universität ulm

1. Klausur zur Veranstaltung

Objektorientierte Programmierung

im Sommersemester 2023

Prüfer: Prof. Dr. Matthias Tichy Fakultät Ingenieurwissenschaften, Informatik, Psychologie

07.08.2023, 9 Uhr **Bearbeitungszeit: 90 min**

Nachname:	Vorname:	Matrikelnummer:
Studiengang und Abschluss:		Fachsemester:

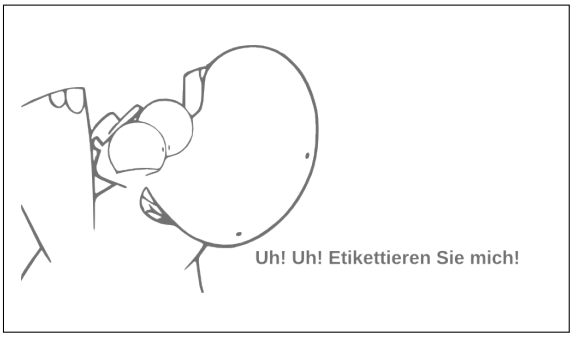
Hiermit erkläre ich, dass ich prüfungsfähig bin.
Sollte ich nicht auf der Liste der angemeldeten Studierenden aufgeführt sein, dann nehme ich hiermit zur Kenntnis, dass diese Prüfung nicht gewertet werden wird.

_____

Datum, Unterschrift des Prüfungsteilnehmers

Zur allgemeinen Beachtung:

- Füllen Sie das Deckblatt vollständig und korrekt aus.
- Lesen Sie sich zunächst die Klausur sorgfältig durch (die Aufgaben sind auf 17 Seiten mit 2 Seiten pro Blatt verteilt).
- Bearbeiten Sie die Aufgaben direkt auf den Aufgabenblättern.
- 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.

Punkteverteilung								
1	2	3	4	5	6	7	$\sum$	**Note**
von 12	von 14	von 14	von 9	von 14	von 6	von 7	von 76	
								Korrektur

Einsichtnahme ohne Nachkorrektur ◯ Einsichtnahme mit Nachkorrektur ◯

Aufgabe 1 - Wissensfragen **12 Punkte**

Kreuzen Sie zu jeder Frage die korrekte Antwortmöglichkeit an. Zu jeder Frage existiert nur eine korrekte Aussage.

a) Gegeben eine Variable a eines beliebigen Datentyps in Java. Welche der folgenden Aussagen über den folgenden Ausdruck ist korrekt?
Ausdruck: `(Object) a`

- ☐ Der Aufruf führt immer zu einer `ClassCastException`.
- ☐ Der Aufruf ist immer korrekt.
- ☐ Der Aufruf funktioniert nur auf Strings.
- ☐ Der Aufruf funktioniert nicht auf primitiven Datentypen.
- ☐ Der Aufruf funktioniert nur auf primitiven Datentypen.

b) Bei dem Methodenaufruf der Methode `exercise1(new int[] {5,6}, 3, "Hello")` in Java

- ☐ werden Referenzen auf alle Parameter übergeben.
- ☐ werden Kopien aller Werte übergeben.
- ☐ werden Kopien des 1. und 3. Parameter und eine Referenz auf den 2. übergeben.
- ☐ werden Kopien der 1. und 2. Parameter und eine Referenz auf den 3. übergeben.
- ☐ werden Referenzen auf die 1. und 3. Parameter und eine Kopie des 2. Parameter übergeben.

c) Gegeben eine Methode `int doStuff(int i)`. Welches der folgenden return-Statements ist zur Compilezeit **nicht** erlaubt?

- ☐ `return i;`
- ☐ `return --i;`
- ☐ `return (Integer) null;`
- ☐ `return "3";`
- ☐ `return 5;`

d) Welche Größe in bits hat der Datentyp `double` in Java?

- ☐ 1 bit
- ☐ 8 bit
- ☐ 16 bit
- ☐ 32 bit
- ☐ 64 bit

e) Gegeben folgenden Java Code. Welche konkrete Methodenimplementierung wird beim Aufruf in Zeile 8 verwendet?

```
1 class A {int dyn() {...}}
2 class B extends A {int dyn() {...}}
3 class C {int dyn() {...}}
4 class D extends B {int dyn() {...}}
5 //...
6 public static void main(String... args) {
7     A x = (B) new D();
8     x.dyn()
9 }
```

- ☐ Die der Klasse `A`.
- ☐ Die der Klasse `B`.
- ☐ Die der Klasse `C`.
- ☐ Die der Klasse `D`.
- ☐ Keine. Der Aufruf führt zu einem `RuntimeError`.

f) Gegeben folgenden Java Code. Welche Kombination an Aufrufen und Zugriffen ausgeführt zwischen Zeilen 10-12 ist korrekt?

```
1  public class E {
2      private int x;
3      public String doThings() {...}
4      public double y;
5  }
6  //...
7  public class F extends E {
8      public static int doStuff() {...}
9      private boolean z;
10     public char exercise(F f) {
11         // 1. Befehl
12         // 2. Befehl
13     }
14 }
```

- ☐ Zeile 11: `var u = f.z;`
  Zeile 12: `doThings();`
- ☐ Zeile 11: `var u = f.y;`
  Zeile 12: `E.doStuff();`
- ☐ Zeile 11: `var u = f.y;`
  Zeile 12: `E.doThings();`
- ☐ Zeile 11: `var u = f.x;`
  Zeile 12: `doStuff();`
- ☐ Zeile 11: `var u = f.x;`
  Zeile 12: `var w = f.y;`

Aufgabe 2 - Imperative & Objektorientierte Programmierung 5 + 3 + 3 + 3 = 14 Punkte

a) Betrachten Sie folgende Klassen welche einen Binärbaum mit rekursiv implementierter binärer Suche umsetzen. Implementieren Sie die Methode `findL(...)` der Klasse `Tree`, welche ebenfalls binäre Suche im Baum umsetzt. Jedoch **ohne** Rekursion, nur mit Hilfe von Schleifen.

```
1  public class Tree {
2      TreeElement root;
3      public TreeElement find(int value) {
4          if (root == null) {return null;}
5          if(root.value == value) {return root;}
6          else {return root.find(value);}
7      }
8  }
```

```
1  public class TreeElement {
2      int value;
3      TreeElement left,right;
4      public TreeElement find(int value) {
5          if(this.value == value) {
6              return this;
7          } else if (value <= this.value) {
8              return left==null?null:left.find(value);
9          } else {
10             return right==null?null:right.find(value);
11         }
12     }
13 }
```

```
public TreeElement findL(int value) {
```

b) Betrachten Sie den folgende Quellcode und beantworten Sie die Fragen.

```
1  public class Ex2b{
2      public static void main(String[] args){
3          int[][] matrix = {{1,2},{3,4,5}};
4          bar(matrix)
5          System.out.println(matrix[1][0]);
6          foo(matrix);
7      }
8      public static void bar(int[][] matrix) {
9          matrix[1][0] = 10;
10         matrix[1] = matrix[0];
11     }
12     public static void foo(int[][] matrix) {
13         System.out.println(matrix[0][3]);
14     }
15 }
```

i. Was was wird in Zeile 5 ausgegeben?

- ☐ Die Zahl 10 wird ausgegeben.
- ☐ Die Zahl 1 wird ausgegeben
- ☐ Die Zahl 3 wird ausgegeben
- ☐ `null` wird ausgegeben

ii. Was geschieht beim Aufruf der Methode `foo`?

- ☐ Ein `RuntimeError`
- ☐ Die Zahl 2 wird ausgegeben
- ☐ Die Zahl 3 wird ausgegeben
- ☐ `null` wird ausgegeben

c) Beschreiben Sie, in ganzen Sätzen, 2 Unterschiede und eine Gemeinsamkeit zwischen einer `abstrakten Klasse` und einem `Interface` in Java.

d) Beschreiben Sie, in ganzen Sätzen, was man unter dem Konzept *information hiding* bzw. *Datenkapselung* versteht, warum es angewendet wird und mit welchen Syntaxkonstrukten dies in Java umsetzt wird.

Diese Seite wurde für ein besseres Layout leer gelassen.

Aufgabe 3 - OOP

4 + 8 + 2 = 14 Punkte

Betrachten Sie folgende Klassen:

```java
record Tuple(String BookTitle, Page Page) {};

public class Library {
    //Maps Genretype to Collection of books of that genre
    private Map<String, Collection<Book>> books;
}
```

```java
public class Book {
    public String title;
    protected List<Page> pages;
}
```

```java
public class Page {
    public String content;
}
```

a) Implementieren Sie die Methode `insertBook(...)` der Klasse `Library`, welche ein Buch mit übergebenem Genre in die Menge der Bücher zu diesem Genre hinzufügt. Achten Sie dabei darauf, dass Buchtitel pro Genre eindeutig sein müssen. Sollte ein Buch mit selbem Namen bereits existieren soll dieses mit dem übergebenen Buch ersetzt werden.

```java
public void insertBook(Book book, String genre) {
```

b) Implementieren Sie die Methode `findPagesContent(...)` welche alle Bücher herausfiltert die eine Page enthalten deren Inhalt den übergebenen String beinhaltet. Das Rückgabeformat soll dabei eine `Collection` von `Tuple` sein, so dass für jedes gefundene Buch ein Tuple mit dem Titel des Buchs und der ersten Seite die den übergebenen String beinhaltet, in der Collection enthalten ist. Verwenden Sie zur Lösung dieser Aufgabe keine Schleifen sondern ausschließlich Methoden aus der Java Collection Streams-API.

```
public Collection<Tuple> findPagesContent(String needle) {
```

c) Beschreiben Sie wie in Java Lambda Ausdrücke mit Hilfe von OOP Konzepten technisch umgesetzt sind.

Aufgabe 4 - Decorator-Pattern und JavaIO

7 + 2 = 9 Punkte

Betrachten Sie das Interface `Collection<E>` und die implementierende Klasse `LinkedList<E>`:

```
1 public interface Collection<E> {
2     boolean add(E e);
3     boolean remove(Object o);
4     // weitere Methoden ...
5 }
6 public class LinkedList<E> implements Collection<E> { //... }
```

a) Implementieren Sie eine Klasse `FilteredCollectionDecorator<E>` auf Basis des Interfaces `Collection<E>`. Verwenden Sie hierfür das **Decorator-Pattern**. Ein `FilteredCollectionDecorator<E>` soll beim Hinzufügen eines Elements durch Aufruf der Methode `boolean add(E e)` mittels eines Filters prüfen, ob das Element hinzugefügt werden soll (dann Rückgabewert true) oder nicht (dann Rückgabewert false). Der Filter soll durch einen Lambda-Ausdruck vom Typ `Predicate<T>` angegeben werden können. Die übrigen Methoden (spezifisch hier nur die Methode `boolean remove(Object o)`) sollen kein geändertes Verhalten zeigen. Achten Sie darauf, dass die durch das Interface definierten Signaturen der Methoden nicht verändert werden.

```
public class FilterCollectionDecorator<E>
```

b) 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.

```
public static void printIfExist(String[] paths) throws IOException {
```

Aufgabe 5 - XML, JSON, Build Tools

8 + 3 + 3 = 14 Punkte

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

```
<?xml version="1.0" encoding="UTF-8"?>
<album name="Great Debut">
    <songs>7</songs>
    <band>
        <artist>Mr. X</artist>
        <artist>Ms. X</artist>
    </band>
</album>
```

a) Implementieren Sie die Methode *change(...)* (aus dem gegebenen Code-Ausschnitt), welche (1) den Namen des Albums in *music.xml* in „Good Debut“ ändert, (2) die Anzahl der Lieder von 7 zu 8 ändert und (3) ein neues Bandmitglied (*artist*) mit dem Namen „The X” hinzufügt.
Hinweis: Sie können davon ausgehen, dass die notwendigen Klassen importiert sind (*Document* ist *org.w3c.dom.Document*).
Hinweis: Beachten Sie hierfür auch die Teile *Element*, *NodeList*, *Node* und *Document* des **CheatSheets** am Ende der Klausur.
Hinweis: *Element* und *Document* sind Subinterfaces von *Node*.

```
DocumentBuilder parser = DocumentBuilderFactory.newInstance()
        .newDocumentBuilder();
Document doc = parser.parse("music.xml");
change(doc);
Transformer trans = TransformerFactory.newInstance().newTransformer();
trans.transform(new DOMSource(doc), new StreamResult(
        "music_changed.xml"));
```

```
public static void change(Document doc) {
```

b) Definieren Sie ein JSON Dokument, welches äquivalent zum gegebenen XML Dokument ist (die Manipulation in Teilaufgabe a) wird dabei nicht beachtet).

c) Nennen und beschreiben Sie **3** Vorteile, die sich durch die Verwendung eines Build Automation Tools wie z.B Gradle ergeben.

Aufgabe 6 - GUI

2 + 4 = 6 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 eine Anwendung des Composite-Patterns zeigen, und erklären Sie, in ganzen Sätzen, auf Basis dieser Stellen, was dieses Pattern ist. Nennen und erläutern Sie 1 Vorteil des Composite-Patterns.

b) 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.

```
public void start(Stage primaryStage) {
var sp = new ScrollPane();
var button = new Button();
```

Aufgabe 7 - Threads **7 Punkte**

a) In dieser Aufgabe sollen Sie eine Näherung von Pi mittels der Bailey-Borwein-Plouffe Formel berechnen. Diese ist als Blackbox in der Funktion *BBPFormula(int n)* gegeben. Pi ist gleich der unendlichen Summe (von n = 0 bis unendlich) über diese Formel. Ergänzen Sie den folgenden Code um 50 (gespeichert in *amount*) Summanden der Summe parallel zu berechnen. Dabei soll das Ergebnis parallel in der *DoubleAdder* Variable *acc* gespeichert werden. Der *AtomicInteger amount* gibt an wieviele Summanden noch berechnet werden müssen, dabei soll jeder Thread iterativ den aktuell höchsten noch nicht berechneten Summanden berechnen (*amount* dekrementieren) und so lange laufen bis insgesamt *amount* viele Summanden berechnet wurden. Verwenden Sie zur Synchronisation der Threads nicht das *synchronized* Keyword, sondern die Methoden der Klassen *DoubleAdder* und *AtomicInteger*.
Hinweis: Beachten Sie hierfür auch die Teile *Executor*, *DoubleAdder* und *AtomicInteger* des **CheatSheets** am Ende der Klausur.
Hinweis: *ExecutorService* ist ein Subinterface von *Executor*.

```
1  private static DoubleAdder acc = new DoubleAdder();
2  private static AtomicInteger amount = new AtomicInteger(50);

3  public static double BBPFormula(int n) {
4      // Blackbox
5  }

6  public static double calculatePi() {

7      int threads = Runtime.getRuntime().availableProcessors();
8      ExecutorService executorService = Executors
9                  .newFixedThreadPool(threads);

10     for (int i = 0; i < threads; i++)
11     {
12         executorService.execute(() -> {
13             // Zu ergänzen auf der nächsten Seite
14         });
15     }
16     try {
17         executorService.shutdown();
18         executorService.awaitTermination(Integer.MAX_VALUE,
19                     TimeUnit.SECONDS);
20     }
21     catch (InterruptedException ex) {
22         ex.printStackTrace();
23     }
24     return acc.sum();
25 }
```

```
executorService.execute(() -> {

});
```

music.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<album name="Great Debut">
    <songs>7</songs>
    <band>
        <artist>Mr. X</artist>
        <artist>Ms. X</artist>
    </band>
</album>
```

Collection<T>

Modifier and Type	Method and Description
boolean	**add**(E e) Ensures that this collection contains the specified element (optional operation).
boolean	**addAll**(Collection<? extends E> c) Adds all of the elements in the specified collection to this collection (optional operation).
void	**clear**() Removes all of the elements from this collection (optional operation).
boolean	**contains**(Object o) Returns true if this collection contains the specified element.
boolean	**containsAll**(Collection<?> c) Returns true if this collection contains all of the elements in the specified collection.
boolean	**equals**(Object o) Compares the specified object with this collection for equality.
int	**hashCode**() Returns the hash code value for this collection.
boolean	**isEmpty**() Returns true if this collection contains no elements.
Iterator<E>	**iterator**() Returns an iterator over the elements in this collection.
default Stream<E>	**parallelStream**() Returns a possibly parallel Stream with this collection as its source.
boolean	**remove**(Object o) Removes a single instance of the specified element from this collection, if it is present (optional operation).
boolean	**removeAll**(Collection<?> c) Removes all of this collection's elements that are also contained in the specified collection (optional operation).
default boolean	**removeIf**(Predicate<? super E> filter) Removes all of the elements of this collection that satisfy the given predicate.
boolean	**retainAll**(Collection<?> c) Retains only the elements in this collection that are contained in the specified collection (optional operation).
int	**size**() Returns the number of elements in this collection.
default Spliterator<E>	**spliterator**() Creates a **Spliterator** over the elements in this collection.
default Stream<E>	**stream**() Returns a sequential Stream with this collection as its source.
Object[]	**toArray**() Returns an array containing all of the elements in this collection.
<T> T[]	**toArray**(T[] a) Returns an array containing all of the elements in this collection; the runtime type of the returned array is that of the specified array.

Collectors

`static <T> Collector<T,?,List<T>>`	**`toList()`** Returns a `Collector` that accumulates the input elements into a new `List`.
`static <T,K,U> Collector<T,?,Map<K,U>>`	**`toMap(Function<? super T,? extends K> keyMapper, Function<? super T,? extends U> valueMapper)`** Returns a `Collector` that accumulates elements into a `Map` whose keys and values are the result of applying the provided mapping functions to the input elements.
`static <T,K,U> Collector<T,?,Map<K,U>>`	**`toMap(Function<? super T,? extends K> keyMapper, Function<? super T,? extends U> valueMapper, BinaryOperator<U> mergeFunction)`** Returns a `Collector` that accumulates elements into a `Map` whose keys and values are the result of applying the provided mapping functions to the input elements.
`static <T,K,U,M extends Map<K,U>> Collector<T,?,M>`	**`toMap(Function<? super T,? extends K> keyMapper, Function<? super T,? extends U> valueMapper, BinaryOperator<U> mergeFunction, Supplier<M> mapSupplier)`** Returns a `Collector` that accumulates elements into a `Map` whose keys and values are the result of applying the provided mapping functions to the input elements.
`static <T> Collector<T,?,Set<T>>`	**`toSet()`** Returns a `Collector` that accumulates the input elements into a new `Set`.

HashMap

Modifier and Type	Method and Description
void	**clear**() Removes all of the mappings from this map.
Object	**clone**() Returns a shallow copy of this HashMap instance: the keys and values themselves are not cloned.
V	**compute**(**K** key, **BiFunction**<? super **K**,? super **V**,? extends **V**> remappingFunction) Attempts to compute a mapping for the specified key and its current mapped value (or null if there is no current mapping).
V	**computeIfAbsent**(**K** key, **Function**<? super **K**,? extends **V**> mappingFunction) If the specified key is not already associated with a value (or is mapped to null), attempts to compute its value using the given mapping function and enters it into this map unless null.
V	**computeIfPresent**(**K** key, **BiFunction**<? super **K**,? super **V**,? extends **V**> remappingFunction) If the value for the specified key is present and non-null, attempts to compute a new mapping given the key and its current mapped value.
boolean	**containsKey**(**Object** key) Returns true if this map contains a mapping for the specified key.
boolean	**containsValue**(**Object** value) Returns true if this map maps one or more keys to the specified value.
Set<**Map.Entry**<**K**,**V**>>	**entrySet**() Returns a **Set** view of the mappings contained in this map.
void	**forEach**(**BiConsumer**<? super **K**,? super **V**> action) Performs the given action for each entry in this map until all entries have been processed or the action throws an exception.
V	**get**(**Object** key) Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.
V	**getOrDefault**(**Object** key, **V** defaultValue) Returns the value to which the specified key is mapped, or defaultValue if this map contains no mapping for the key.
boolean	**isEmpty**() Returns true if this map contains no key-value mappings.
Set<**K**>	**keySet**() Returns a **Set** view of the keys contained in this map.
V	**merge**(**K** key, **V** value, **BiFunction**<? super **V**,? super **V**,? extends **V**> remappingFunction) If the specified key is not already associated with a value or is associated with null, associates it with the given non-null value.
V	**put**(**K** key, **V** value) Associates the specified value with the specified key in this map.
void	**putAll**(**Map**<? extends **K**,? extends **V**> m) Copies all of the mappings from the specified map to this map.
V	**putIfAbsent**(**K** key, **V** value) If the specified key is not already associated with a value (or is mapped to null) associates it with the given value and returns null, else returns the current value.
V	**remove**(**Object** key) Removes the mapping for the specified key from this map if present.
boolean	**remove**(**Object** key, **Object** value) Removes the entry for the specified key only if it is currently mapped to the specified value.
V	**replace**(**K** key, **V** value) Replaces the entry for the specified key only if it is currently mapped to some value.
boolean	**replace**(**K** key, **V** oldValue, **V** newValue) Replaces the entry for the specified key only if currently mapped to the specified value.
void	**replaceAll**(**BiFunction**<? super **K**,? super **V**,? extends **V**> function) Replaces each entry's value with the result of invoking the given function on that entry until all entries have been processed or the function throws an exception.
int	**size**() Returns the number of key-value mappings in this map.
Collection<**V**>	**values**() Returns a **Collection** view of the values contained in this map.

Stream<T> Interface

Modifier and Type	Method and Description
boolean	**allMatch**(**Predicate**<? super T> predicate) Returns whether all elements of this stream match the provided predicate.
boolean	**anyMatch**(**Predicate**<? super T> predicate) Returns whether any elements of this stream match the provided predicate.
static <T> **Stream.Builder**<T>	**builder**() Returns a builder for a Stream.
<R,A> R	**collect**(**Collector**<? super T,A,R> collector) Performs a **mutable reduction** operation on the elements of this stream using a Collector.
<R> R	**collect**(**Supplier**<R> supplier, **BiConsumer**<R,? super T> accumulator, **BiConsumer**<R,R> combiner) Performs a **mutable reduction** operation on the elements of this stream.
static <T> **Stream**<T>	**concat**(**Stream**<? extends T> a, **Stream**<? extends T> b) Creates a lazily concatenated stream whose elements are all the elements of the first stream followed by all the elements of the second stream.
long	**count**() Returns the count of elements in this stream.
Stream<T>	**distinct**() Returns a stream consisting of the distinct elements (according to **Object.equals(Object)**) of this stream.
static <T> **Stream**<T>	**empty**() Returns an empty sequential Stream.
Stream<T>	**filter**(**Predicate**<? super T> predicate) Returns a stream consisting of the elements of this stream that match the given predicate.
Optional<T>	**findAny**() Returns an **Optional** describing some element of the stream, or an empty Optional if the stream is empty.
Optional<T>	**findFirst**() Returns an **Optional** describing the first element of this stream, or an empty Optional if the stream is empty.
<R> **Stream**<R>	**flatMap**(**Function**<? super T,? extends **Stream**<? extends R>> mapper) Returns a stream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping function to each element.
DoubleStream	**flatMapToDouble**(**Function**<? super T,? extends **DoubleStream**> mapper) Returns an DoubleStream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping function to each element.
IntStream	**flatMapToInt**(**Function**<? super T,? extends **IntStream**> mapper) Returns an IntStream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping function to each element.
LongStream	**flatMapToLong**(**Function**<? super T,? extends **LongStream**> mapper) Returns an LongStream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping function to each element.
void	**forEach**(**Consumer**<? super T> action) Performs an action for each element of this stream.
void	**forEachOrdered**(**Consumer**<? super T> action) Performs an action for each element of this stream, in the encounter order of the stream if the stream has a defined encounter order.
static <T> **Stream**<T>	**generate**(**Supplier**<T> s) Returns an infinite sequential unordered stream where each element is generated by the provided Supplier.
static <T> **Stream**<T>	**iterate**(T seed, **UnaryOperator**<T> f) Returns an infinite sequential ordered Stream produced by iterative application of a function f to an initial element seed, producing a Stream consisting of seed, f(seed), f(f(seed)), etc.
Stream<T>	**limit**(long maxSize) Returns a stream consisting of the elements of this stream, truncated to be no longer than maxSize in length.
<R> **Stream**<R>	**map**(**Function**<? super T,? extends R> mapper) Returns a stream consisting of the results of applying the given function to the elements of this stream.
DoubleStream	**mapToDouble**(**ToDoubleFunction**<? super T> mapper) Returns a DoubleStream consisting of the results of applying the given function to the elements of this stream.
IntStream	**mapToInt**(**ToIntFunction**<? super T> mapper) Returns an IntStream consisting of the results of applying the given function to the elements of this stream.

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.

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.

org.w3c.dom.Element

Modifier and Type	Method and Description
String	getAttribute(String name) Retrieves an attribute value by name.
Attr	getAttributeNode(String name) Retrieves an attribute node by name.
Attr	getAttributeNodeNS(String namespaceURI, String localName) Retrieves an Attr node by local name and namespace URI.
String	getAttributeNS(String namespaceURI, String localName) Retrieves an attribute value by local name and namespace URI.
NodeList	getElementsByTagName(String name) Returns a NodeList of all descendant Elements with a given tag name, in document order.
NodeList	getElementsByTagNameNS(String namespaceURI, String localName) Returns a NodeList of all the descendant Elements with a given local name and namespace URI in document order.
TypeInfo	getSchemaTypeInfo() The type information associated with this element.
String	getTagName() The name of the element.
boolean	hasAttribute(String name) Returns true when an attribute with a given name is specified on this element or has a default value, false otherwise.
boolean	hasAttributeNS(String namespaceURI, String localName) Returns true when an attribute with a given local name and namespace URI is specified on this element or has a default value, false otherwise.
void	removeAttribute(String name) Removes an attribute by name.
Attr	removeAttributeNode(Attr oldAttr) Removes the specified attribute node.
void	removeAttributeNS(String namespaceURI, String localName) Removes an attribute by local name and namespace URI.
void	setAttribute(String name, String value) Adds a new attribute.
Attr	setAttributeNode(Attr newAttr) Adds a new attribute node.
Attr	setAttributeNodeNS(Attr newAttr) Adds a new attribute.

org.w3c.dom.NodeList

Modifier and Type	Method and Description
int	getLength() The number of nodes in the list.
Node	item(int index) Returns the indexth item in the collection.

org.w3c.dom.Node

Modifier and Type	Method and Description
Node	appendChild(Node newChild) Adds the node newChild to the end of the list of children of this node.
Node	cloneNode(boolean deep) Returns a duplicate of this node, i.e., serves as a generic copy constructor for nodes.
short	compareDocumentPosition(Node other) Compares the reference node, i.e.
NamedNodeMap	getAttributes() A NamedNodeMap containing the attributes of this node (if it is an Element) or null otherwise.
String	getBaseURI() The absolute base URI of this node or null if the implementation wasn't able to obtain an absolute URI.
NodeList	getChildNodes() A NodeList that contains all children of this node.
Object	getFeature(String feature, String version) This method returns a specialized object which implements the specialized APIs of the specified feature and version, as specified in .
Node	getFirstChild() The first child of this node.
Node	getLastChild() The last child of this node.
String	getLocalName() Returns the local part of the qualified name of this node.
String	getNamespaceURI() The namespace URI of this node, or null if it is unspecified (see).
Node	getNextSibling() The node immediately following this node.
String	getNodeName() The name of this node, depending on its type; see the table above.
short	getNodeType() A code representing the type of the underlying object, as defined above.
String	getNodeValue() The value of this node, depending on its type; see the table above.
Document	getOwnerDocument() The Document object associated with this node.
Node	getParentNode() The parent of this node.
String	getPrefix() The namespace prefix of this node, or null if it is unspecified.

org.w3c.dom.Node (Fortsetzung)

String	**getPrefix**() The namespace prefix of this node, or null if it is unspecified.
Node	**getPreviousSibling**() The node immediately preceding this node.
String	**getTextContent**() This attribute returns the text content of this node and its descendants.
Object	**getUserData**(String key) Retrieves the object associated to a key on a this node.
boolean	**hasAttributes**() Returns whether this node (if it is an element) has any attributes.
boolean	**hasChildNodes**() Returns whether this node has any children.
Node	**insertBefore**(Node newChild, Node refChild) Inserts the node newChild before the existing child node refChild.
boolean	**isDefaultNamespace**(String namespaceURI) This method checks if the specified namespaceURI is the default namespace or not.
boolean	**isEqualNode**(Node arg) Tests whether two nodes are equal.
boolean	**isSameNode**(Node other) Returns whether this node is the same node as the given one.
boolean	**isSupported**(String feature, String version) Tests whether the DOM implementation implements a specific feature and that feature is supported by this node, as specified in .
String	**lookupNamespaceURI**(String prefix) Look up the namespace URI associated to the given prefix, starting from this node.
String	**lookupPrefix**(String namespaceURI) Look up the prefix associated to the given namespace URI, starting from this node.
void	**normalize**() Puts all Text nodes in the full depth of the sub-tree underneath this Node, including attribute nodes, into a "normal" form where only structure (e.g., elements, comments, processing instructions, CDATA sections, and entity references) separates Text nodes, i.e., there are neither adjacent Text nodes nor empty Text nodes.
Node	**removeChild**(Node oldChild) Removes the child node indicated by oldChild from the list of children, and returns it.
Node	**replaceChild**(Node newChild, Node oldChild) Replaces the child node oldChild with newChild in the list of children, and returns the oldChild node.
void	**setNodeValue**(String nodeValue) The value of this node, depending on its type; see the table above.
void	**setPrefix**(String prefix) The namespace prefix of this node, or null if it is unspecified.
void	**setTextContent**(String textContent) This attribute returns the text content of this node and its descendants.
Object	**setUserData**(String key, Object data, UserDataHandler handler) Associate an object to a key on this node.

org.w3c.dom.Document

Modifier and Type	Method and Description
`Node`	`adoptNode(Node source)` Attempts to adopt a node from another document to this document.
`Attr`	`createAttribute(String name)` Creates an `Attr` of the given name.
`Attr`	`createAttributeNS(String namespaceURI, String qualifiedName)` Creates an attribute of the given qualified name and namespace URI.
`CDATASection`	`createCDATASection(String data)` Creates a `CDATASection` node whose value is the specified string.
`Comment`	`createComment(String data)` Creates a `Comment` node given the specified string.
`DocumentFragment`	`createDocumentFragment()` Creates an empty `DocumentFragment` object.
`Element`	`createElement(String tagName)` Creates an element of the type specified.
`Element`	`createElementNS(String namespaceURI, String qualifiedName)` Creates an element of the given qualified name and namespace URI.
`EntityReference`	`createEntityReference(String name)` Creates an `EntityReference` object.
`ProcessingInstruction`	`createProcessingInstruction(String target, String data)` Creates a `ProcessingInstruction` node given the specified name and data strings.
`Text`	`createTextNode(String data)` Creates a `Text` node given the specified string.
`DocumentType`	`getDoctype()` The Document Type Declaration (see `DocumentType`) associated with this document.
`Element`	`getDocumentElement()` This is a convenience attribute that allows direct access to the child node that is the document element of the document.
`String`	`getDocumentURI()` The location of the document or `null` if undefined or if the `Document` was created using `DOMImplementation.createDocument`.
`DOMConfiguration`	`getDomConfig()` The configuration used when `Document.normalizeDocument()` is invoked.
`Element`	`getElementById(String elementId)` Returns the `Element` that has an ID attribute with the given value.
`NodeList`	`getElementsByTagName(String tagname)` Returns a `NodeList` of all the `Elements` in document order with a given tag name and are contained in the document.

java.util.concurrent.Executor

Modifier and Type	Method and Description
`void`	**`execute`**`(`**`Runnable`** `command)` Executes the given command at some time in the future.

java.util.concurrent.ExecutorService

Modifier and Type	Method and Description
boolean	**awaitTermination**(long timeout, **TimeUnit** unit) Blocks until all tasks have completed execution after a shutdown request, or the timeout occurs, or the current thread is interrupted, whichever happens first.
<T> **List**<**Future**<T>>	**invokeAll**(**Collection**<? extends **Callable**<T>> tasks) Executes the given tasks, returning a list of Futures holding their status and results when all complete.
<T> **List**<**Future**<T>>	**invokeAll**(**Collection**<? extends **Callable**<T>> tasks, long timeout, **TimeUnit** unit) Executes the given tasks, returning a list of Futures holding their status and results when all complete or the timeout expires, whichever happens first.
<T> T	**invokeAny**(**Collection**<? extends **Callable**<T>> tasks) Executes the given tasks, returning the result of one that has completed successfully (i.e., without throwing an exception), if any do.
<T> T	**invokeAny**(**Collection**<? extends **Callable**<T>> tasks, long timeout, **TimeUnit** unit) Executes the given tasks, returning the result of one that has completed successfully (i.e., without throwing an exception), if any do before the given timeout elapses.
boolean	**isShutdown**() Returns `true` if this executor has been shut down.
boolean	**isTerminated**() Returns `true` if all tasks have completed following shut down.
void	**shutdown**() Initiates an orderly shutdown in which previously submitted tasks are executed, but no new tasks will be accepted.
List<**Runnable**>	**shutdownNow**() Attempts to stop all actively executing tasks, halts the processing of waiting tasks, and returns a list of the tasks that were awaiting execution.
<T> **Future**<T>	**submit**(**Callable**<T> task) Submits a value-returning task for execution and returns a Future representing the pending results of the task.
Future<?>	**submit**(**Runnable** task) Submits a Runnable task for execution and returns a Future representing that task.
<T> **Future**<T>	**submit**(**Runnable** task, T result) Submits a Runnable task for execution and returns a Future representing that task.

java.util.concurrent.atomic.DoubleAdder

Modifier and Type	Method and Description
void	**add**(double x) Adds the given value.
double	**doubleValue**() Equivalent to **sum()**.
float	**floatValue**() Returns the **sum()** as a float after a narrowing primitive conversion.
int	**intValue**() Returns the **sum()** as an int after a narrowing primitive conversion.
long	**longValue**() Returns the **sum()** as a long after a narrowing primitive conversion.
void	**reset**() Resets variables maintaining the sum to zero.
double	**sum**() Returns the current sum.
double	**sumThenReset**() Equivalent in effect to **sum()** followed by **reset()**.
String	**toString**() Returns the String representation of the **sum()**.

java.util.concurrent.atomic.AtomicInteger

Modifier and Type	Method and Description
int	**accumulateAndGet**(int x, IntBinaryOperator accumulatorFunction) Atomically updates the current value with the results of applying the given function to the current and given values, returning the updated value.
int	**addAndGet**(int delta) Atomically adds the given value to the current value.
boolean	**compareAndSet**(int expect, int update) Atomically sets the value to the given updated value if the current value == the expected value.
int	**decrementAndGet**() Atomically decrements by one the current value.
double	**doubleValue**() Returns the value of this AtomicInteger as a double after a widening primitive conversion.
float	**floatValue**() Returns the value of this AtomicInteger as a float after a widening primitive conversion.
int	**get**() Gets the current value.
int	**getAndAccumulate**(int x, IntBinaryOperator accumulatorFunction) Atomically updates the current value with the results of applying the given function to the current and given values, returning the previous value.
int	**getAndAdd**(int delta) Atomically adds the given value to the current value.
int	**getAndDecrement**() Atomically decrements by one the current value.
int	**getAndIncrement**() Atomically increments by one the current value.
int	**getAndSet**(int newValue) Atomically sets to the given value and returns the old value.
int	**getAndUpdate**(IntUnaryOperator updateFunction) Atomically updates the current value with the results of applying the given function, returning the previous value.
int	**incrementAndGet**() Atomically increments by one the current value.
int	**intValue**() Returns the value of this AtomicInteger as an int.
void	**lazySet**(int newValue) Eventually sets to the given value.
long	**longValue**() Returns the value of this AtomicInteger as a long after a widening primitive conversion.
void	**set**(int newValue) Sets to the given value.
String	**toString**() Returns the String representation of the current value.
int	**updateAndGet**(IntUnaryOperator updateFunction) Atomically updates the current value with the results of applying the given function, returning the updated value.
boolean	**weakCompareAndSet**(int expect, int update) Atomically sets the value to the given updated value if the current value == the expected value.