Chapter 1: Strings, Numbers, and Math

- **Hello**
  - lower-case
  - remove whitespaces

- **oehlll**

- **Sunday**
- **Monday**
- **Tuesday**
- **Wednesday**
- **Thursday**
- **Friday**
- **Saturday**
<html>
  <body>
    <h2>Hello world!</h2>
  </body>
</html>
<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>AND</th>
<th>OR</th>
<th>XOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Chapter 2: Objects, Immutability, and Switch Expressions

requireNonNullElseThrow(T obj, X exception)  T
requireNonNullElseThrowIAE(T obj, String message)  T
requireNonNullElseThrowIAE(T obj, Supplier<String> messageSupplier)  T
requireNotNullElseThrow(T obj, Supplier<? extends X> exceptionSupplier)  T


SCP  SCP


SCP  SCP

deepClone(T o)  T
depClonelnstance(T o, Object... dontCloneThese)  T
fastCloneOrnewInstance(Class<T> c)  T
shallowClone(T o)  T
copyPropertiesOfInheritedClass(T src, E dest)  void
dontClone(Class<?>... c)  void
dontCloneInstanceOf(Class<?>... c)  void
equals(Object obj)  boolean
class()  Class<?>
getDumpCloned()  IDumpCloned
...
```java
public Player createPlayer(PlayerTypes playerType) {
    return switch (playerType) {
        case TENNIS ->
            new TennisPlayer();
    }
}
```
Chapter 3: Working with Date and Time

<table>
<thead>
<tr>
<th>Time Zone</th>
<th>UTC Time</th>
<th>Local Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perth, Australia</td>
<td>8:00a - 11:30p</td>
<td>1:00p - 4:30p</td>
</tr>
<tr>
<td>Bucharest, Romania</td>
<td>10:00a - 1:30p</td>
<td>5:00a - 8:30p</td>
</tr>
</tbody>
</table>

```java
DEFAULT_TIME_ZONE

- dateToInstant(Date date) -> Instant
- dateToLocalDate(Date date) -> LocalDate
- dateToLocalDateTime(Date date) -> LocalDateTime
- dateToLocalTime(Date date) -> LocalTime
- dateToOffsetDateTime(Date date) -> OffsetDateTime
- dateToOffsetTime(Date date) -> OffsetTime
- dateToZonedDateTime(Date date) -> ZonedDateTime
- instantToDate(Instant instant) -> Date
- localDateTimeToDate(LocalDateTime localDateTime) -> Date
- localDateToDate(LocalDate localDate) -> Date
- localTimeToDate(LocalTime localTime) -> Date
- offsetDateTimeToDate(OffsetDateTime offsetDateTime) -> Date
- offsetTimeToDate(OffsetTime offsetTime) -> Date
- zonedDateTimeToDate(ZonedDateTime zonedDateTime) -> Date
```

```
Chapter 4: Type Inference

Decompilation of the class containing this method:

```java
public Object fetchTransferableData(String data) throws UnsupportedFlavorException, IOException {
    StringSelection ss = new StringSelection(data);
    DataFlavor[] df = ss.getTransferDataFlavors();
    Object obj = ss.getTransferData(df[0]);
    return obj;
}
```

Decompilation of the class containing these declarations:

```java
int intNumber = 10;
int longNumber = 10;
int floatNumber = 10;
int doubleNumber = 10;
```
Chapter 5: Arrays, Collections, and Data Structures
containsElementObjectV1(T[] arr, T toContain)
containsElementObjectV2(T[] arr, T toContain, Comparator<? super T> c)
containsElementObjectV3(T[] arr, T toContain, Comparator<? super T> c)
containsElementV1(int[] arr, int toContain)
containsElementV2(int[] arr, int toContain)
containsElementV3(int[] arr, int toContain)
findIndexOfElementObjectV1(T[] arr, T toFind)
findIndexOfElementObjectV2(T[] arr, T toFind, Comparator<? super T> c)
findIndexOfElementObjectV3(T[] arr, T toFind, Comparator<? super T> c)
findIndexOfElementV1(int[] arr, int toFind)
findIndexOfElementV2(int[] arr, int toFind)

sortByKeyList(Map<K, V> map)
sortByKeyStream(Map<K, V> map, Comparator<? super K> c)
sortByKeyTreeMap(Map<K, V> map)
sortByValueList(Map<K, V> map)
sortByValueStream(Map<K, V> map, Comparator<? super V> c)
The diagram illustrates the process of path compression in data structures. The table on the left side provides the values, indices, binary representations, LSB, and the range of responsibility for each value.

- **Values**: 5, 11, 0, 9, 4, 1, 3, 0, 13, 7, 9, 12, 8, 5, 1, 3, 0
- **Indices**: 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0
- **Binary Representations**: The binary representations of the values are shown.
- **LSB**: The least significant bit (LSB) of each value's binary representation.
- **Range of Responsibility**: The range of values that each value affects.
- **BIT**: The bit position in the range of responsibility.

Path compression is a technique used to improve the efficiency of data structures like balanced search trees. It reduces the number of nodes that need to be accessed during a `find` operation by shortening the path to the root.
### Range Queries Examples

- $\sum[2,9] = \sum[1,9] - \sum[1,1] = (9 + 58) - (3) = 55$
- $\sum[5,10] = \sum[1,10] - \sum[1,5] = (3 + 58) - (17) = 44$
- $\sum[9,13] = \sum[1,13] - \sum[1,9] = (9 + 8 + 58) - (0 + 58) = 17$
- $\sum[15,16] = \sum[1,16] - \sum[1,15] = (91) - (9 + 8 + 58) = 16$

### Pairwise Independent Hash Functions

```
add element
```

```
H1  H2  H3
```

```
<table>
<thead>
<tr>
<th>Value</th>
<th>Index</th>
<th>Binary</th>
<th>LSB</th>
<th>Range of responsibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>16</td>
<td>10000₂</td>
<td>$2^4$</td>
<td>16</td>
</tr>
<tr>
<td>11</td>
<td>15</td>
<td>01111₂</td>
<td>$2^0$</td>
<td>16</td>
</tr>
<tr>
<td>0</td>
<td>14</td>
<td>01110₂</td>
<td>$2^2$</td>
<td>16</td>
</tr>
<tr>
<td>9</td>
<td>13</td>
<td>01101₂</td>
<td>$2^0$</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>01100₂</td>
<td>$2^2$</td>
<td>16</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>01011₂</td>
<td>$2^0$</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>01010₂</td>
<td>$2^1$</td>
<td>16</td>
</tr>
<tr>
<td>0</td>
<td>9</td>
<td>01001₂</td>
<td>$2^0$</td>
<td>16</td>
</tr>
<tr>
<td>13</td>
<td>8</td>
<td>01000₂</td>
<td>$2^3$</td>
<td>16</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>00111₂</td>
<td>$2^0$</td>
<td>16</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>00110₂</td>
<td>$2^1$</td>
<td>16</td>
</tr>
<tr>
<td>12</td>
<td>5</td>
<td>00101₂</td>
<td>$2^0$</td>
<td>16</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>00100₂</td>
<td>$2^2$</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>00011₂</td>
<td>$2^0$</td>
<td>16</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>00010₂</td>
<td>$2^1$</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>00001₂</td>
<td>$2^0$</td>
<td>16</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>16</td>
</tr>
</tbody>
</table>

```

The diagram shows the process of adding an element to the system and checking if it is present using the AND operation on the outputs of the hash functions $H1$, $H2$, and $H3$. The result is:

```
AND(H1(0), H2(0), H3(0)) AND (1, 1, 1)
```

If the result is true, the element is present; otherwise, it may or may not be true.
Chapter 6: Java I/O Paths, Files, Buffers, Scanning, and Formatting

Raw JSON
{"type": "Gac", "weight": 2000}
{"type": "Hemi", "weight": 1200}

Array-like JSON
[{
  "type": "Gac",
  "weight": 2000
}, {
  "type": "Hemi",
  "weight": 1200
}]

Map-like JSON
{
  "A": {
    "type": "Gac",
    "weight": 2000
  },
  "B": {
    "type": "Hemi",
    "weight": 1200
  }
}

CSV
gaac,2000
hemi,1500
cantaloupe,800
golden prize,2300
crenshaw,3000

This is a file for testing mismatches between two files!
file1.txt

This is a file for testing mismatches between two files!
file2.txt

This is a file for testing mismatches between two files!
file3.txt

This is a file for testing mismatches between two files!
file4.txt
buffer
circular buffer
### Obtained

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>78910</td>
<td>0.9276730641526881</td>
<td></td>
</tr>
<tr>
<td>83222</td>
<td>0.28423903775300785</td>
<td></td>
</tr>
<tr>
<td>5593</td>
<td>0.866538798997145</td>
<td></td>
</tr>
<tr>
<td>57329</td>
<td>0.9145723363689985</td>
<td></td>
</tr>
<tr>
<td>61443</td>
<td>0.41527451214386724</td>
<td></td>
</tr>
<tr>
<td>9043</td>
<td>0.8442927124583571</td>
<td></td>
</tr>
<tr>
<td>474</td>
<td>0.915912616950742</td>
<td></td>
</tr>
<tr>
<td>45763</td>
<td>0.04438867226365116</td>
<td></td>
</tr>
<tr>
<td>26671</td>
<td>0.4648636732351614</td>
<td></td>
</tr>
<tr>
<td>24096</td>
<td>0.12870733626570974</td>
<td></td>
</tr>
</tbody>
</table>

### Wanted

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>78910</td>
<td>0.928</td>
<td></td>
</tr>
<tr>
<td>83222</td>
<td>0.284</td>
<td></td>
</tr>
<tr>
<td>5593</td>
<td>0.867</td>
<td></td>
</tr>
<tr>
<td>57329</td>
<td>0.915</td>
<td></td>
</tr>
<tr>
<td>61443</td>
<td>0.415</td>
<td></td>
</tr>
<tr>
<td>9043</td>
<td>0.844</td>
<td></td>
</tr>
<tr>
<td>474</td>
<td>0.916</td>
<td></td>
</tr>
<tr>
<td>45763</td>
<td>0.044</td>
<td></td>
</tr>
<tr>
<td>26671</td>
<td>0.465</td>
<td></td>
</tr>
<tr>
<td>24096</td>
<td>0.129</td>
<td></td>
</tr>
</tbody>
</table>

---

**doubles.txt**

```
78,910 bytes
83,222 bytes
5,593  bytes
57,329 bytes
61,443 bytes
9,043  bytes
474   bytes
45,763 bytes
26,671 bytes
24,096 bytes
```

---

**people.txt**

```
Matt,Kyle,23,San Francisco;
Darel,Der,50,New York; Sandra,Hui,40, Dallas;
Leonard,Vurt,43,Bucharest; Mark,Seil,19, Texas; Ulm,Bar,43, Kansas
```
Chapter 7: Java Reflection Classes, Interfaces, Constructors, Methods, and Fields

Compiled from "Car.java"
public class modern.challenge.Car {
  public modern.challenge.Car() {
    throws java.lang.NoSuchFieldException, java.lang.IllegalArgumentException, java.
    lang.IllegalArgumentException;
  } // Engine=class modern/challenge/Car$Engine of class modern/challenge/Car

SourceFile: "Car.java"
NestMembers:
  modern/challenge/Car$Engine
InnerClasses:
  public #14 = #6 of #4;
  // Engine=class modern/challenge/Car$Engine of class modern/challenge/Car

package modern.challenge;

@Fruit(name = "melon", value = "delicious")
public class Melon extends @Family Cucurbitaceae implements @ByWeight Comparable {
```java
@Ripe(true)
public void eat() throws @Runtime IllegalStateException {
}

@Ripe(true)
public void eat() throws @Runtime IllegalStateException {
}

public @Shape("oval") List<Seed> seeds() {
    return Collections.emptyList();
}

public void slice(@Ripe(true) @Shape("square") int noOfSlices) {
}

@Unit
private final int weight;

@Fruit(name = "melon", value = "delicious")
public class Melon extends @Family Cucurbitaceae implements @ByWeight Comparable {

@Fruit(name = "melon", value = "delicious")
public class Melon extends @Family Cucurbitaceae implements @ByWeight Comparable {
```
```java
module org.player {
    requires org.tournament;
}
```

```
module org.tournament {
    exports com.management;
}
```

---

**Diagram Description:**

- **Dynamic Proxy:**
  - `method1()`
  - `method2()`
  - `method3()`

- **InvocationHandler Implementation:**
  - `method1()`
  - `method2()`
  - `method3()`

- **Dispatch Invocation:**
  - The invocation handler to dispatch method invocations to

Chapter 8: Functional Style Programming - Fundamentals and Design Patterns

```java
public class GacMelonPredicate implements MelonPredicate {
    public boolean test(Melon melon) {
        return "gac".equalsIgnoreCase(melon.getType());
    }
}

public class HugeMelonPredicate implements MelonPredicate {
    public boolean test(Melon melon) {
        return melon.getWeight() > 5000;
    }
}

List<Melon> filterMelons(List<Melon> melons, MelonPredicate predicate) {...}
```
List<Melon> europeans = Filters.filterMelons(melons, new MelonPredicate() {
    @Override
    public boolean test(Melon melon) {
        return "europe".equalsIgnoreCase(melon.getOrigin());
    }
});

(this anonymous inner class creation can be turned into a lambda expression.

(File folder, String fileName) -> folder.canRead() && fileName.endsWith(".pdf");

<table>
<thead>
<tr>
<th>Anonymous</th>
<th>Function</th>
<th>Concise</th>
<th>Passably</th>
</tr>
</thead>
<tbody>
<tr>
<td>no explicit name</td>
<td>doesn't belong to a class</td>
<td>small</td>
<td>pass it as argument or store it</td>
</tr>
</tbody>
</table>

try (Scanner scanner = new Scanner(
    Path.of("doubles.txt"), StandardCharsets.UTF_8)) {
}
Chapter 9: Functional Style Programming - a Deep Dive
After:

```java
stream(): anna
stream(): bob
stream(): christian
filter(): christian
map(): CHRISTIAN
stream(): carmen
filter(): carmen
map(): CARMEN
stream(): rick
stream(): carla
filter(): carla
map(): CARLA
sorted(): CARLA
sorted(): CARMEN
sorted(): CHRISTIAN
```

Exception in thread "main" java.lang.NullPointerException
   at modern.challenge.Main.lambda$main$1(Main.java:16)
...
\[ H = \frac{n}{\frac{1}{x_1} + \frac{1}{x_2} + \cdots + \frac{1}{x_n}} = \frac{n}{\sum_{i=1}^{n} \frac{1}{x_i}} = \left( \frac{\sum_{i=1}^{n} x_i^{-1}}{n} \right)^{-1} \]
Chapter 10: Concurrency - Thread Pools, Callables, and Synchronizers
1. Thread T waits two other threads
2. T2 completed and decrement the counter from 2 to 1
3. T1 completed and decrement the counter from 1 to 0
4. T completed

1. Three threads heading to the barrier
2. One thread waiting at the barrier
3. Two threads waiting at the barrier
4. All threads are at the barrier
5. The threads go further
1. Each thread is performing its tasks
2. T2 arrives at the exchange point and waits for T1
3. T1 arrives at the exchange point as well
4. T1 and T2 exchange objects and continue their tasks
Chapter 11: Concurrency - Deep Dive

Fork tasks in smaller subtasks in a recursive approach

task small enough

Evaluate partial results in parallel

Join the partial results to obtain the final result

[Diagram of task dependency]

[Diagram of process flow]
CompletableFuture.supplyAsync() -> {
    // Code prone to exception
    return "result1";
}).thenApply(r1 -> {
    // Code prone to exception
    return "result2";
}).thenApply(r2 -> {
    // Code prone to exception
    return "result3";
}).thenAccept(r3 -> {
    // Code prone to exception
}).

Exception in supplyAsync()  Exception in 1st thenApply()  Exception in 2nd thenApply()  Exception in thenAccept()
newCondition

```java
Lock/ReentrantLock lock = new ReentrantLock();
Condition condition = lock.newCondition();
public void execute() throws InterruptedException {
    lock.lock();
    try {
        ...
        while/if(some_condition) {
            condition.await();
        }
    } finally {
        lock.unlock();
    }
    lock.lock();
    try {
        condition.signal(All);()
    } finally {
        lock.unlock();
    }
```

When `await()` is called the thread releases the lock. After getting the signal to continue
the thread must acquire the lock again.

Multiple Readers

One Writer
Chapter 12: Optional

No images
Chapter 13: The HTTP Client and WebSocket APIs

---

Lorem Ipsum Generator

---

```text
Content-Disposition: form-data; name="filefield"; filename="figure.png"
Content-Type: image/png

---

```