Java overview

Java is a byte-compiled language.

Java has
static types.

primitive values *vs* objects

Memory model

Primitive values

`int` • `double` • `boolean`

other built-in types …

```
int x = 3;
int y = x;
int z = 3;
```

x	y	z
3	3	3

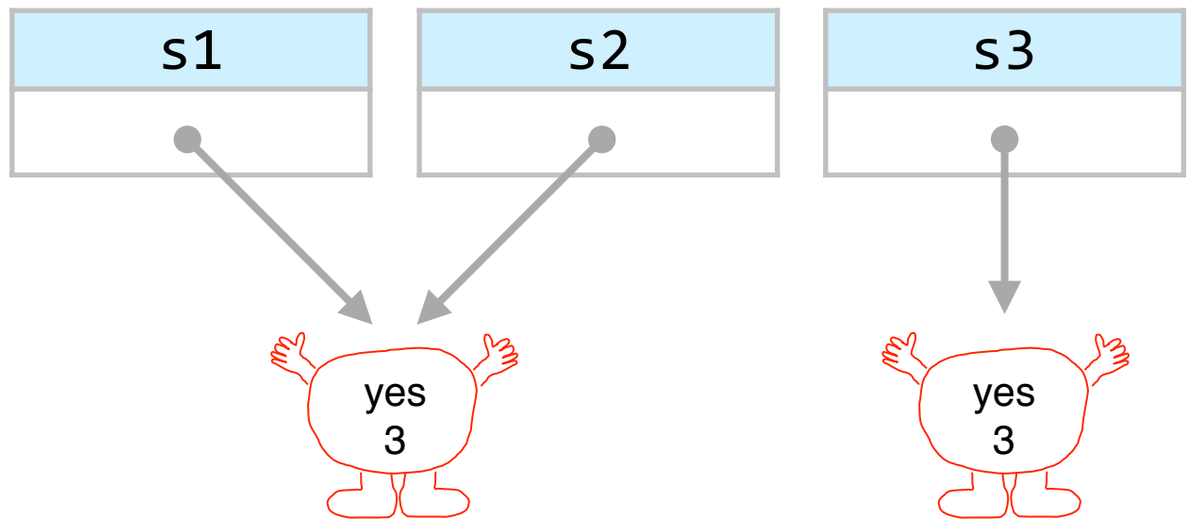
Java directly stores primitive values.

Objects

`String` • `LinkedList`

other library & user-defined types …

```
String s1 = "yes";
String s2 = s1;
String s3 = "yes";
```



Java stores references to objects.

== *vs* .equals

==

compares what's in the box

```
x == y;   // true
y == z;   // true
s1 == s2; // true
s2 == s3; // false
```

.equals

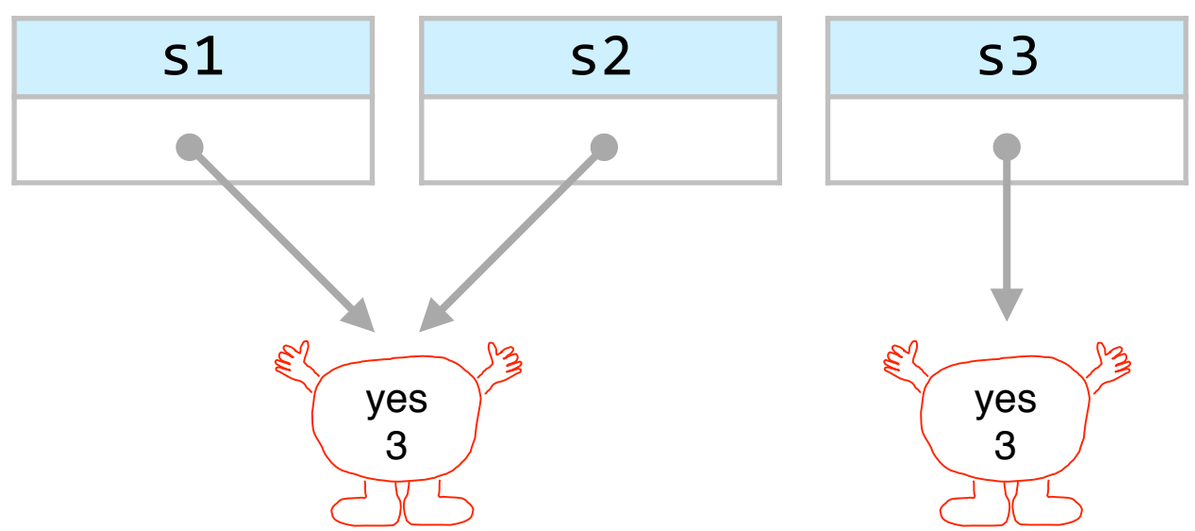
calls a method (usually checks for equal values)

```
s1.equals(s2);   // true
s2.equals(s3);   // true
```

```
int x = 3;
int y = x;
int z = 3;
```

x	y	z
3	3	3

```
String s1 = "yes";
String s2 = s1;
String s3 = "yes";
```



	primitives	**objects**
variable stores the value	✔	
variable stores a reference		✔
supports ==	✔	✔ but it 's probably not what you want
supports `.equals`		✔
we can define new kinds		✔
type name starts with lower-case letter	✔ `int value  // primitive`	
type name starts with upper-case letter		✔ `Dog lucky  // object`

Object-oriented Programming (again ☺)

1110000001001110001010011100110110111110100000 10

Binary

```
;-------------------------------------------------------------
; zstr_count:
; Counts a zero-terminated ASCII string to determine its size
; in:   eax = start address of the zero terminated string
; out:  ecx = count = the length of the string

zstr_count:                     ; Entry point
    mov  ecx, -1                ; Init the loop counter, pre-decrement
                                ;  to compensate for the increment
.loop:
    inc  ecx                    ; Add 1 to the loop counter
    cmp  byte [eax + ecx], 0    ; Compare the value at the string's
                                ;  [starting memory address Plus the
                                ;  loop offset], to zero
    jne  .loop                  ; If the memory value is not zero,
                                ;  then jump to the label called '.loop',
                                ;  otherwise continue to the next line
.done:
                                ; We don't do a final increment,
                                ;  because even though the count is base 1,
                                ;  we do not include the zero terminator in the
                                ;  string's length
    ret                         ; Return to the calling program
```

https://upload.wikimedia.org/wikipedia/commons/f/f0/Zstr_count_x86_assembly.png

Assembly

```
      IA = NA
      IB = NB
1     IF (IB.NE.0) THEN
        ITEMP = IA
        IA = IB
        IB = MOD(ITEMP, IB)
        GOTO 1
      END IF
      NGCD = IA
```

https://en.wikibooks.org/wiki/Fortran/Fortran_examples

Higher-level constructs

A Case against the GO TO Statement.

by Edsger W.Dijkstra

Technological University

Eindhoven, The Netherlands

Structured programming

```
def sumValues(values):
    sum = 0
    for value in values:
        sum += value
    return sum

sumValues([1,2,3])
```

Functions

```
x = 3

def f(x):
  x *= 2
  return x * 4

def g(y):
  global x
  x *= 2
  return x + y

x = "hi there!"
x = f(g(x))

print x
```

Global and local data

```
def sumValues(values):
    sum = 0
    for value in values:
        sum += value
    return sum
```

```
sumValues([1,2,3])
```

behavior

data

Programs = Behavior + Data

What is object-oriented programming good for?

Object-oriented programming helps us manage the complexity of programs by:

1. **combining data with the behavior** that operates over it
2. breaking large programs into smaller, **self-contained** pieces
3. separating **interface** (*what* a piece of code can do) from **implementation** (*how* that piece of code works)

Note: there's an underlying assumption that your program is complex enough to need OOP.

An object…

- combines **data (fields)** and **behavior (methods)**
- is self-contained (and knows about itself)
- separates **interface (what)** from **implementation (how)**

Object-oriented programming languages **differ** in:

- how the programmer specifies an object's **interface**
- how the programmer specifies an object's **implementation**
- how objects are **created**, **initialized**, **queried**, and **updated**
- **encapsulation** mechanism
  how strictly the language *enforces* the separation between interface & implementation

Object-oriented Programming

in Java

A class is like...

a blueprint

http://allexincasa.ig.com.br/tag/arquitetura/

Objects are like...

houses

http://allexincasa.ig.com.br/tag/arquitetura/

http://curbed.com/uploads/simpsons_house_1-%281%29.jpg

A class is like…

a cookie cutter

ecx.images-amazon.com/images/I/21owTyO6HaL.jpg

Objects are like…

cookies

eclecticrecipes.com/wp-content/uploads/2013/02/heart-6.jpg

images.edge-generalmills.com/9b6a8635-686e-4b7d-863b-7dd3d8d25a04.jpg

A class is like…

factory

si.wsj.net/public/resources/images/P1-AO506_TURNPI_G_20090129173936.jpg

Objects are like…

cars

A class is like…

factory

www.tipsquirrel.com/wp-content/uploads/2010/09/Extrude1.jpg

Objects are like…

delicious,
totally edible
playdough

class: a blueprint for an object; contains implementation

object: a self-contained instance of a class

field: stores data

method: defines a behavior

constructor: initializes an object's fields

getter: a method that lets us read an object's data

setter: a method that lets us change an object's data

this: how an object knows about itself

interface: what an object can do

implementation: how an object does its thing

public: indicates a piece of the interface

private: indicates a piece of the implementation

```java
class Point {
    /** the x (horizontal) coordinate */
    private double x;

    /** the y (vertical) coordinate */
    private double y;

    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }

    public double getX() {
        return this.x;
    }

    public void setX(double x) {
        this.x = x;
    }

    public double getY() {
        return this.y;
    }

    public void setY(double y) {
        this.y = y;
    }

    /**
     * returns the sum of this point and another
     *
     * @param other another Point object
     * @return a new Point, the sum of this and other
     */
    public Point add(Point other) {
      return new Point(this.getX() + other.getX(),
                       this.getY() + other.getY());
    }
}
```

field definition

Javadoc comment

constructor definition

getter

field access

setter

method definition

getter

setter

(this is an object)

class

Javadoc comment

constructor call

lots of (getter) method calls!

object!

Be on the lookout for

- Where's the interface? Where's the implementation?
- How to create, initialize, query, and update an object
- How does Java enforce separation of interface & implementation?

- object-oriented vocabulary
- good programming practices
- good programming style
- when (not) to use a particular object-oriented feature

- how to do things in Java
- how to do things in Eclipse

- questions / confusions / pondering

An Excel-ent analogy

Fields are like a spreadsheet

Class definition ≈ columns

a class defines the names and types (but not the values) of fields

	color	capacity	fullness
Colleen's mug	blue	100	100
Ben's jug	puce	1000	500
Zach's coffee cup	white & green	100000	0

Objects ≈ rows

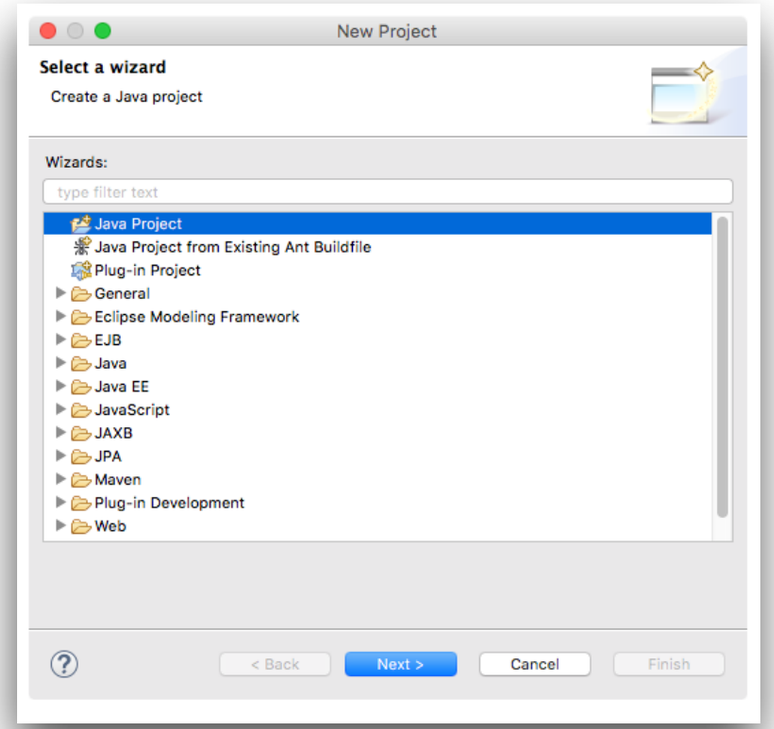
each object has specific values for its field

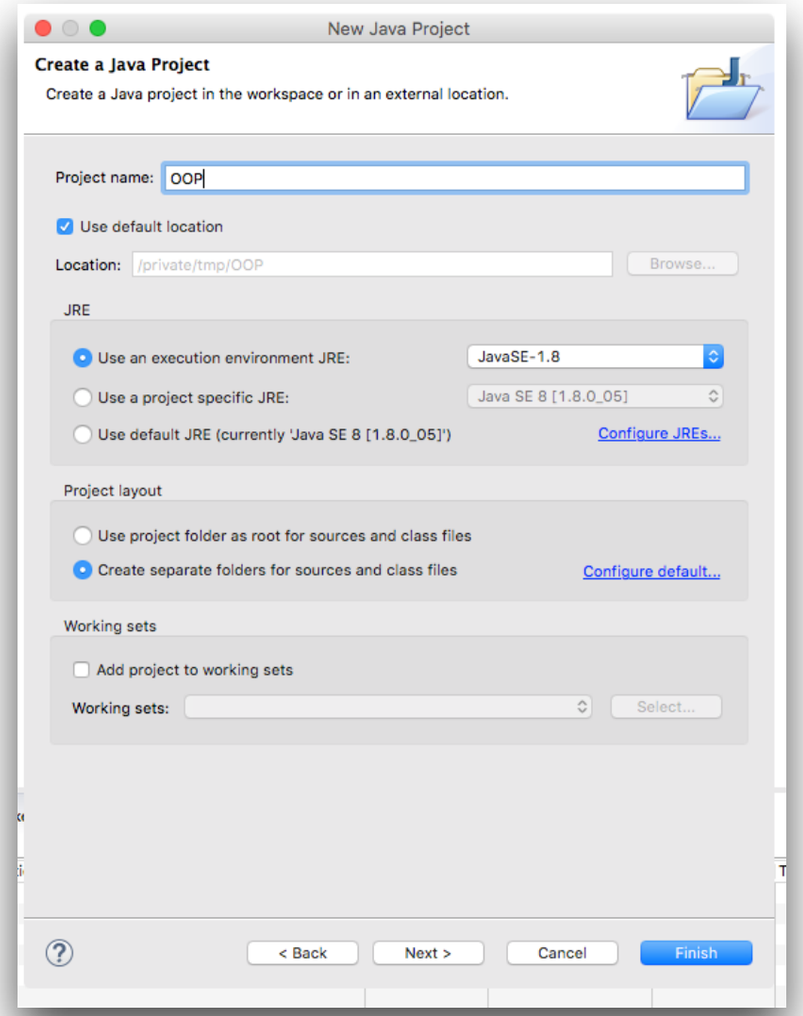
Build this up, piece-by-piece:

```
public class DrinkContainer {
    /** describes the color of the container */
    private String color;

    /** amount of liquid the container can hold, in milliliters */
    private int capacity;

    /** the amount of liquid currently in the container */
    private int fullness;

    public DrinkContainer(String color, int capacity) {
        this.capacity = capacity;
        this.color = color;
        this.fullness = 0;
    }

    public String getColor() {
        return this.color;
    }

    public void setColor(String newColor) {
        this.color = newColor;
    }

    public int getCapacity() {
        return this.capacity;
    }

    public void setCapacity(int newCapacity) {
        this.capacity = newCapacity;
    }

    public int getFullness() {
        return this.fullness;
    }

    /**
     * Sets the new liquid amount for the mug. If the new amount is negative or
     * exceeds the mug's capacity, the amount is unchanged.
     *
     * @param newAmount
     */
    public void setFullness(int newAmount) {
        if (newAmount >= 0 && newAmount <= this.getCapacity()) {
            this.fullness = newAmount;
        }
    }

    /**
     * Fills the cup to capacity
     */
    public void fill() {
        this.setFullness(this.getCapacity());
```

How to create an Eclipse Project

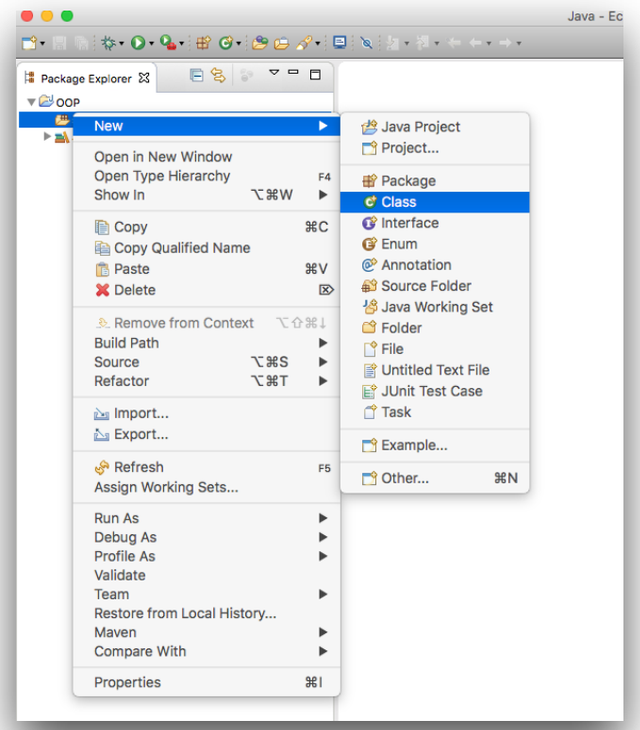
File → New Java Project



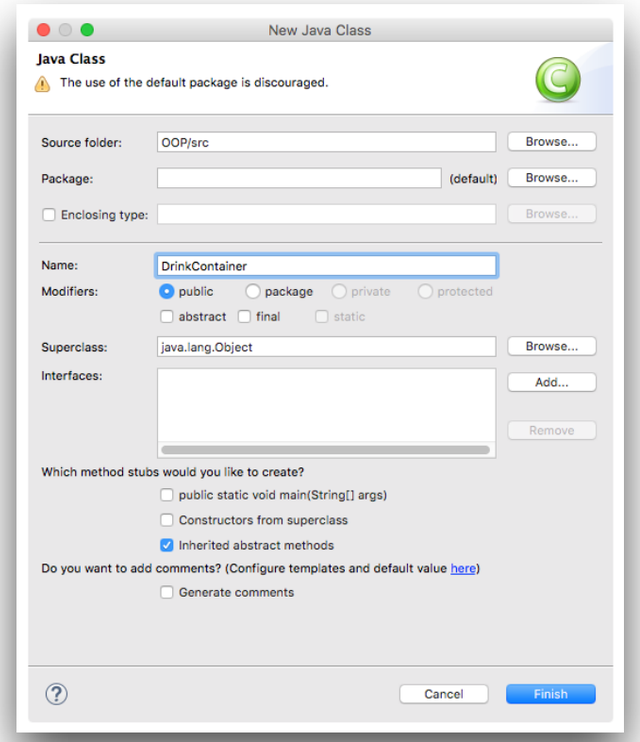


How to create a new Java class

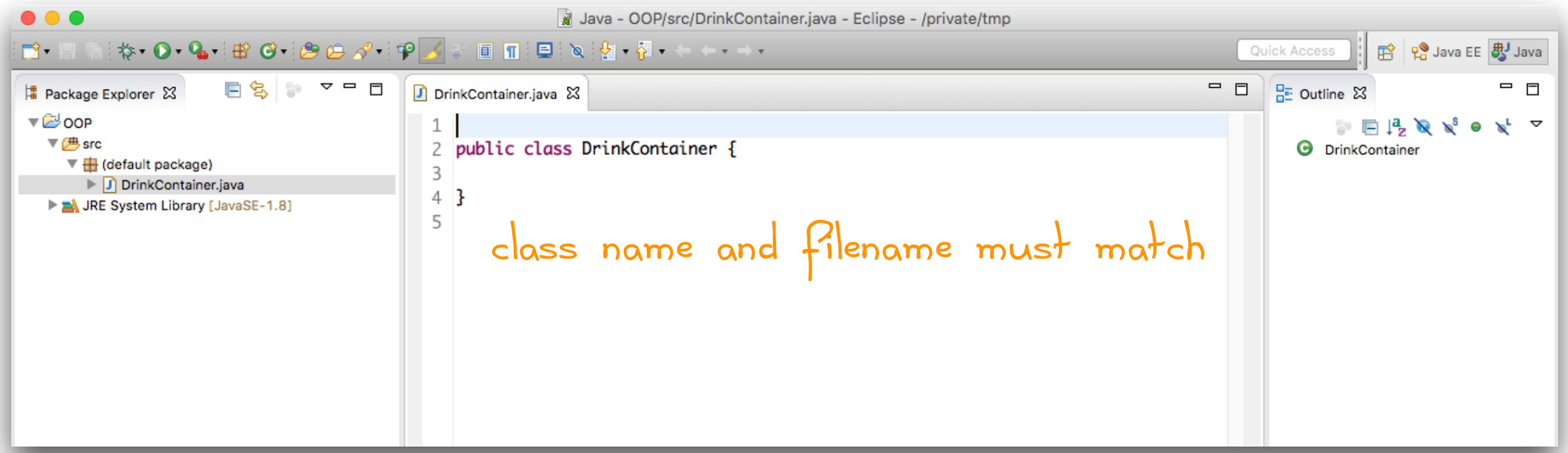
Right-click the src folder



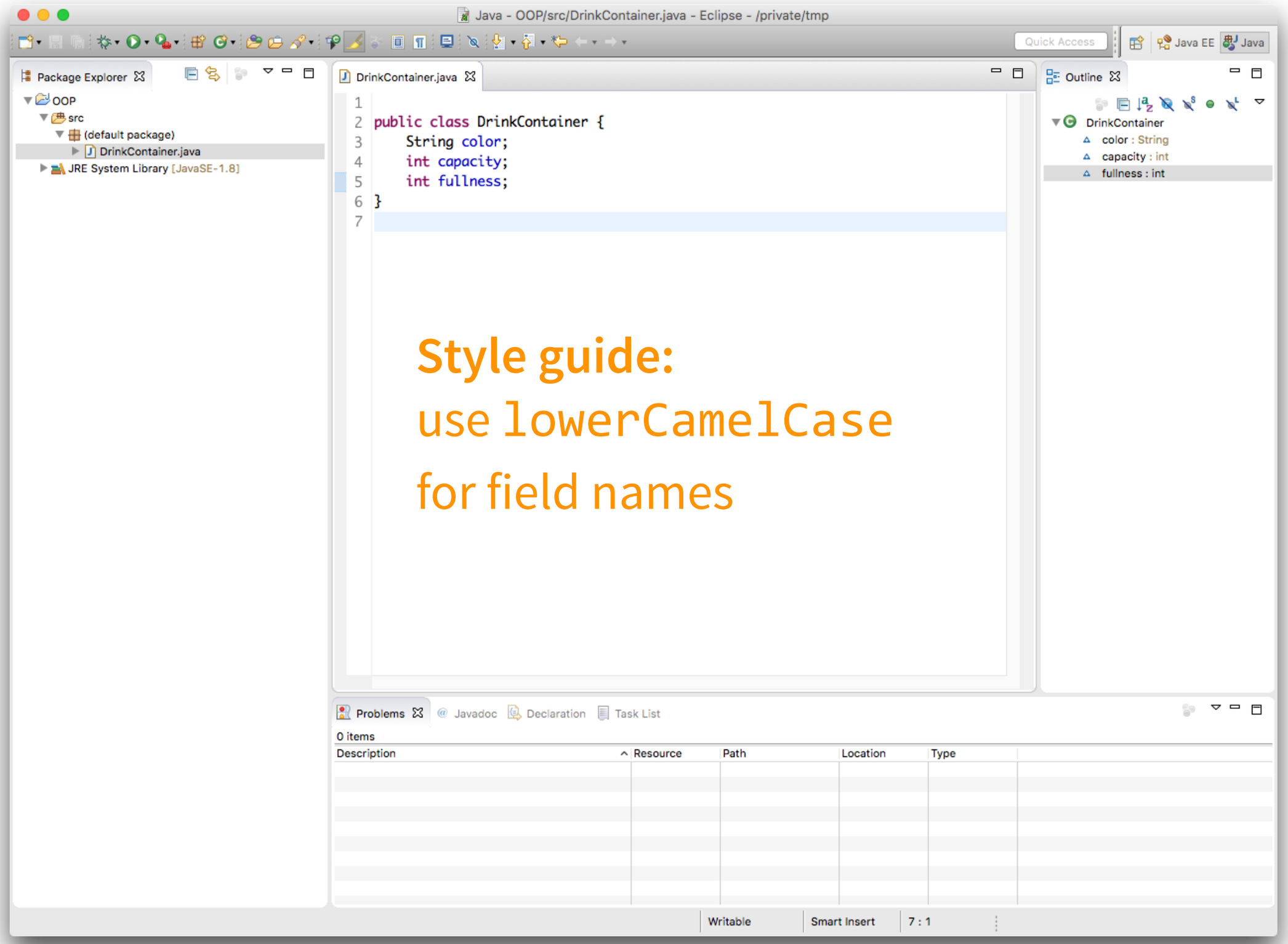
Give the class a good name



Style guide:
use `UpperCamelCase`
for class names

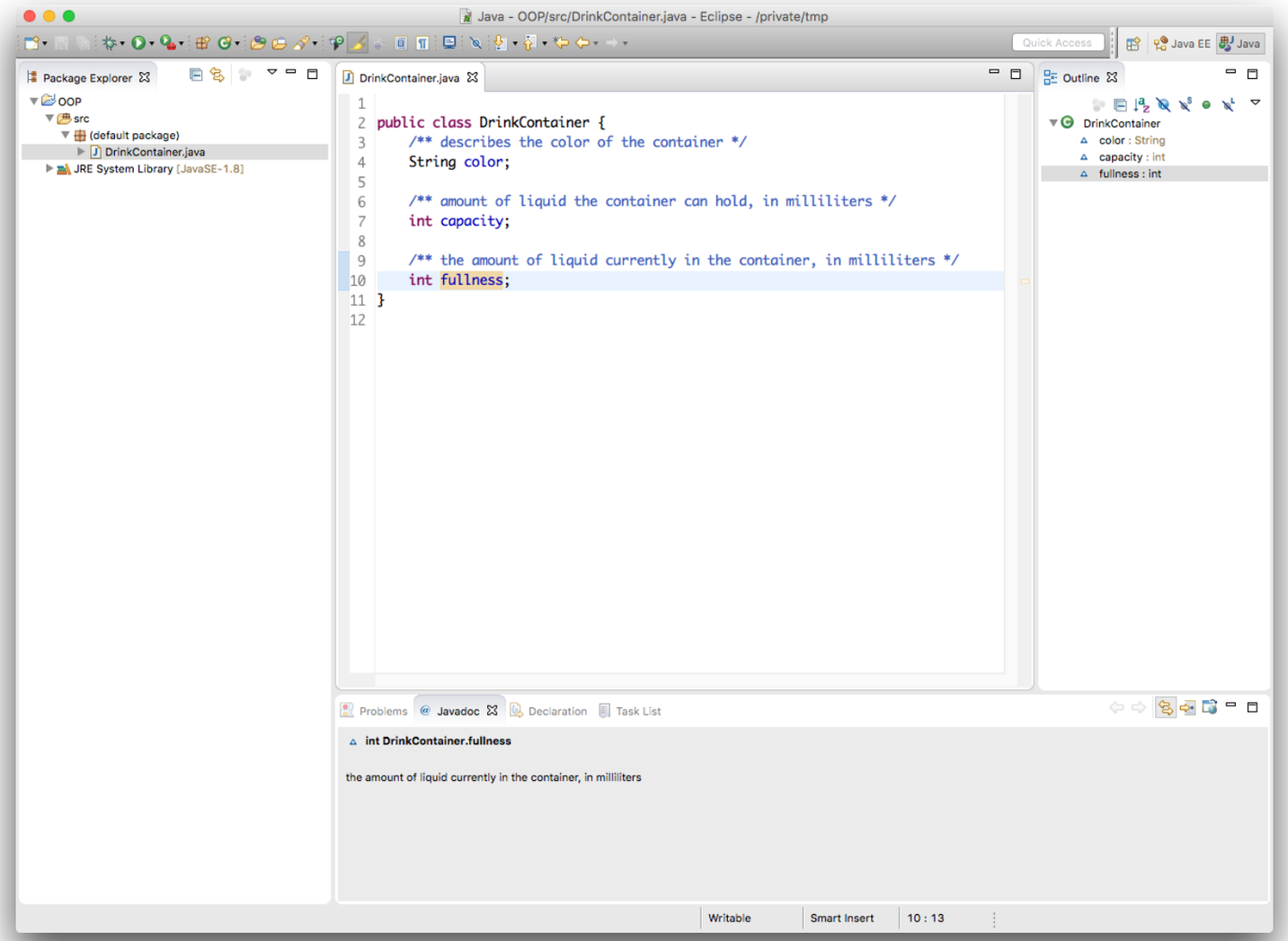


Field definitions go at top of class

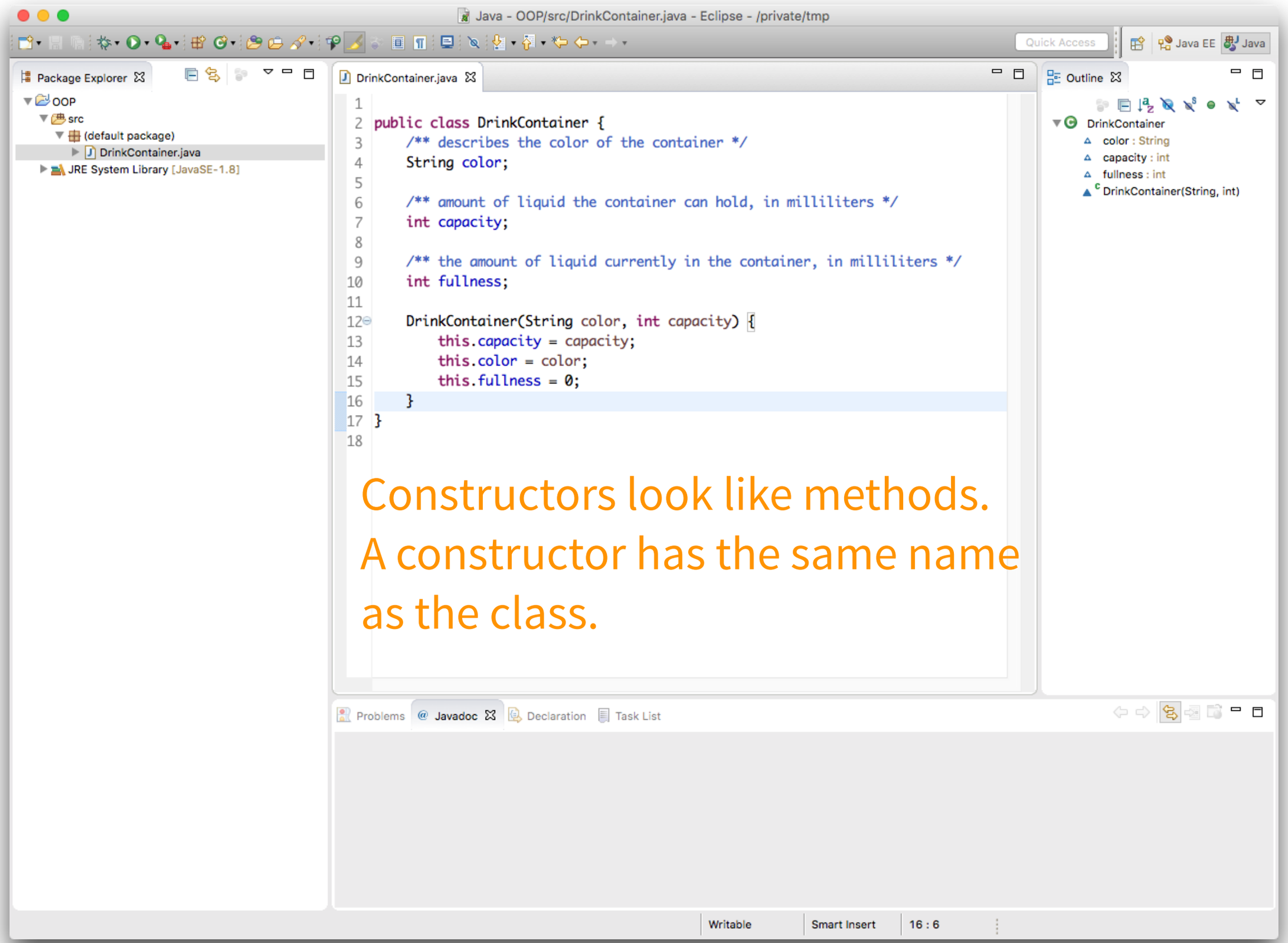


Good programming practice

Document your fields (using Javadoc).



A constructor initializes an object



Constructors look like methods.
A constructor has the same name as the class.

Good programming practice

Always use `this`.

It’s not a universally agreed-upon practice, but we’re going to follow it.