen und beschreiben Sie **2** Vorteile und **einen** Nachteil, die die Verwendung von Buildautomatisierungs-Tools wie Maven mit sich bringen.

Lösung

z.B.: Dependency Management wird vollständig übernommen. Projekte können einfach auf verschiedenen Rechnern importiert und verwendet werden. Bauen und Packen von Entwickelten Projekten/Libraries kann automatisiert werden. Zusätzliches Tooling muss installiert sein um Projekte verwenden zu können.

Aufgabe 4 - ER-Modellierung

5 + 2 + 4 + 1 = 12 Punkte

An einem Film sind mehrere Personen beteiligt. Entweder spielen diese Personen im Film mit oder sie sind Regisseur. In einem Film müssen mindesten 5 Personen mitspielen, aber es gibt nur genau einen Regisseur. Personen können in mehreren Filmen mitspielen und Regisseur sein, nie aber beides gleichzeitig beim selben Film. Filme werden von ihrem Drehstart bis zum Drehende an einem *exklusiven* Drehort gedreht. Drehorte können nach Drehschluss eines Filmes auch für den Dreh neuerer Filme genutzt werden.

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

Lösung

Person $5..* - <$ *actsIn* $> -0..*$ Movie
Person $1- <$ *isRegisseur* $> -0..*$ Movie
Movie $0..* - <$ *filmsAt* $> -0..*$ Location

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

Lösung

- Person nicht gleichzeitig Schauspieler und Regisseur bei gleichem Film. - An Drehort immer nur ein Film gleichzeitig.

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

Person(**PID**)
ActorsInMovie(***PID***,***MID***)
RInMovie(***PID***,***MID***)
Movie(**MID**)
Location(**LID**)
MovieLocation(***MID***,***LID***,start,end)

d) Nennen Sie **alle** Eigenschaften, die aus dem E-R-Diagramm nicht in das relationale Datenbankschema übertragbar sind.

Lösung

Kardinalitäten sind nicht mehr darstellbar.

Aufgabe 5 - SQL

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

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

Schiffe		
SID	Name	**MID**

Rennen			
RID	RennName	Startzeit	Preisgeld

Schiffsrennen	
SID	**RID**

Matrosen	
MID	MName

Besatzung	
MID	**SID**

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 *Schiffe*.
Hinweis: Dies geschieht nur im Kontext dieser Teilaufgabe und hat keine Auswirkungen auf andere Teilaufgaben.

Lösung

```
DROP TABLE Schiffe
```

b) Fügen Sie die Tabelle *Häfen* 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 Häfen(
ID INT NOT NULL,
NAME VARCHAR(20) NOT NULL,
CAPACITY INT NOT NULL,
PRIMARY KEY (ID)
);
```

c) Verdoppeln Sie das Preisgeld aller Rennen, an denen das Schiff des Matrosen *Tichy* teilnimmt.

Lösung

```
UPDATE Rennen
SET Preisgeld = 2 * Preisgeld
WHERE RID IN (
SELECT SR.RID
FROM Rennen
NATURAL JOIN Schiffsrennen as SR
NATURAL JOIN Schiffe
NATURAL JOIN Matrosen
```

```
WHERE MName = 'Tichy'
)
```

d) Fügen Sie für jedes Rennen mit einem Preisgeld *größer* als 10000 ein *'Benefiz'* Rennen mit der selben Startzeit und einem halb so hohen Preisgeld ein.

Lösung

```
INSERT INTO Rennen
SELECT 'Benefiz', Startzeit, Preisgeld/2
FROM Rennen
WHERE Preisgeld > 10000
```

e) Wozu dienen im Kontext von SQL **Transaktionen** und welche Funktionalitäten stellen diese zur Verfügung?

Lösung

Transaktionen erlauben es zusammengehörige Operationen zu "klammernünd somit als Block auszuführen. Transaktionen erlauben es Datenbanken immer in konsistenten Zuständen zu halten und ermöglichen bei Fehlern innerhalb der Transaktion alle anderen Operationen der Transaktion rückgängig zu machen.

Aufgabe 6 - GUI

2 + 5 + 3 + 5 = 15 Punkte

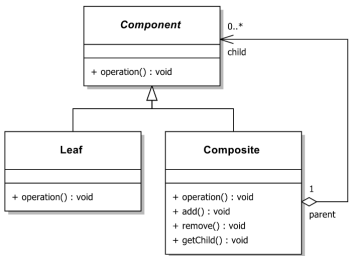
a) Erläutern Sie, was im Kontext von JavaFX der sogenannte **Scene Graph** ist, wie er aufgebaut ist und was er repräsentiert.

Lösung

Der Scene Graph ist ein hierarchischer **Baum**, dessen Knoten alle visuellen Elemente einer FX-Anwendung beinhaltet.

b) Beschreiben Sie das **Composite Pattern** mit Hilfe des Klassendiagrams, welches dieses Pattern abbildet, und erklären Sie hierbei insgesamt **3** Vor- oder Nachteile, die das Pattern mit sich bringt.
Hinweis: Ihr Klassendiagramm muss nicht alle Methoden enthalten, nur solche, die Sie für Ihre Erklärungen benötigen.

Lösung



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.

c) Erklären Sie **in ganzen Sätzen**, warum in Frameworks für grafische Oberflächen (wie z.B. JavaFX) gerne auf Eventhandling zurückgegriffen wird, statt mit dem Observer-Pattern zu arbeiten.

Lösung

Die Update/Notify-Methoden ermöglichen keine Unterscheidung der Ereignisse die beobachtet wurden. Dies muss stattdessen mittels Fallunterscheidungen und Typecasts gehandhabt werden. Events erlauben genau diese Unterscheidung und ermöglichen somit eine dedizierte Behandlung verschiedener Events und sogar das komplette ignorieren irrelevanter Events was mit Observern nicht möglich ist.

d) Welchen Code müssen Sie schreiben, damit beim Mausklick eines Buttons angezeigt wird, wie oft dieser gedrückt wurde?
Hinweis: Verwenden Sie für das Setzen des Textes die Methode `setText(String text)` der Klasse `Button`.

Lösung

```
int counter = 0;
public void start(Stage primaryStage) {
    var sp = new StackPane();
    var button = new Button();
    sp.getChildren().add(button);


    button.setOnMouseClicked(new EventHandler<MouseEvent>() {
        @Override
        public void handle(MouseEvent event) {
            button.setText("" + (++counter));
        }
    });
    primaryStage.setScene(new Scene(sp));
    primaryStage.show();
}
```

Aufgabe 7 - Threads

5 + 3 + 4 = 12 Punkte

a) Nennen und beschreiben Sie **zwei** Probleme, die bei der Synchronisation von Abläufen entstehen können und erläutern Sie jeweils einen kurzen Sachverhalt, in dem das Problem auftritt.

Lösung

Deadlock: mehrere Abläufe warten darauf, dass andere, ebenfalls Wartende Abläufe eine Bedingung erfüllen. Beispiel: 2 Threads benötigen die selben 2 Locks in umgekehrter Reihenfolge. Jeder lockt das erste Lock und wartet nun auf das jeweils andere.
Starvation: Ein Ablauf wartet für immer, dass er Arbeiten darf. Beispiel: 2 Threads benötigen das selbe Lock. Erster Thread nimmt sich das Lock und gibt es nie wieder frei, da er in einer Endlosschleife etwas abarbeitet.

b) *"Je mehr Threads oder Prozesse ich für mein Programm verwende, desto schneller wird es"*.
Bewerten Sie diese Aussage unter den Gesichtspunkten: **zu bearbeitendes Problem** und **Overhead**.
Begründen Sie Ihre Bewertung **in ganzen Sätzen**.

Lösung

Die Aussage in dieser Form ist falsch. Es gibt Probleme für die sich eine Parallelisierung grundsätzlich nicht anbietet, weil eine Partitionierung nicht möglich ist. Außerdem ist es so, dass je geringer die eigentlich durchzuführende Rechnung im Vergleich zum Overhead durch die Erstellung und Abstimmung zwischen den einzelnen Threads ist, desto schlechter ist die Skalierbarkeit.

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 myLock = new Object();
6          var myOtherLock = new Object();
7          var aLocks = new Locks(myLock, myOtherLock);
8          var aOtherLocks = new Locks(aLocks, myLock);

9          aLocks.returnStuff();
10         aOtherLocks.doStuff();
11     }

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

16     public synchronized int returnStuff() {
17         return 42;
18     }

19     public void doStuff() {
20         synchronized (l1) {
21             //...
22         }
23         synchronized (this) {
24             //...
25         }
26         synchronized (l2) {
27             //...
28         }
29     }
30 }
```

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

aLocks, aLocks, aOtherLocks, myLock

Aufgabe 1 - OOP

```java
package exam01.ex01.design.other;
public enum BookType {
    ADVENTURE, ROMAN, FANTASY;
}
```

```java
package exam01.ex01.design;
public class Library {
    private <typeA> booksByType;
}
```

```java
package exam01.ex01.design.other;
public abstract class AbstractBook {
    protected String title;
    protected <typeB> pages;
}
```

```java
package exam01.ex01.design.other;
public class Page {
    private int number;
    private String content;
}
```

Ein Bekannter bittet Sie darum, Software für seine private Büchersammlung weiter zu entwickeln, die er selbst nicht fertig stellen kann, da ihm das nötige Know-How fehlt. Er wünscht sich, dass mit der Software all seine Bücher komplett digitalisiert werden können, er sich aber immer noch wie in seiner heimischen Bibliothek fühlt. Dafür sollen sich digitale Bücher verhalten wie echte Bücher und weiterhin *geordnet* nach Genre (BookType) und schnell auffindbar in der Bibliothek stehen. Ihr Bekannter versichert ihnen zudem, dass keine zwei Bücher in seiner Sammlung den selben Namen tragen und dass dies auch niemals vorkommen darf.

JavaOOP

Optional

Modifier and Type	Method	Description
static <T> Optional<T>	empty()	Returns an empty Optional instance.
boolean	equals(Object obj)	Indicates whether some other object is "equal to" this Optional.
Optional<T>	filter(Predicate<? super T> predicate)	If a value is present, and the value matches the given predicate, returns an Optional describing the value, otherwise returns an empty Optional.
<U> Optional<U>	flatMap(Function<? super T,? extends Optional<? extends U>> mapper)	If a value is present, returns the result of applying the given Optional-bearing mapping function to the value, otherwise returns an empty Optional.
T	get()	If a value is present, returns the value, otherwise throws NoSuchElementException.
int	hashCode()	Returns the hash code of the value, if present, otherwise 0 (zero) if no value is present.
void	ifPresent(Consumer<? super T> action)	If a value is present, performs the given action with the value, otherwise does nothing.
void	ifPresentOrElse(Consumer<? super T> action, Runnable emptyAction)	If a value is present, performs the given action with the value, otherwise performs the given empty-based action.
boolean	isEmpty()	If a value is not present, returns true, otherwise false.
boolean	isPresent()	If a value is present, returns true, otherwise false.
<U> Optional<U>	map(Function<? super T,? extends U> mapper)	If a value is present, returns an Optional describing (as if by ofNullable(T)) the result of applying the given mapping function to the value, otherwise returns an empty Optional.
static <T> Optional<T>	of(T value)	Returns an Optional describing the given non-null value.
static <T> Optional<T>	ofNullable(T value)	Returns an Optional describing the given value, if non-null, otherwise returns an empty Optional.
Optional<T>	or(Supplier<? extends Optional<? extends T>> supplier)	If a value is present, returns an Optional describing the value, otherwise returns an Optional produced by the supplying function.
T	orElse(T other)	If a value is present, returns the value, otherwise returns other.
T	orElseGet(Supplier<? extends T> supplier)	If a value is present, returns the value, otherwise returns the result produced by the supplying function.
T	orElseThrow()	If a value is present, returns the value, otherwise throws NoSuchElementException.
<X extends Throwable> T	orElseThrow(Supplier<? extends X> exceptionSupplier)	If a value is present, returns the value, otherwise throws an exception produced by the exception supplying function.
Stream<T>	stream()	If a value is present, returns a sequential Stream containing only that value, otherwise returns an empty Stream.
String	toString()	Returns a non-empty string representation of this Optional suitable for debugging.

Aufgabe 5 - SQL

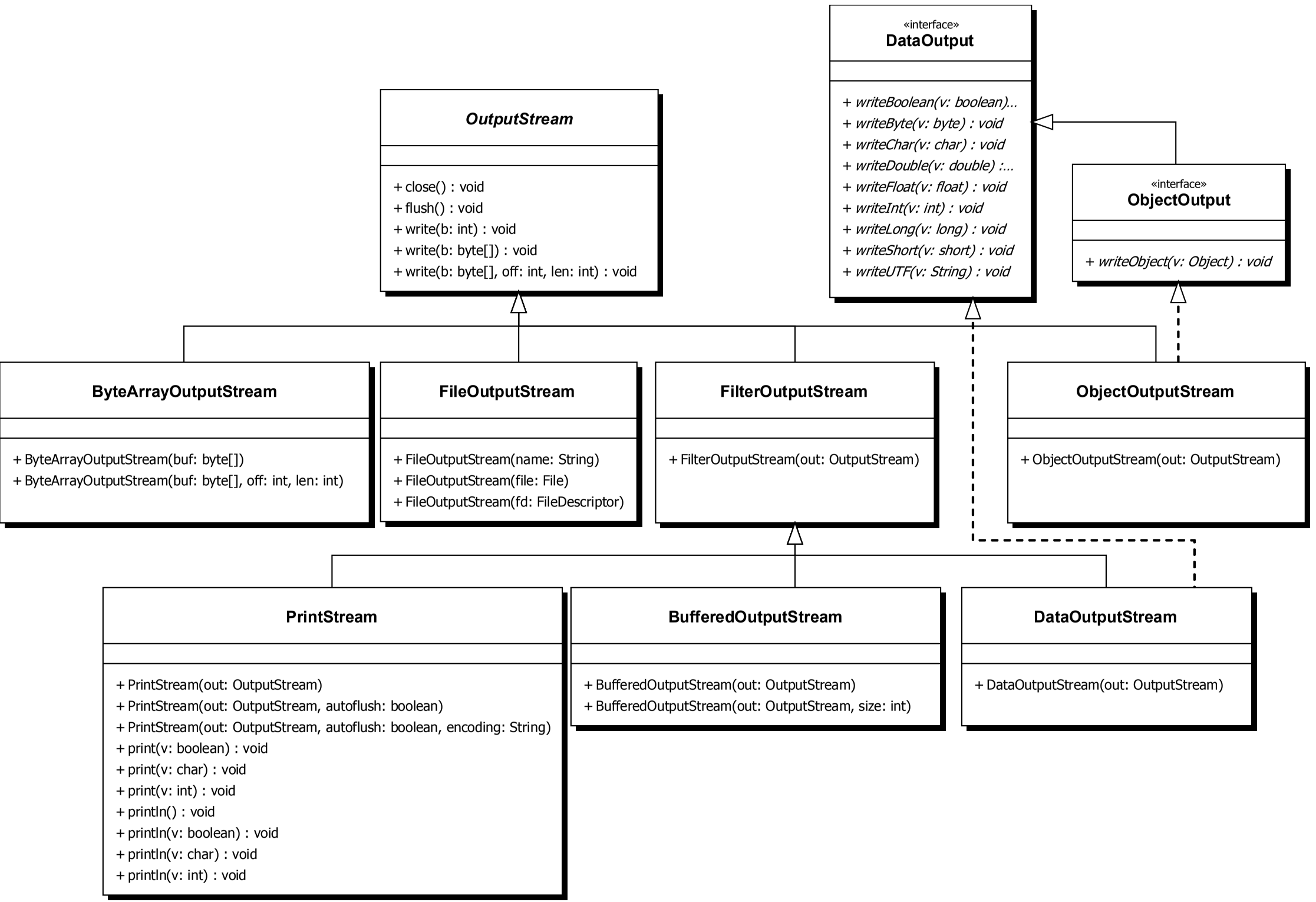
Schiffe		
SID	Name	**MID**

Rennen			
RID	RennName	Startzeit	Preisgeld

Schiffsrennen	
SID	**RID**

Matrosen	
MID	MName

Besatzung	
MID	**SID**


«interface»
DataOutput
+ writeBoolean(v: boolean)...
+ writeByte(v: byte) : void
+ writeChar(v: char) : void
+ writeDouble(v: double) :...
+ writeFloat(v: float) : void
+ writeInt(v: int) : void
+ writeLong(v: long) : void
+ writeShort(v: short) : void
+ writeUTF(v: String) : void
«interface»
ObjectOutput
+ writeObject(v: Object) : void
OutputStream
+ close() : void
+ flush() : void
+ write(b: int) : void
+ write(b: byte[]) : void
+ write(b: byte[], off: int, len: int) : void
ByteArrayOutputStream
+ ByteArrayOutputStream(buf: byte[])
+ ByteArrayOutputStream(buf: byte[], off: int, len: int)
FileOutputStream
+ FileOutputStream(name: String)
+ FileOutputStream(file: File)
+ FileOutputStream(fd: FileDescriptor)
FilterOutputStream
+ FilterOutputStream(out: OutputStream)
ObjectOutputStream
+ ObjectOutputStream(out: OutputStream)
PrintStream
+ PrintStream(out: OutputStream)
+ PrintStream(out: OutputStream, autoflush: boolean)
+ PrintStream(out: OutputStream, autoflush: boolean, encoding: String)
+ print(v: boolean) : void
+ print(v: char) : void
+ print(v: int) : void
+ println() : void
+ println(v: boolean) : void
+ println(v: char) : void
+ println(v: int) : void
BufferedOutputStream
+ BufferedOutputStream(out: OutputStream)
+ BufferedOutputStream(out: OutputStream, size: int)
DataOutputStream
+ DataOutputStream(out: OutputStream)

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.