LECTURE 4: ENCAPSULATION, PART II

CS 2110
Fall 2021

LECTURE 4: ENCAPSULATION, PART II

PART 1: DOCUMENTATION
SPECIFICATIONS
CLASS INVARIANTS

CS 2110
Fall 2021

Agenda

Previously in 2110:

- OOP: objects, classes, methods, fields
- Encapsulation: access modifiers, constructors

Today:

- About Eclipse workspaces
- Documentation
- Testing
- Static components
- Exceptions

Where is your Eclipse workspace?

To find out where the Workspace is on your hard drive:

Preferences -> General -> Workspace

Look at Show full workspace path:

When you want to upload (or just get at) a .java file that is in an Eclipse project, don't drag it from Eclipse! That removes it from the project!

Instead, get it from the hard drive –make sure you don't delete it!

Demo: Show the .java files in src and .class files in bin

Documentation needed

Other programmers do not want to read your code

They do need to read your documentation

Meme: https://tenor.com/view/nope-not-doing-that-gif-12221568

Comments in Java

```
// This is a one-line comment.

int x= 2110;  // So is this.

/* This is a block comment. It can spread
   over many lines. */

/** This is a Javadoc comment. <br>
  * It has special meaning. It can be extracted to HTML. <br>
  * It creates hover-over text in Eclipse. */
```

We'll explain the html tag <br> in a demo

Specification: fields

- Each field should have a Javadoc comment
- private fields: Explain to **implementers:**
  - Meaning of field
  - Any constraints on it
- public fields: Explain to **clients**

```
/** The denominator.  Not 0. */
int den;
```

Specification: Fields

- Class invariant: collection of all constraints on fields
  - Javadoc specs on fields
  - Possibly separate block comments

Demo fraction4

- Purpose of a constructor (revisited): Initialize object's state at creation to truthify class invariant

Urban Dictionary: To turn an obvious or blatant lie into the truth, and sell it boldly, with conviction.

Gries: To cause to be true. Contrast: falsify.

Specification: Methods (& constructors)

Each method must have a Javadoc comment.

- It **precedes** the method header.
- It is a command to do something
- It must mention/explain all parameters.

For a function

```
/** Return the maximum of x and y. */
public int max(int x, int y) { … }
```

or

```
/** = the maximum of x and y. */
public int max(int x, int y) { … }
```

The value of the call

max(x, y)

=

the maximum of x and y.

Specification: Methods (& constructors)

Each method must have a Javadoc comment.

- It precedes the method header.
- It is a command to do something
- It must mention/explain all parameters.

For a procedure

```
/** Sort b*/
void max(int[] b) { … }

/** Make this fraction be in reduced form*/
void reduce() { … }
```

Specification: Methods (& constructors)

Each method must have a Javadoc comment.

- It precedes the method header.
- It must mention/explain all parameters.

For a constructor

```
/** Constructor: An instance with numerator n
 *  and denominator d.
    Throw an IllegalArgumentException if d is 0 */
Fraction4(int n, int d) { … }
```

Specification: Methods (& constructors)

- For public methods, explain to **clients:**
  - Precondition: what they must guarantee
  - Postcondition: what method must guarantee
  - Do not mention private components!
- Private methods would explain to **implementers**

```
/** Constructor: an instance with num n and den d.
 *  Precondition: d is not 0.  */
Fraction3(int n, int d) { … }
```

Short quiz

Preconditions

```
/** Constructor: an instance with numerator n and
  *  denominator d.
  *  Precondition: d != 0. */
public Fraction3(int n, int d) {


   ...
}
```

If precondition false, code can do anything…

Image: https://www.flickr.com/photos/swedpix/36960629121

Preconditions

```
/** Constructor: an instance with numerator n and
  *  denominator d.
  *  Precondition: d != 0. */
public Fraction3(int n, int d) {
   assert d != 0;
   …
}
```

If precondition false, code can do anything…

But: Catch errors more quickly by using an assert statement

Preconditions

- Assert the precondition:
  - Good defensive programming
  - Not necessitated by precondition
- Argument types are not part of precondition in Java
  - Do not write "n and d must be ints"
  - The compiler already guarantees that, and we see the types in the method declaration

More about Documentation

When you write documentation, be aware of the style. Compare often to ones you see us use in this course!

Read JavaHyperText: Style Guide, Section 3

https://www.cs.cornell.edu/courses/JavaAndDS/JavaStyle.html#Comments

Demo Fraction4
in Eclipse

The end

LECTURE 4: ENCAPSULATION, PART II

PART 2: JUNIT TESTING

CS 2110
Fall 2021

Testing

Microsoft Word:

Has hundreds of features.

Fix a bug or add functionality?

- Have to ensure that all features still work
- Need a way to save test cases
- Must be able to run many test cases automatically, with a click of a button
- Generally, too many test cases to look carefully at results
- Output of testing should show only test cases that fail

TESTING IS THE TOPIC OF THIS WEEK'S RECITATION

JavaHyperText: Eclipse -> JUnit testing

Eclipse comes with a feature called JUnit testing.

Used to save test cases and run them any time with a click of a button.

JavaHyperText includes instructions on using JUnit testing.

Top horizontal navigation bar:

Eclipse -> 7. JUnit testing

It contains one short video, which you must watch as part of this lecture. It is designed to make you aware of an error you might make if you are not careful.

Two points to watch out for

Use ONLY this in a JUnit testing class:

assertEquals(expected value, computed value);

Conditional expression: Type ? Into JavaHyperText Filter field

bool-exp ? exp1 : exp2

e.g. 5 < 2 ? “bb” : “cc” value is “cc”
e.g. 2 < 5 ? 5 : 3+7 value is 5

The end

LECTURE 4:
ENCAPSULATION, PART II

PART 3: STATIC COMPONENTS
METHOD MAIN

CS 2110
Fall 2021

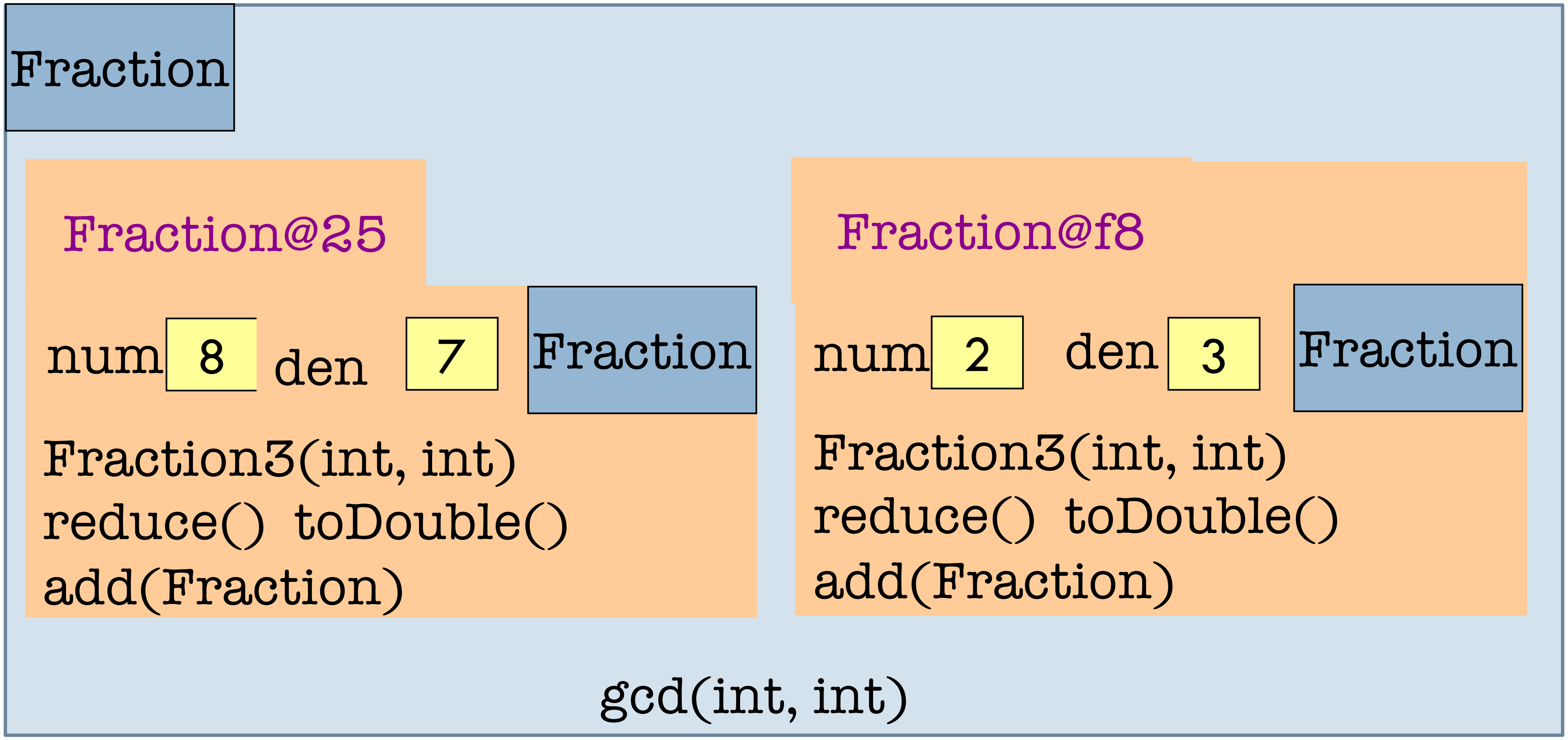
Static Components

- Some behavior does not depend on object's state
- Static methods belong to the class as a whole, not the objects in particular

```
private static int gcd(int a, int b) { … }
int g= Fraction.gcd(2,4)
```

- Some data are not state of an object, e.g.,
  - static double Math.PI
  - static PrintStream System.out
- Static fields belong to the class as a whole, not the objects in particular

Static components



To reference static gcd from another class: Fraction.gcd(5, 3)

When to make a method static

If a method does not refer directly to a field of the class or an instance method of the class, make it static.

e.g. gcd(a, b)
    max(x, y)

Class Math contains ONLY static methods and variables.

Look at class Math

Method main

```
class ... {
   public static void main(String[] args) { ... }
}
```

- **Application:** Java program with such a method
- Execution begins with method main

Modifier final

- Used with field or local variable
- Final variable cannot be changed
- Useful for constants
- Often use names in all caps

```
public static final Fraction4 ONE=
      new Fraction4(1);

public static final double PI =
      3.14159265358979323846;
```

The end

LECTURE 4: ENCAPSULATION, PART II

PART 3: EXCEPTIONS

CS 2110
Fall 2021

Exceptions

Image: https://cornellbigred.com/news/2019/10/3/football-georgetown-recap-2019.aspx

Exceptions

- Used when something has gone wrong
  - An assert statement fails
  - An argument is illegal, so nothing can be returned
  - A method is called with no object, e.g. null.m(…)
  - Division by 0, e.g. 5/x where x is 0.
- Java: We introduce exceptions now. We will complete the discussion in next lecture after we introduce subclasses and inheritance

Demo1: **output** of a thrown exception

Here are some Exceptions

ArithmeticException

ArrayIndexOutOfBoundsException

AssertionError

IllegalArgumentException,

IndexOutOfBoundsException

IOException

NegativeArraySizeException

NullPointerException

UnsupportedOperationException

That's enough for now. The name says why it is thrown.

Lecture 5: We explain the whole hierarchy of objects that can be thrown. For now, this is all you need

These are names of classes that come with Java!

Throwing Exceptions

- Exceptions are objects
- Create just like other objects:
  `new ExceptionClass(args);`

```
if (c == 0)
    throw ArithmeticException("div by 0");
```

```
if (d == 0) {
  throw new IllegalArgumentException()}
```

new-exp can have 0 args or one String arg (overloading!)

Catching Exceptions: try-statement

```
try {
        code1 that might throw an exception
} catch (ArithmeticException e) {
        code2 that handles it
}
```

Changes the normal flow of execution

Case 1: code1 doesn't throw exception. When code1 finishes, that's the end of execution of the try-statement.

Catching Exceptions: try-statement

```
try {
        code1 that might throw an exception
} catch (ArithmeticException e) {
        code2 that handles it
}
```

Changes the normal flow of execution

Case 2: code1 throws an ArithmeticException. The catch-clause catches it, and code2 is executed. When finished, that's the end of execution of the try-statement.

Catching Exceptions: try-statement

```
try {
        code1 that might throw an exception
} catch (ArithmeticException e) {
        code2 that handles it
}
```

Changes the normal flow of execution

Case 3: code1 throws an exception other than an ArithmeticException. The exception is thrown out to the place that called the method in which this try-statement resides.

Demo2: **output** of a thrown exception

Catching Exceptions

```
try {
        code that might throw an exception
} catch (ArithmeticException e) {
        code that handles it
} catch (IndexOutOfBoundsException e) {
        code that handles it
}
```

Demo3: **more than one catch clause**

Summary: Encapsulation

- Encapsulation: enclose the object
  - Make design robust
  - Build big programs
  - One of the key techniques of OOP

- New Java features we covered:
  - Javadoc
  - JUnit testing
  - Static components
  - Exceptions –more next time

Your Turn: Read in JavaHyperText

- getter, setter
- class invariant
- javadoc (comment)
- specification (method), precondition, postcondition
- style Guide section 3 (Documentation)
- static (field, method)
- final
- exceptions: all the pages in drop-down menu "Exceptions" (wait until after Thursday)