universität ulm

1. Klausur zur Vorlesung

Programmierung von Systemen

im Sommersemester 2021

Institut für Softwaretechnik und Programmiersprachen
Prof. Dr. Matthias Tichy

16.08.2021, 9 Uhr **Bearbeitungszeit 90min**

Nachname:	Vorname:	Matrikelnummer:
Studiengang und Abschluss:		Platznummer:

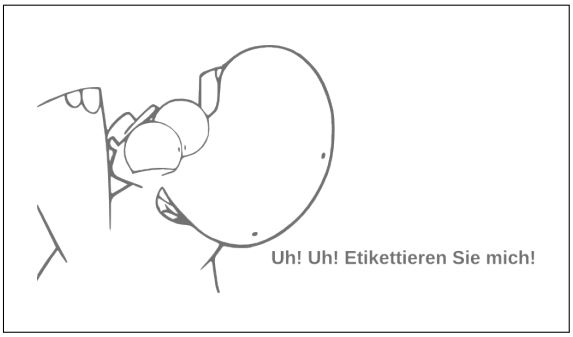
Hiermit erkläre ich, dass ich prüfungsfähig bin.
Sollte ich nicht auf der Liste der angemeldeten Studierenden aufgeführt sein, dann nehme ich hiermit zur Kenntnis, dass diese Prüfung nicht gewertet werden wird.

_____

Datum, Unterschrift des Prüfungsteilnehmenden

Zur allgemeinen Beachtung:

- Füllen Sie das Deckblatt vollständig und korrekt aus.
- Lesen Sie sich zunächst die Klausur sorgfältig durch (sie besteht aus 16 Aufgabenseiten plus Anhang).
- Hilfsmittel sind nicht erlaubt.
- Schreiben Sie ihre Lösungen **lesbar** direkt auf das jeweilige Aufgabenblatt.
- Aufgaben, welche nicht mit einem dokumentenechten Stift in den Farben blau oder schwarz bearbeitet worden sind, werden nicht bewertet.
- Zusätzlich benötigtes Papier wird Ihnen von der Aufsicht zur Verfügung gestellt.



Viel Erfolg !

Punkteverteilung								
1	2	3	4	5	6	7	$\sum$	**Note**
von 14	von 13	von 10	von 12	von 14	von 15	von 12	von 90	
								Korrektur
Einsichtnahme ohne Nachkorrektur ◯				Einsichtnahme mit Nachkorrektur ◯				

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.*

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

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.

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.

```
public Optional<AbstractBook> getBookByName(String name) {
```

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.

```
public void insertBook(AbstractBook book, BookType type) {
```

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.

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

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.

```
public void listFilesUsingFilesList(String dir) throws IOException {
```

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

Betrachten Sie die folgende *pom.xml*.

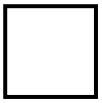
```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns=http://maven.apache.org/POM/4.0.0>
3      <groupId>de.uulm.sp.pvs</groupId>
4      <artifactId>Klausur</artifactId>

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.

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.

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



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.

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

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.

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

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.

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.

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

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.

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

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.

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.

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.

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`.

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

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.

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**.

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.

Falls Sie noch Platz benötigen, so können Sie dieses Blatt benutzen. Weitere Blätter erhalten Sie von der Aufsicht. Machen Sie **eindeutig kenntlich**, welche Aufgaben hier bearbeitet bzw. fortgesetzt werden und **streichen** Sie den Lösungsversuch eindeutig durch, den sie nicht bewertet haben wollen.

Diese Seite wurde für ein besseres Layout leer gelassen.

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

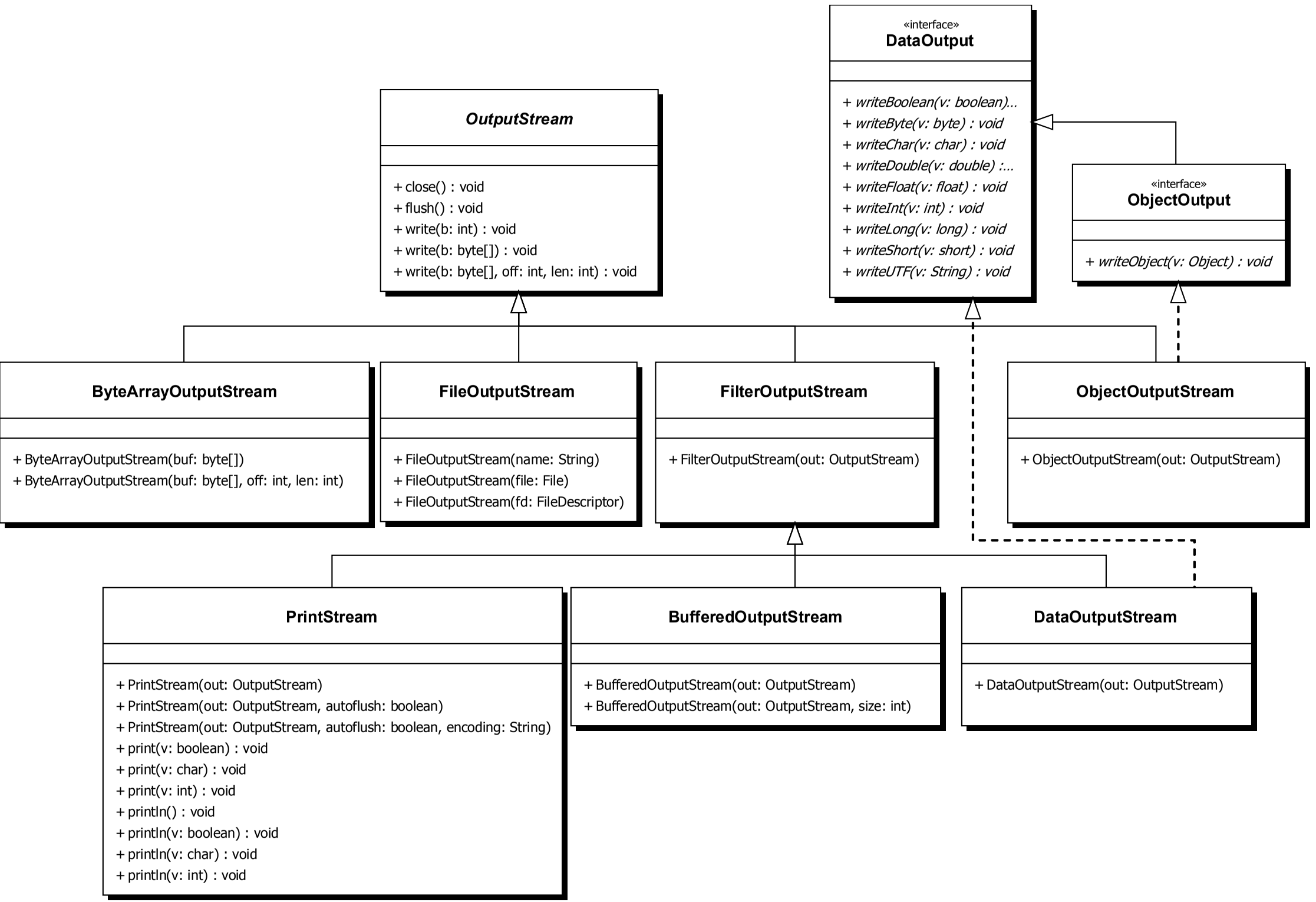
Schiffe		
SID	Name	**MID**

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.