LEC 13

CSE 122

Interfaces

Questions during Class?

Raise hand or send here

sli.do #cse122



BEFORE WE START

Talk to your neighbors:

What is your favorite music/video/sound to have on while studying?

Respond on sli.do!

Instructor: Brett Wortzman

TAs:

Archit	Daniel	Naomi	Sreshta
Blake P.	David	Neha	Suyash
Blake R.	Hanna	Nicole	Trisha
Cady	Isis	Rio	Wesley
Carson	Jun	Rohan	Yash
Cole	Landon	Shreyank	
Connor	Michelle	Somya	

Music: 🎷 CSE 122 26au Lecture Tunes 🎶

Lecture Outline

- Announcements
- Interface Review
- More Shapes!

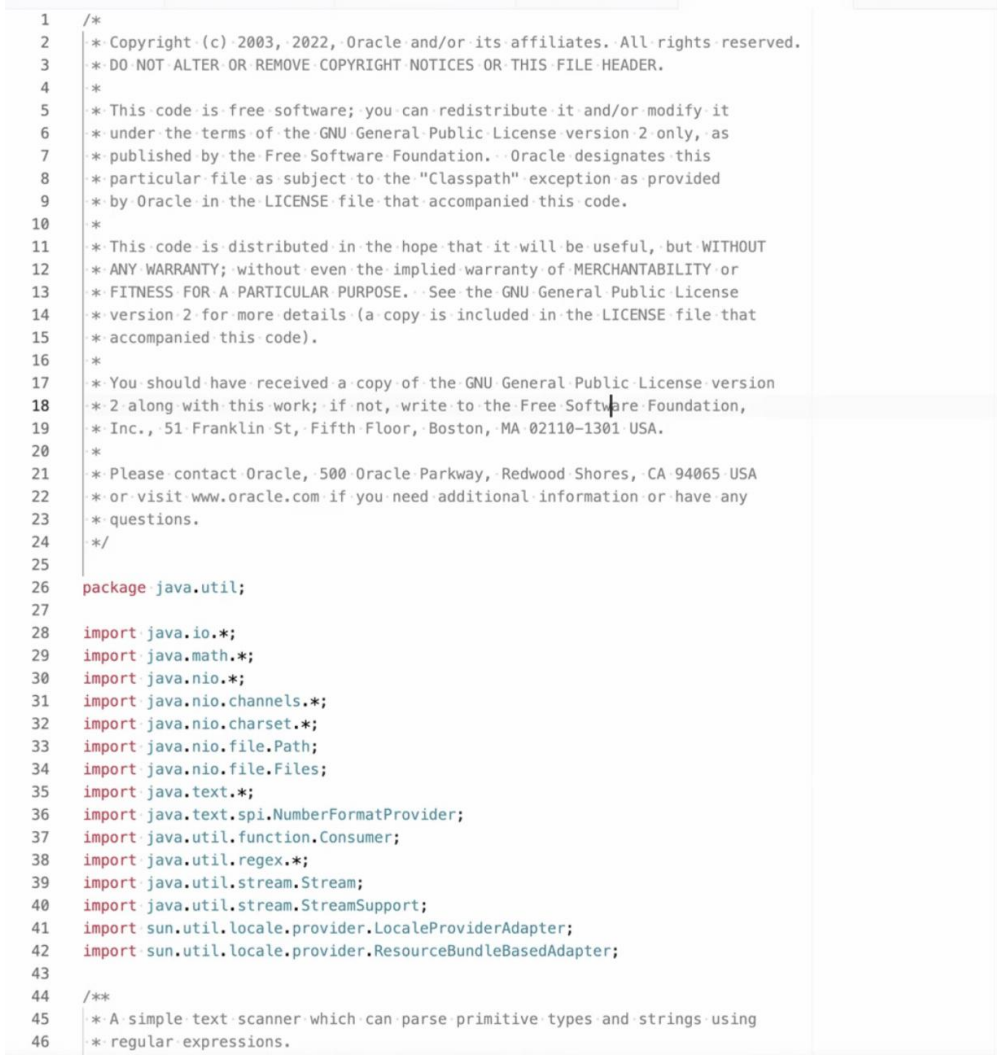
Announcements

- CQ2 ongoing
  - Sign up early!
- No class on 11/24 or 11/25
- PQ2 on Tuesday, 12/1
  - Right after Thanksgiving

Lecture Outline

- Announcements
- Interfaces Review
- More Shapes!

Recall: Abstraction



```
/*
 * Copyright (c) 2003, 2022, Oracle and/or its affiliates. All rights reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Oracle designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Oracle in the LICENSE file that accompanied this code.
 *
 * This code is distributed in the hope that it will be useful, but WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
 * FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Oracle, 500 Oracle Parkway, Redwood Shores, CA 94065 USA
 * or visit www.oracle.com if you need additional information or have any
 * questions.
 */

package java.util;

import java.io.*;
import java.math.*;
import java.nio.*;
import java.nio.channels.*;
import java.nio.charset.*;
import java.nio.file.Path;
import java.nio.file.Files;
import java.text.*;
import java.text.spi.NumberFormatProvider;
import java.util.function.Consumer;
import java.util.regex.*;
import java.util.stream.Stream;
import java.util.stream.StreamSupport;
import sun.util.locale.provider.LocaleProviderAdapter;
import sun.util.locale.provider.ResourceBundleBasedAdapter;

/**
 * A simple text scanner which can parse primitive types and strings using
 * regular expressions.
```

vs

A `Scanner` gets text input and turns it into data.

Recall from L6: Wait, ADT? Interfaces?

- **Abstract Data Type (ADT)**: A *description of the idea* of a data structure including what operations are available on it and how those operations should behave. For example, the English explanation of what a list should be.

- **Interface**: Java construct that lets programmers *specify what methods a class should have*. For example the `List` interface in java.

- **Implementation:** *Concrete code* that meets the specified interface. For example, the `ArrayList` and `LinkedList` classes that implement the `List` interface.

```
List<String> myList = new ArrayList<>();
```

Interfaces

Interfaces serve as a sort of "contract"– in order for a class to implement an interface, it must fulfill the contract requirements.

The contract requirements are certain methods that the class must implement.

`Lists`

One ADT we've talked a lot about in this course is a list.

Within Java, there exists a `List` interface – its contract includes methods like:

`add`, `clear`, `contains`, `get`, `isEmpty`, `size`, `indexOf`

There's also an `ArrayList` class (implementation)

To get the certificate, it must include all these methods (and any others the `List` interface specifies)

Interfaces vs. Implementation

Interfaces require certain methods, but they do not say anything about how those methods should be implemented – that's up to the class! 🏅

`List` is an interface

- `ArrayList` is a class that implements the `List` interface
- `LinkedList` is a class that implements the `List` interface
- …

Why interfaces?

```
public static void fill(WaterBottle w) {…}
```

This method only accepts a `WaterBottle`!

```
public static void fill(Jar j) {…}
public static void fill(MeasuringCup m) {…}
public static void fill(Tub t) {…}
```

Why interfaces?

Flexibility



```
public static void fill(Container c) {…}
```

This method can accept either a:

- `WaterBottle`
- Jar
- MeasuringCup
- `Tub` or
- Any other class that implements `Container`!

Why interfaces?

Flexibility



```
public static void method(Set<String> s) {…}
```

This method can accept either a:

- `HashSet<String>` or
- `TreeSet<String>` or
- Any other class that implements `Set` and whose element type is `String`!

Why interfaces?

Abstraction



```
public static void method(Set<String> s) {…}
```

This method can accept either a:

- `HashSet<String>` or
- `TreeSet<String>` or
- Any other class that implements `Set` and whose element type is `String`!

Lecture Outline

- Announcements
- Interfaces Review
- **More Shapes!** 

Classes can Implement Multiple Interfaces

A class can implement multiple interfaces – it's like one person signing multiple contracts!

If a class implements an interface A and an interface B, it'll just have to include all of A's required methods along with all of B's required methods

An interface can `extend` another

You can have one interface extend another

So if `public interface A extends B`, then any class that implements A must include all the methods in A's interface and all the methods in B's interface

An interface can extend another

We can write another interface:

`Polygon` that `extends` `Shape`

- "`Polygon is a Shape`"

`Make modifications such that:`

- `Square` is a `Polygon` (and `Shape`)
- `Triangle` is a `Polygon` (and `Shape`)
- `Circle` is a `Shape` (but not a `Polygon`)