CS 10:
Problem solving via Object Oriented Programming

Introduction

Agenda

1. You, me, and this course

2. Dive into Object Oriented Programming (OOP)

Let's start with our backgrounds

Your background

- How did you satisfy the pre-reqs?
  - CS 1
  - ENGS 20
  - AP exam
  - Other

- CS majors? Minors? Not sure?

My background

This course is about solving problems with OOP, not simply how to program in Java

- Focus will be on solving problems with Object Oriented Programming (OOP), and you'll learn some Java along the way
- OOP is not the only way to solve problems, but it can be useful
- The course has three main components that overlap somewhat:
  1. Object Oriented Programming concepts and Java basics
  2. Abstract Data Types (ADTs) such as queues, stacks, trees, and graphs that form building blocks for solving problems (you'll see these ADTs again and again in CS)
  3. Solving wide range of real problems (graphics manipulation, characterize social networks, play Kevin Bacon game, compress files, analyze text…)
- You will learn far more by actually implementing things than you will by simply reading the material (or only attending lectures)

Material will be covered in lecture, section meetings, homework, and exams

Lectures:

- Come to class, read the course notes and find slides at: http://www.cs.dartmouth.edu/~tjp/cs10
- Laptop use in class in side seats only!

Section meetings (5%):

- You will be assigned to section with about 10 other students
- Section leader will reinforce concepts, answer questions, grade assignments
- Recitation session: 1 hour/week; programming drills (5%)
- Office hours: 3 hours/week

Homework (40%):

- Readings from *Data Structures and Algorithms in Java, 6th Edition*
- Short Assignments (SA) (10%) – due before next class
- Problem Sets (PS) (30%) – longer problems, generally 7-10 days, can work with a partner (note partner name on submission, or indicate no partner)

Exams (55%):

- Two midterms (15% each)
- One final (25%)

We will also be using Canvas and Piazza for announcements and help

Canvas

- Course announcements and homework submissions
- Section assignments
- Exam study material

Piazza (access via Canvas)

- Q&A forum
- Ask questions, get answers
- Don’t post code!

Let me know if you don’t have access!

Short Assignment 0 (SA-0) is out, complete survey before noon tomorrow

SA-0

- Find it on Canvas
- Take course survey to understand your background and assign you to a section
- Set up development environment
  - Instructions and screen shots provided on website
  - I'm using Eclipse 2018-12
- Create your first Java class
- Read course policies and honor code
- Please complete survey **before noon tomorrow** (or risk getting assigned to inconvenient section time!)

Agenda

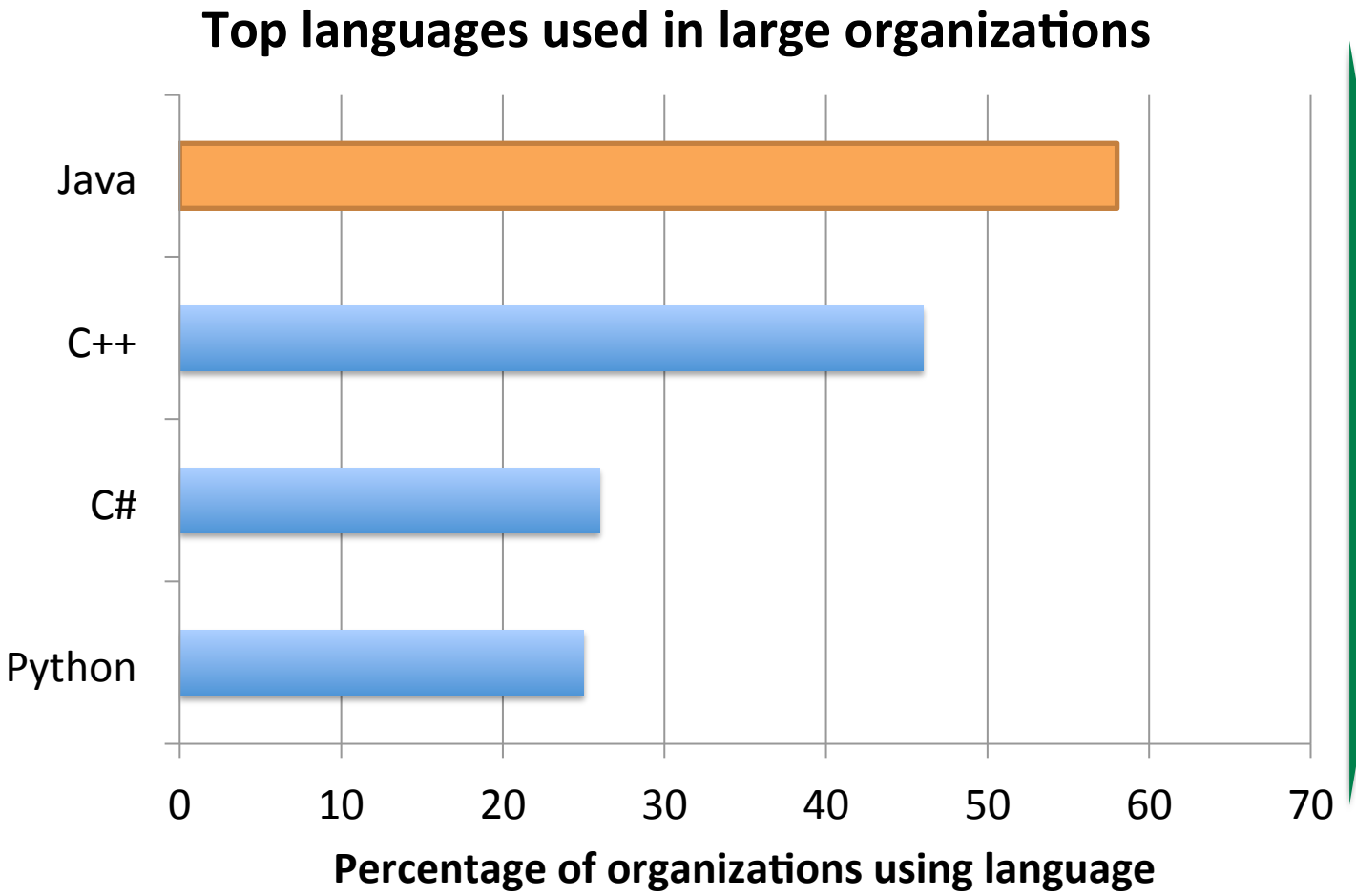
1. You, me, and this course
2. Dive into Object Oriented Programming (OOP)

OOP relies on four main pillars to create robust, adaptable, and reusable code

Four "pillars" of OOP

1. **Abstraction**
   - Boil complicated systems down to most fundamental parts
   - Name those parts and describe *what* they do, but not *how* they do it
   - Leads to Abstract Data Types (ADTs) – describes functionality (*interface* in Java); does not specify particular data structure to use in implementation
2. **Encapsulation**
   - Binds code (called *methods*) and data together into one self-contained "thing" (called an *object* in Java)
   - Objects (defined by *classes* in Java) implement an *interface* using specific data structures
   - Users of objects do not need to know exactly how the object works internally; generally trust that it works as expected
   - Example: can drive a car without knowing how an internal combustion engine works
3. **Inheritance**
   - Create "specialty" versions of objects that "inherit" functionality of parent, then customize behavior (more next class); reduces code redundancy
4. **Polymorphism**
   - Same name, multiple meanings (more next class)

OOP is popular, especially in large organizations



- Each of the most common languages is object oriented
- Java is particularly popular in large organizations

Source: https://www.cloudfoundry.org/wp-content/uploads/Developer-Language-Report_FINAL.pdf

Why is OOP in general, and Java in particular, so popular?

Approved answer: because it makes solving many types of problems easy (or perhaps easier)

Paul Graham's answer: it keeps mediocre programmers from doing too much damage

- In the real world, on a single project you may have dozens (or hundreds) of programmers working with thousands of objects – no one knows them all
- People come and go during the course of a non-trivial project – maintaining corporate knowledge is difficult
- We will see that objects can help prevent well-meaning people from making costly mistakes

We will be using Java, these things may blow your mind

Depending on your background, this may be weird:

- Must compile a program before it runs (so everything must be correct ahead of run time)
- Declare variables and can't change type
- White space/brackets
- For-each loops

Onward to OOP glory!

Image: http://siftd.net/?_escaped_fragment_=/content/24285/gameface-episode-52

In keeping with tradition, we'll start with "Hello world"

HelloWorld.java

1. Start Eclipse, create "cs10" Java Project (only need to do this one time)
2. Create "day1" Source folder to logically group your source code (e.g., "PS1" Source folder holds all the source code for Problem Set 1)
3. Create new "HelloWorld" class in "day1" source folder
   - File on disk is "HelloWorld.java"
   - Class Name is "HelloWorld"
   - Eclipse "stubs" out "main" method (where program execution starts)

Other items of note:

Javadoc

- Java documentation feature
- Enter description for Class or method
- Starts with "/**", ends with "*/"
- Can add tags such as "@author" or "@param"

main() is where action starts

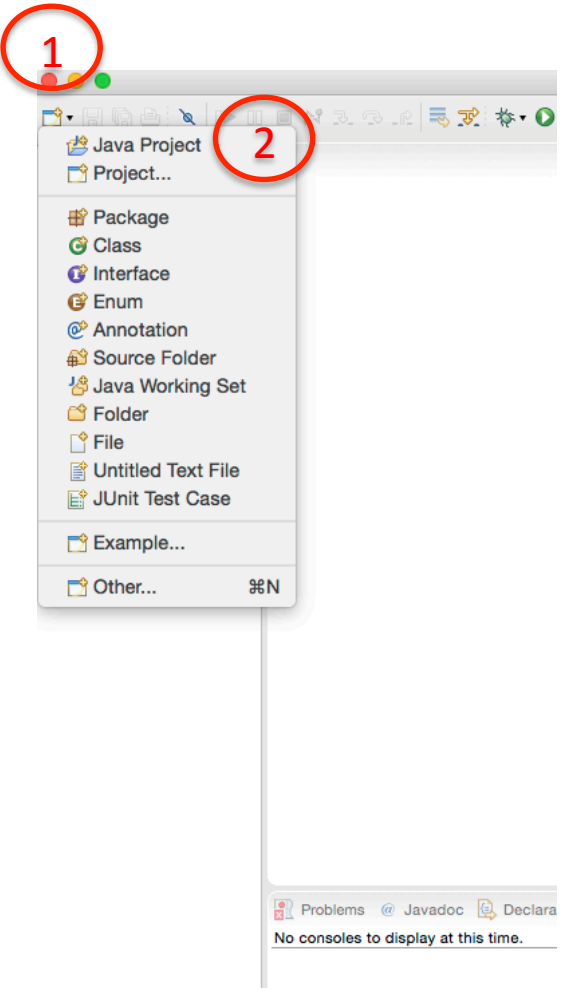
Add System.out.println("Hello World") to output to the console

Press green "Run" button to run (might need Run dropdown "run as")

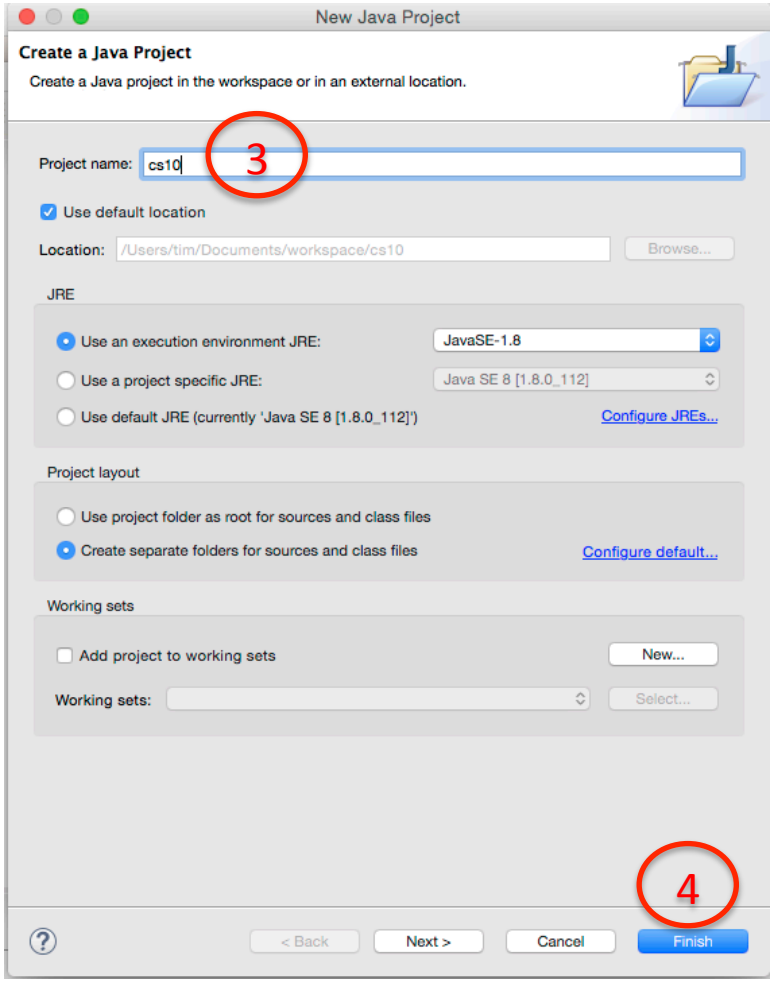
Can run using "Debug" button

1. Create “cs10” Java Project (only need to do this one time)

1. Click “New” dropdown
2. Select “Java Project”

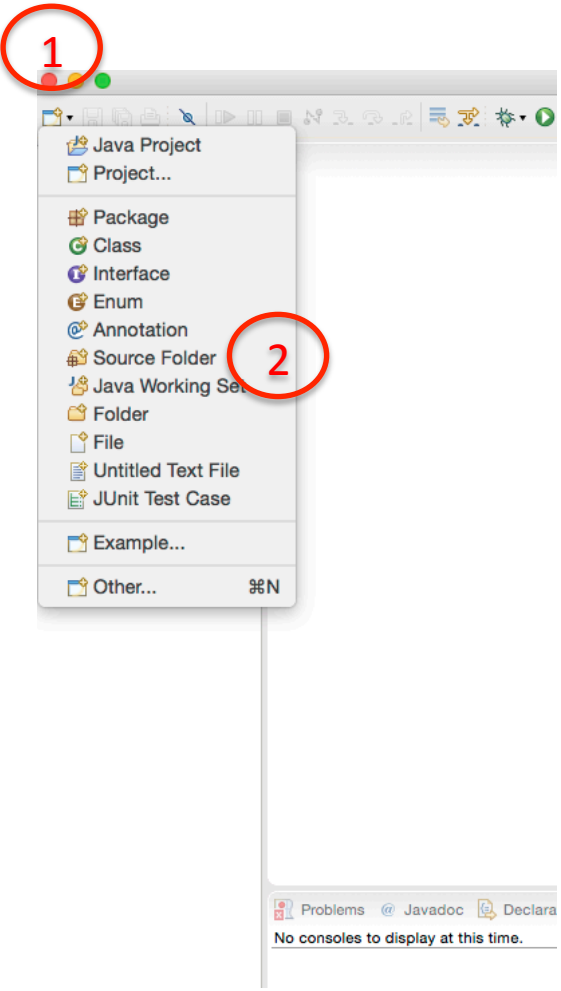


3. Enter “cs10” as Project name
4. Click “Finish”

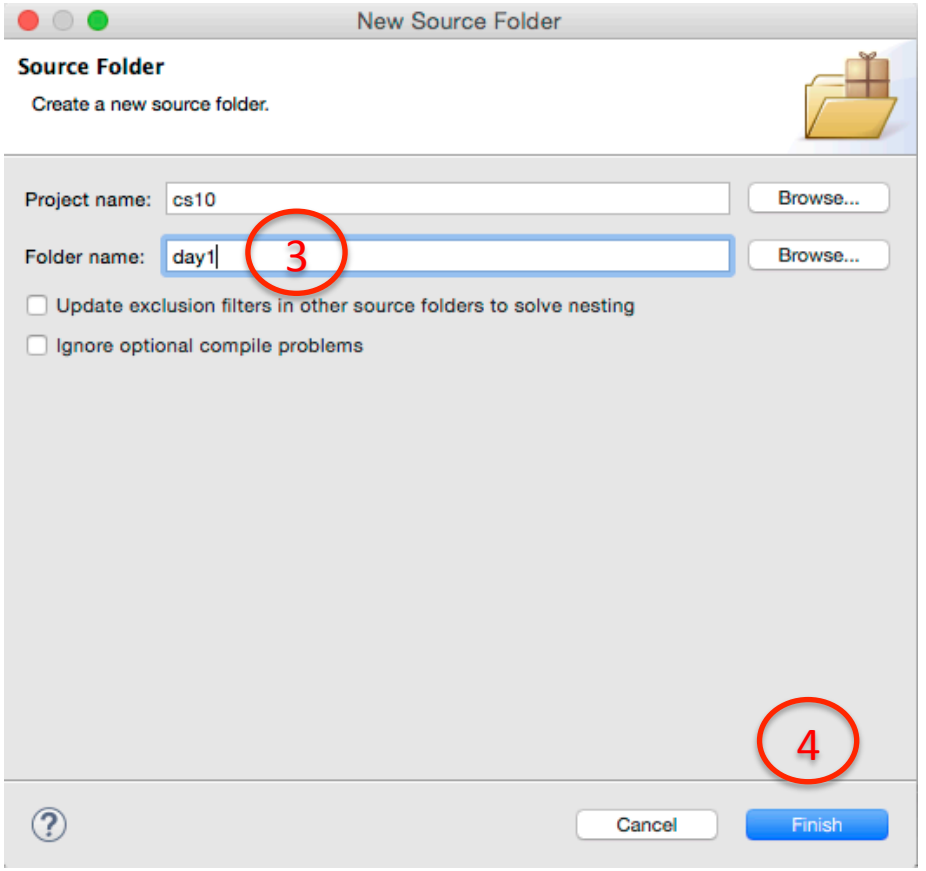


2. Create “day1” Source folder to hold your source code for day one of class

1. Click “New” dropdown
2. Select “Source Folder”

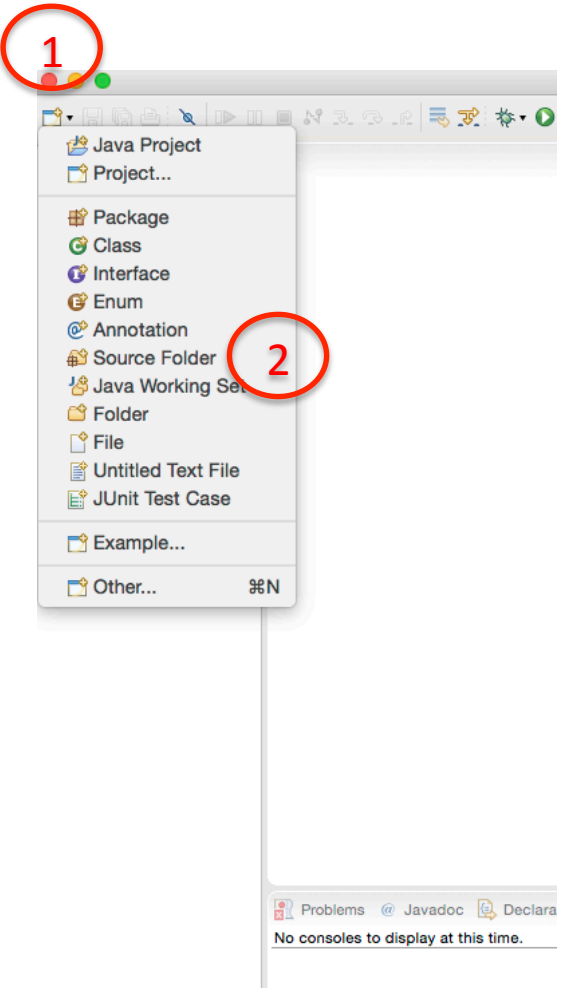


3. Enter Folder name (“day1”)
4. Click “Finish”

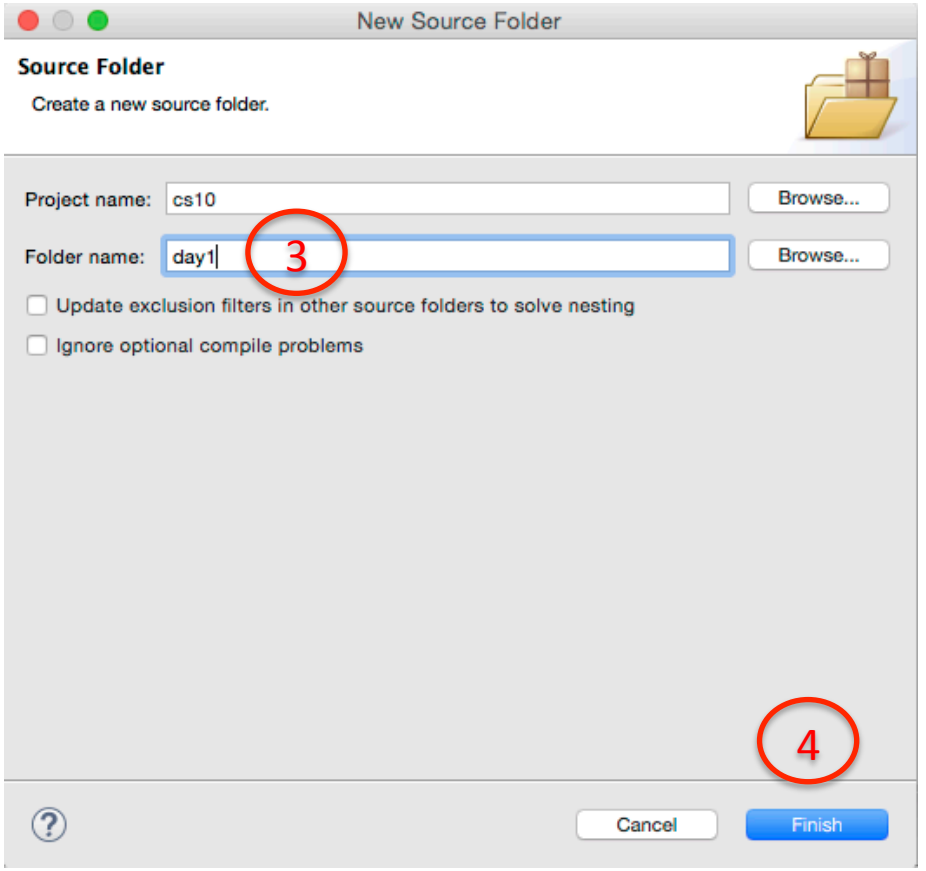


2. Create “day1” Source folder to hold your source code for day one of class

1. Click “New” dropdown
2. Select “Source Folder”



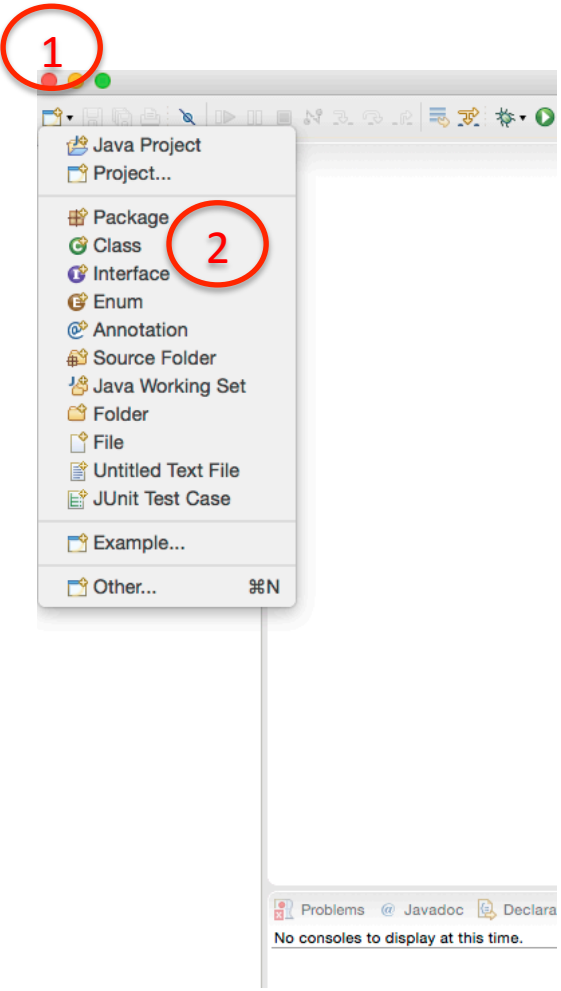
3. Enter Folder name (“day1”)
4. Click “Finish”



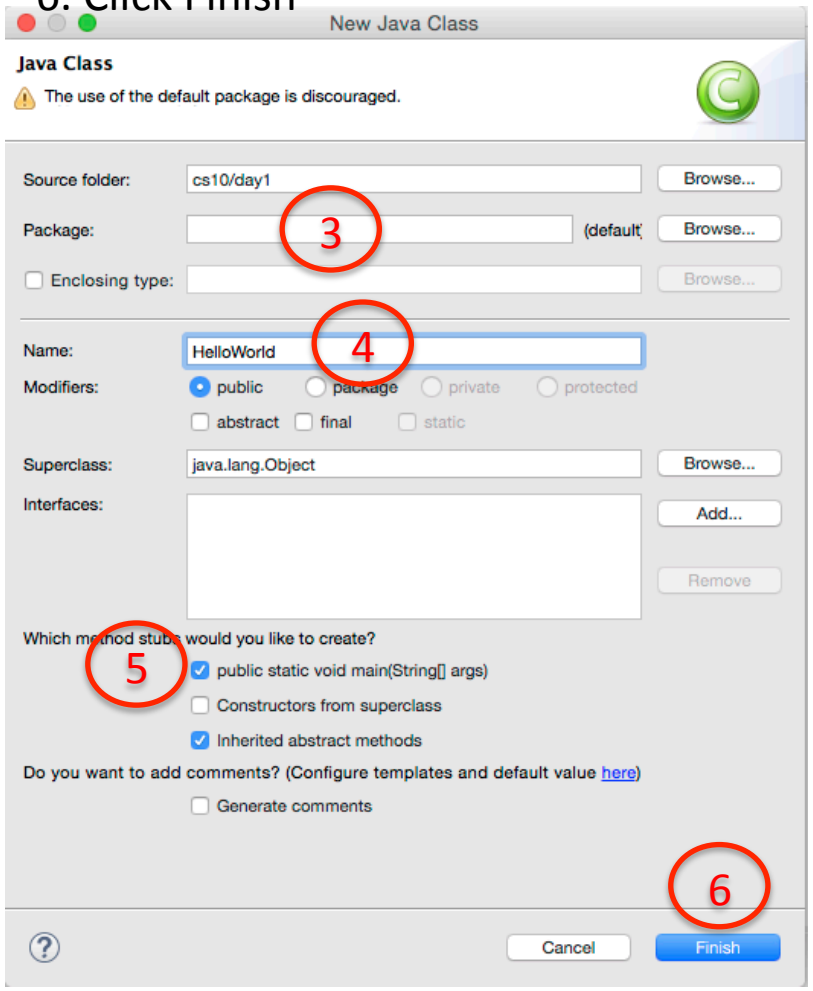
Source folders are a useful way to organize your code (ex. PS1 Source folder contains all code for Problem Set 1)

3. Create new “HelloWorld” class in “day1” source folder

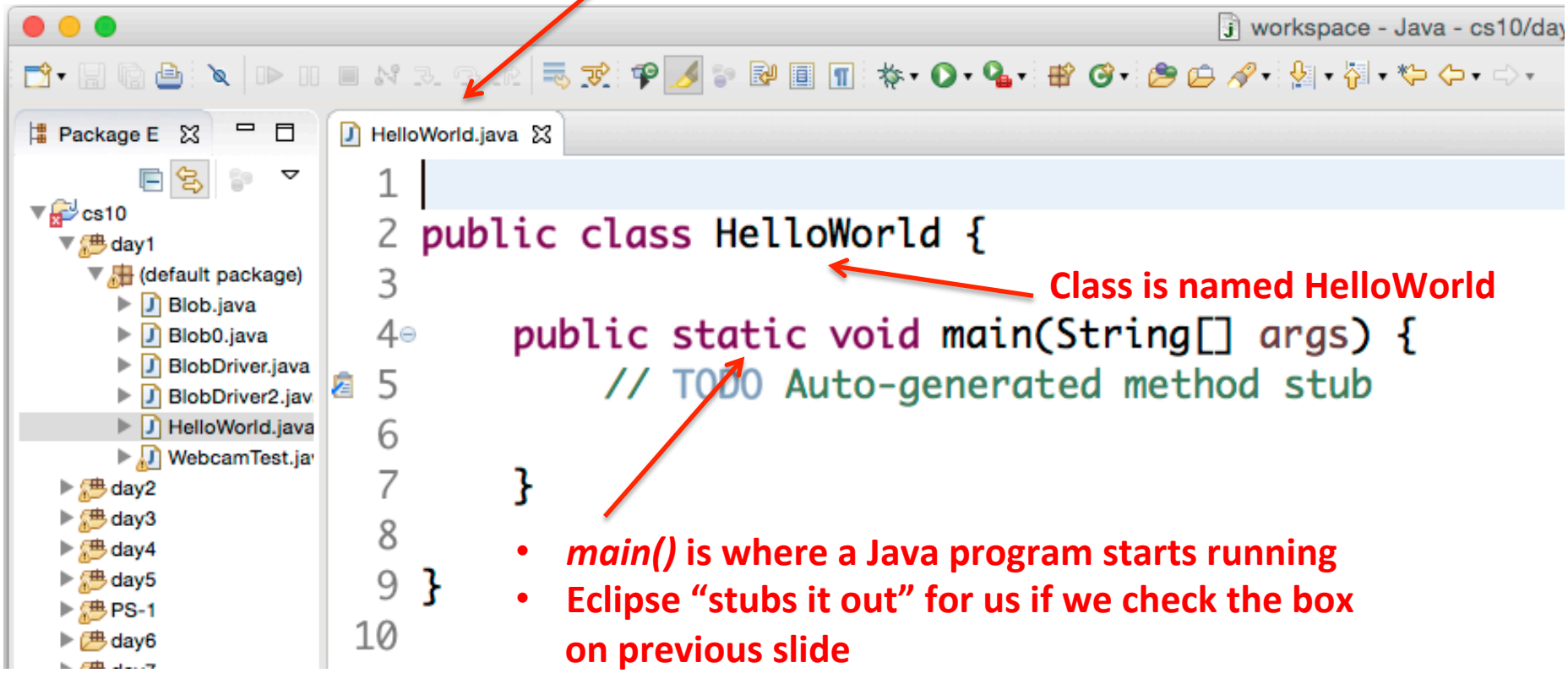
1. Click “New” dropdown
2. Select “Class”



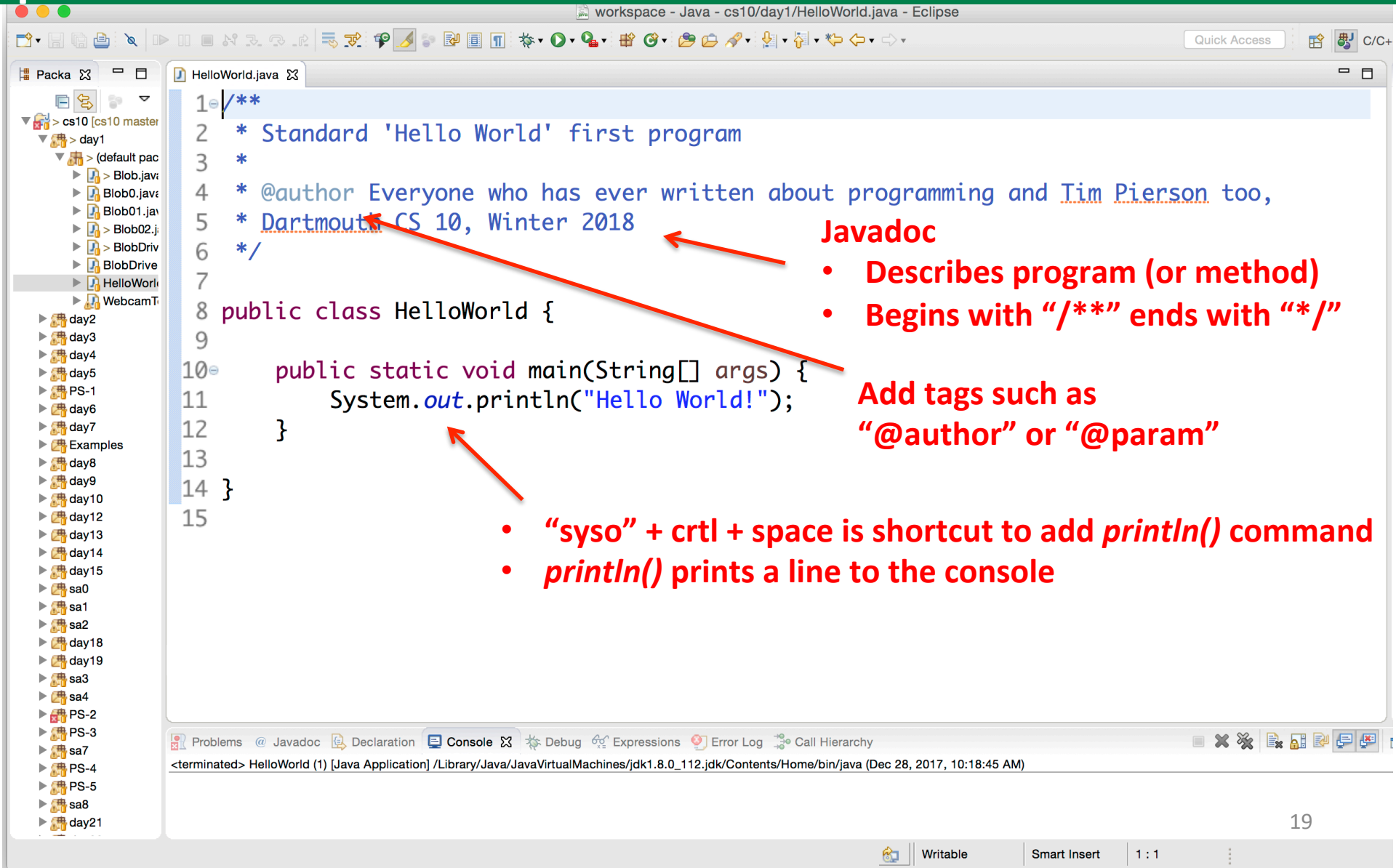
3. Leave Package empty
4. Enter Class Name “HelloWorld”
5. Check “public static void main…”
6. Click Finish



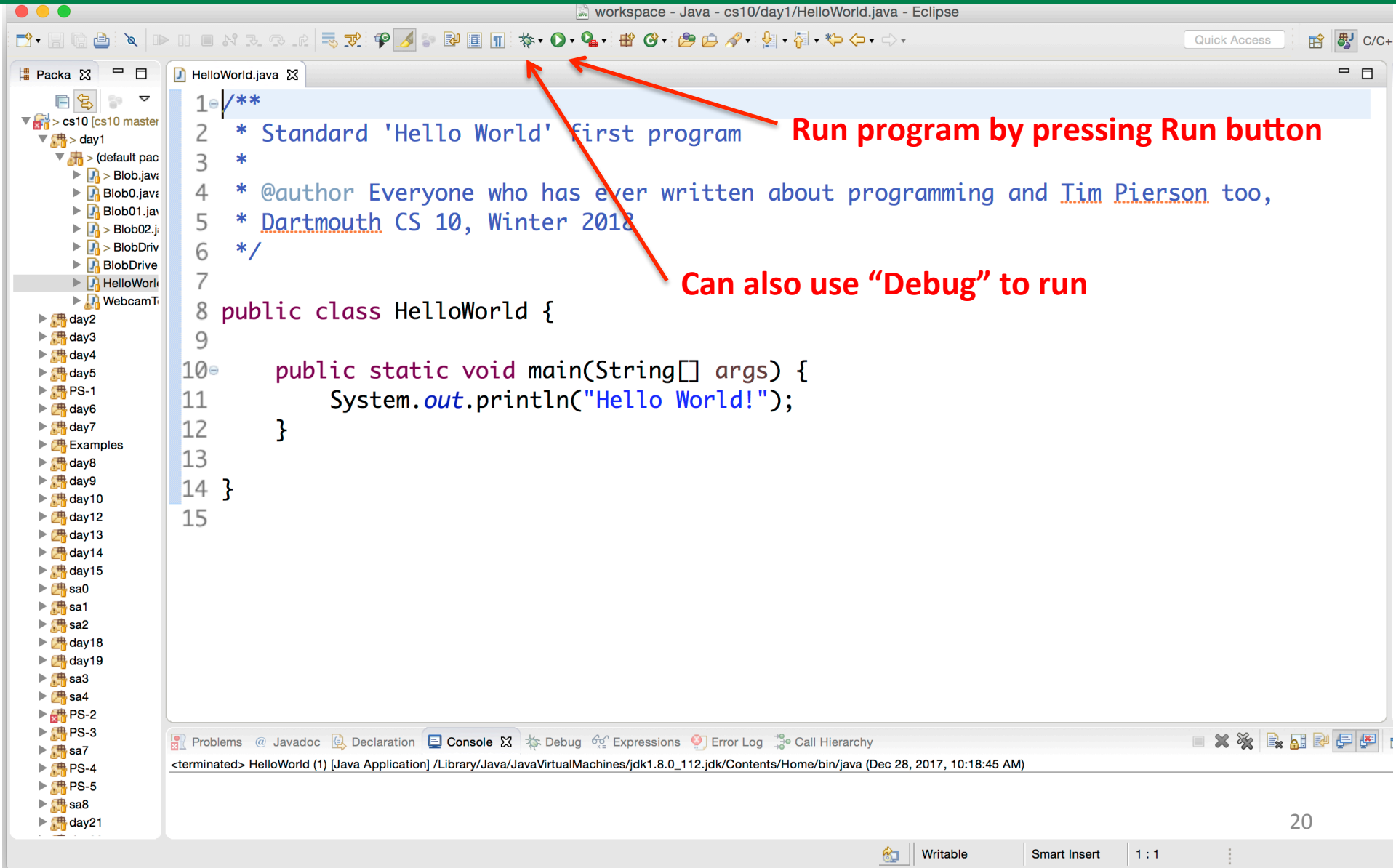
Eclipse creates HelloWorld.java "boilerplate" code



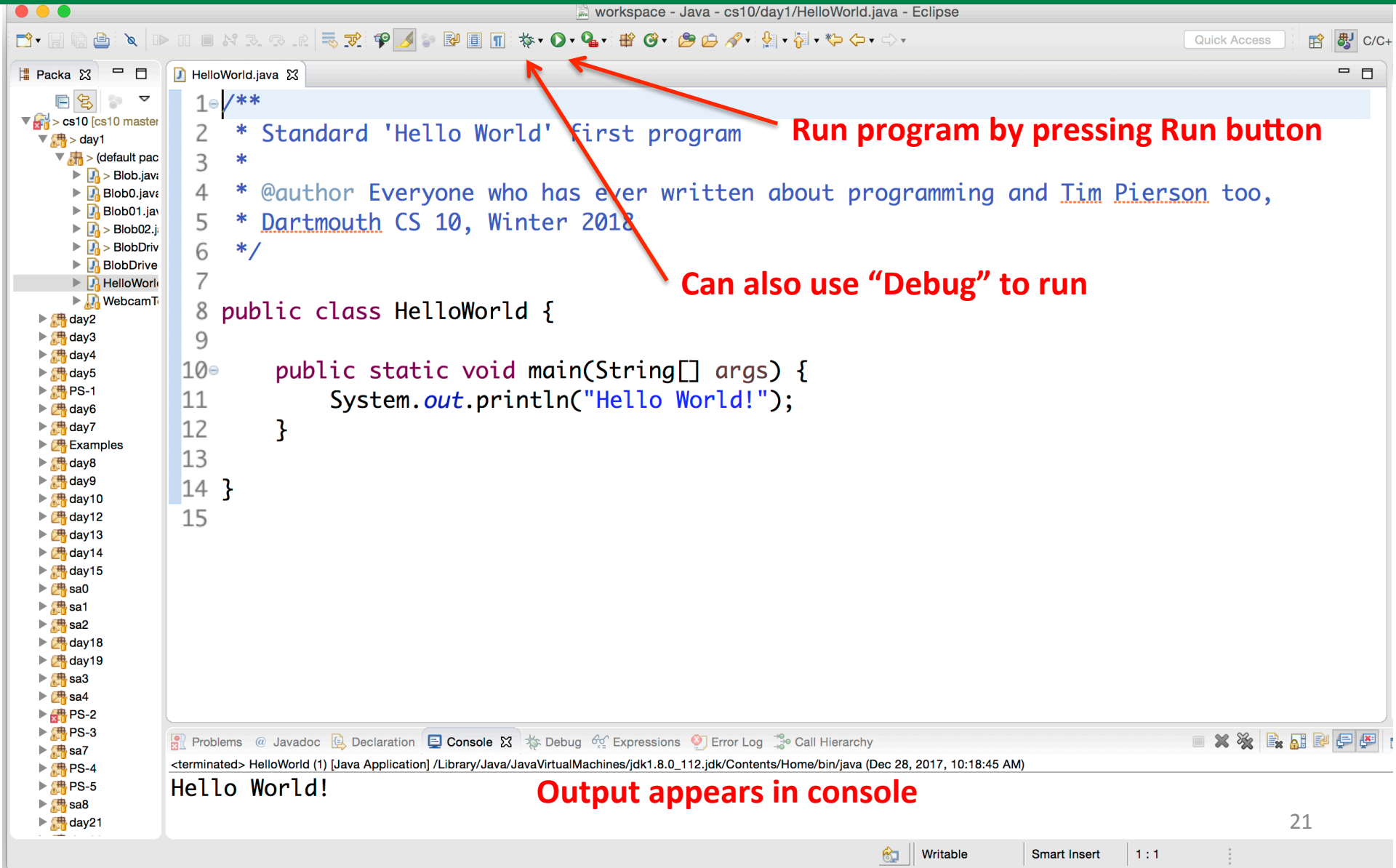
We can flesh out the boilerplate code to print “Hello World!” to the console



Running the program prints “Hello World!” to console



Running the program prints “Hello World!” to console



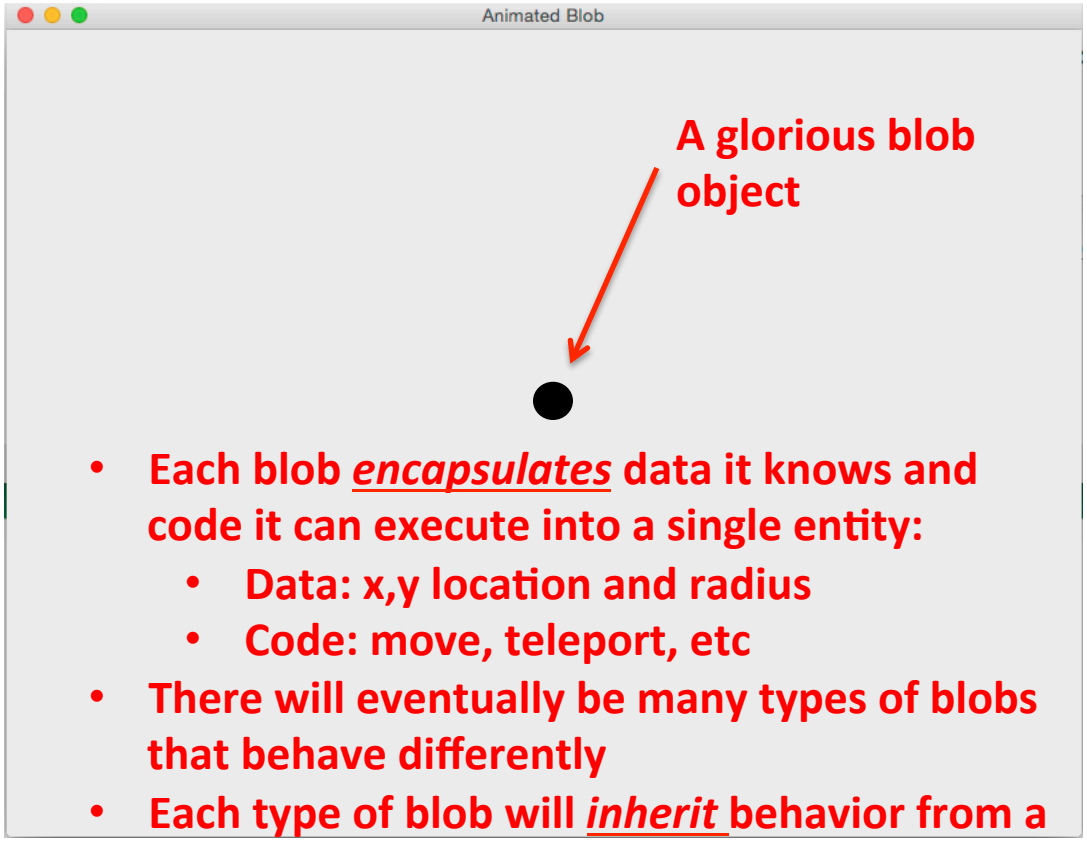
Today we will focus on encapsulation

Encapsulation

- Binds code (methods) and data together into one self-contained “thing”, called an object in Java
- Each object has its own data about itself (e.g., x,y screen coordinates)
- Objects can make data about itself public or private
- Private data allows an object to control access to data from outside (e.g., if private, then only the object itself can alter its internal data)

We start with different types of "blob" objects that will move around the screen

Blobs are graphical objects that move around the screen

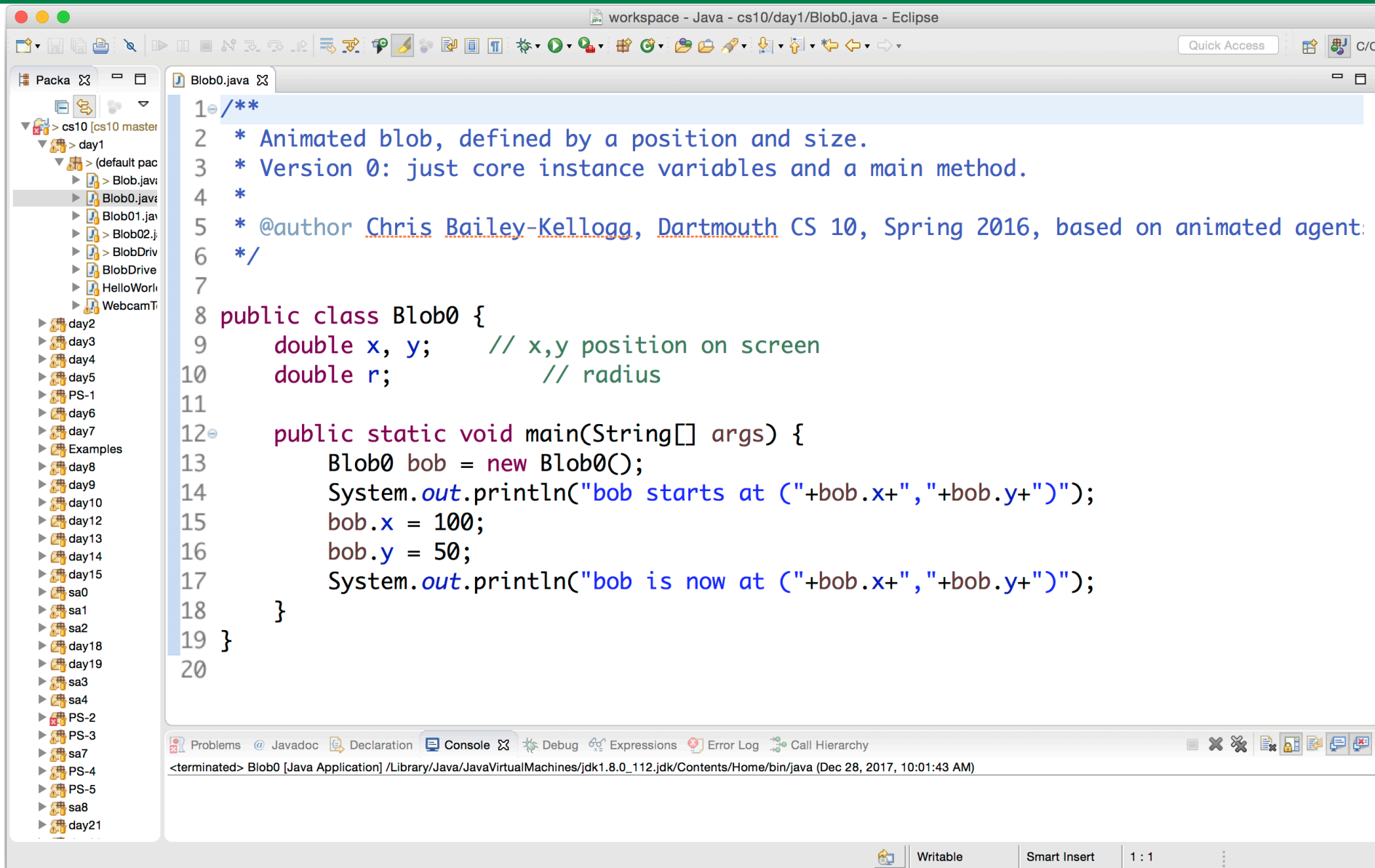


- **Each blob *encapsulates* data it knows and code it can execute into a single entity:**
  - **Data: x,y location and radius**
  - **Code: move, teleport, etc**
- **There will eventually be many types of blobs that behave differently**
- **Each type of blob will *inherit* behavior from a *base class***

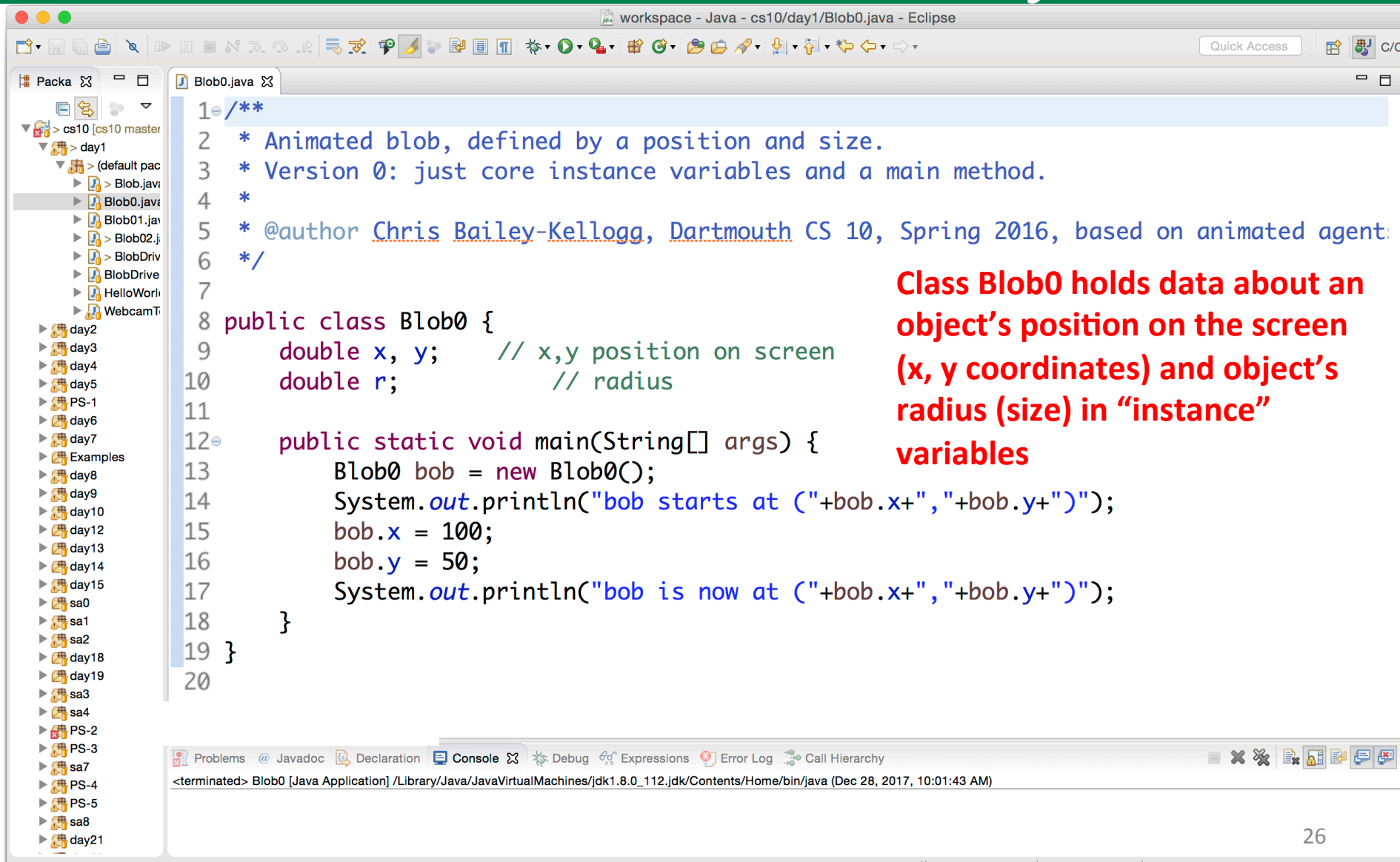
- We will model blobs as *objects*
- Objects encapsulate:
  - Data they know (e.g., x,y location and radius)
  - Actions they can take (e.g, move, teleport) called *methods*
- Objects are defined by a *class*
  - Like a blueprint – a class tells how to create an object (such as a house)
  - Does not itself create objects
- Each object is *instantiated* (created) from the class in Java using the "new" keyword
- There can be many objects created from the same class (like there can be many houses built from the same blueprint)


ENOUGH TALK
SHOW US HOW IT WORKS
memegenerator.net

Blob0.java

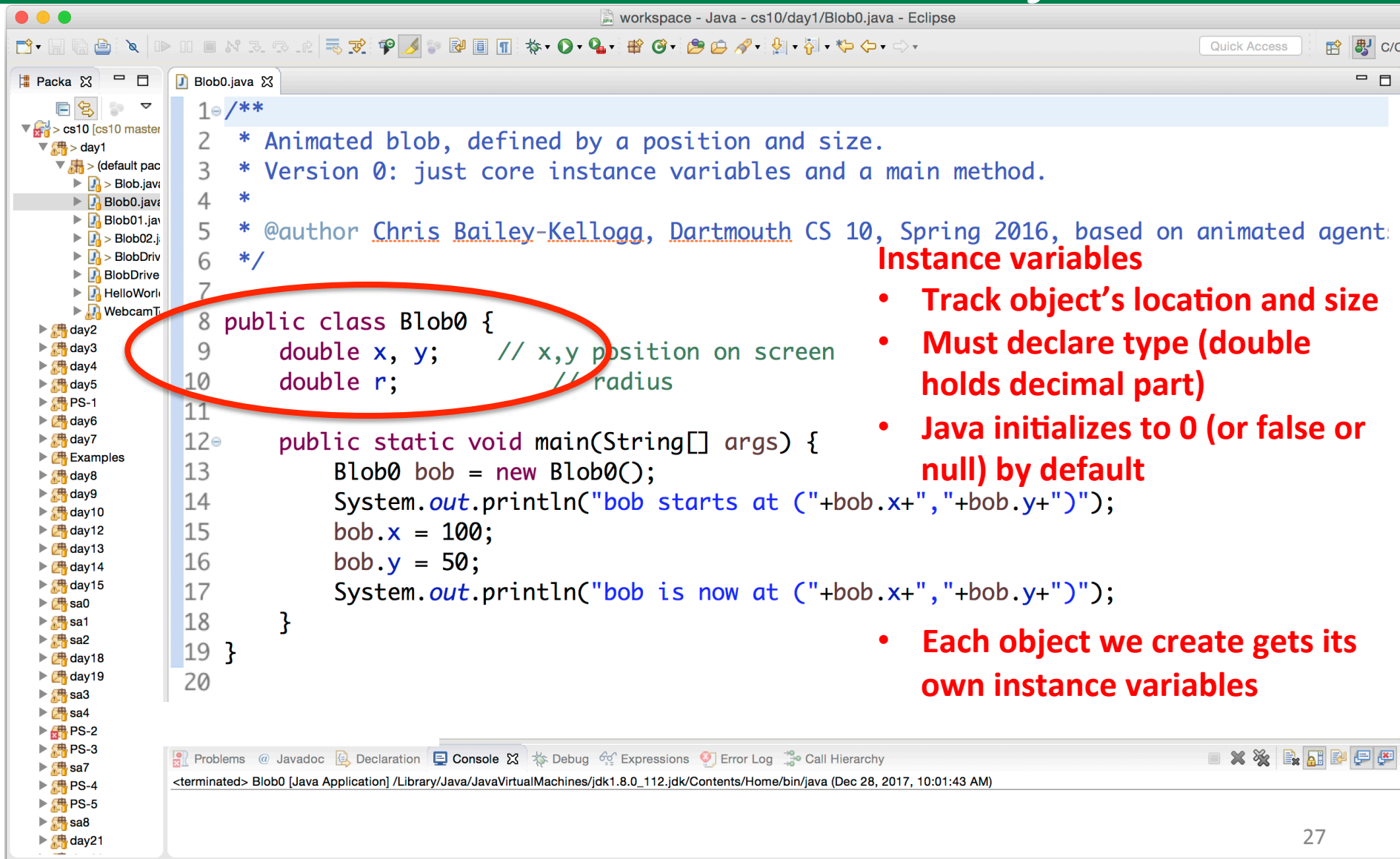

workspace - Java - cs10/day1/Blob0.java - Eclipse
Quick Access
Packa
Blob0.java
> cs10 [cs10 master
> day1
> (default pac
> Blob.java
Blob0.java
Blob01.jav
> Blob02.j
> BlobDriv
BlobDrive
HelloWorl
WebcamT
day2
day3
day4
day5
PS-1
day6
day7
Examples
day8
day9
day10
day12
day13
day14
day15
sa0
sa1
sa2
day18
day19
sa3
sa4
PS-2
PS-3
sa7
PS-4
PS-5
sa8
day21
1 /**
2 * Animated blob, defined by a position and size.
3 * Version 0: just core instance variables and a main method.
4 *
5 * @author Chris Bailey-Kellogg, Dartmouth CS 10, Spring 2016, based on animated agent
6 */
7
8 public class Blob0 {
9 double x, y; // x,y position on screen
10 double r; // radius
11
12 public static void main(String[] args) {
13 Blob0 bob = new Blob0();
14 System.out.println("bob starts at ("+bob.x+","+bob.y+")");
15 bob.x = 100;
16 bob.y = 50;
17 System.out.println("bob is now at ("+bob.x+","+bob.y+")");
18 }
19 }
20
Problems
Javadoc
Declaration
Console
Debug
Expressions
Error Log
Call Hierarchy
<terminated> Blob0 [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Dec 28, 2017, 10:01:43 AM)
Writable
Smart Insert
1 : 1

Blob0: Our first "real" class uses instance variables to store data about objects



Class Blob0 holds data about an object's position on the screen (x, y coordinates) and object's radius (size) in "instance" variables

Blob0: Our first "real" class uses instance variables to store data about objects

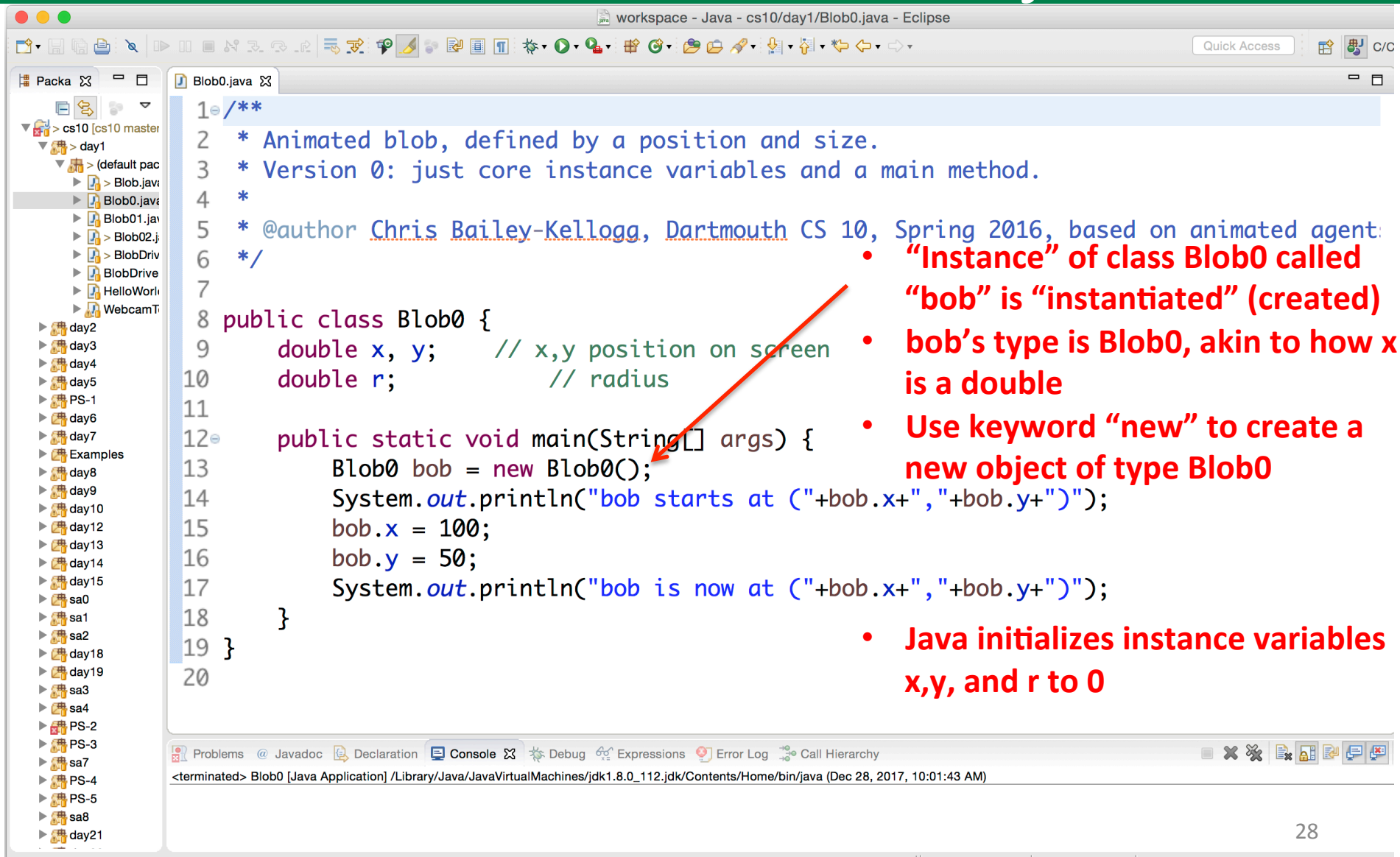


Instance variables

- **Track object's location and size**
- **Must declare type (double holds decimal part)**
- **Java initializes to 0 (or false or null) by default**

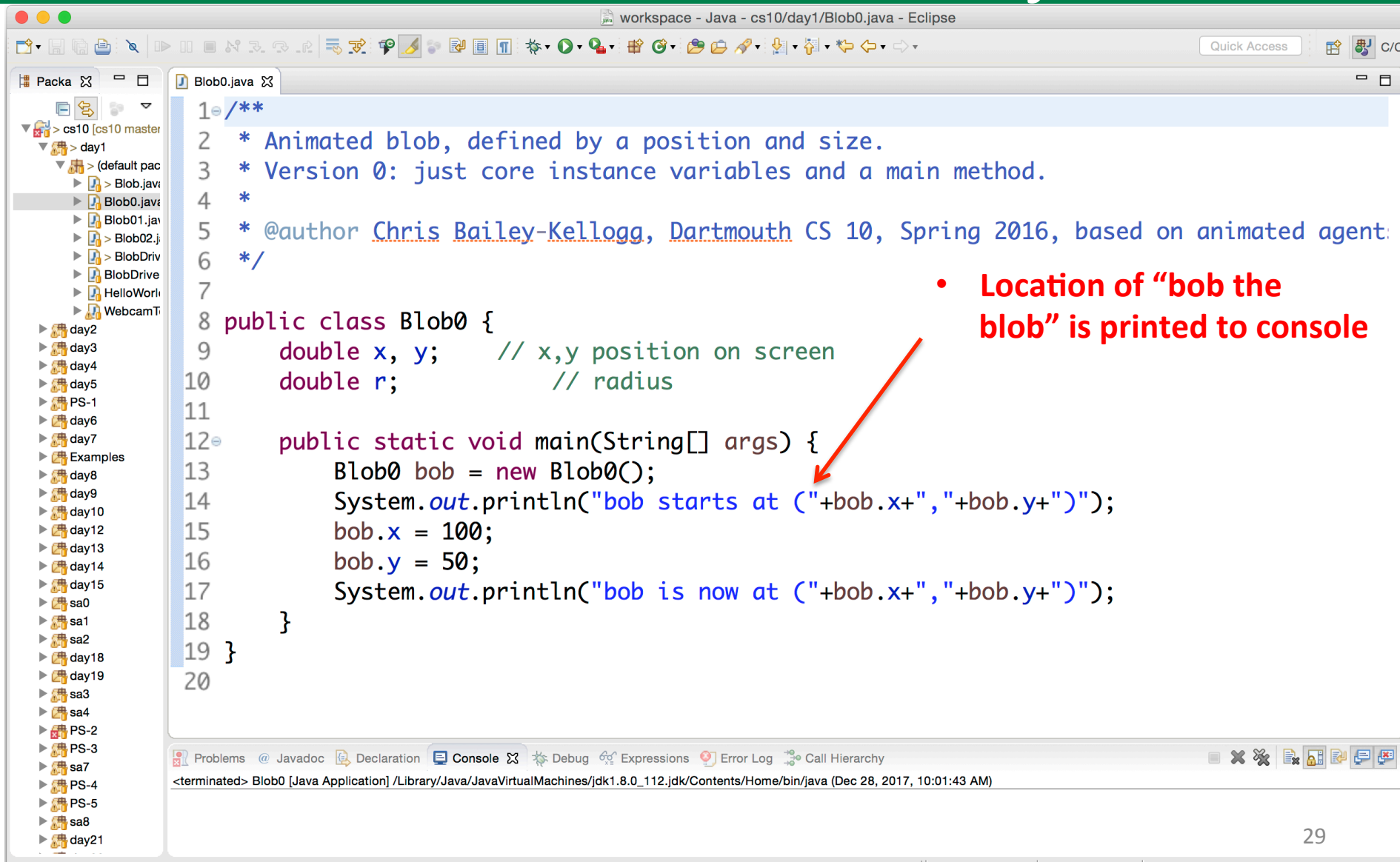
- **Each object we create gets its own instance variables**

Blob0: Our first "real" class uses instance variables to store data about objects



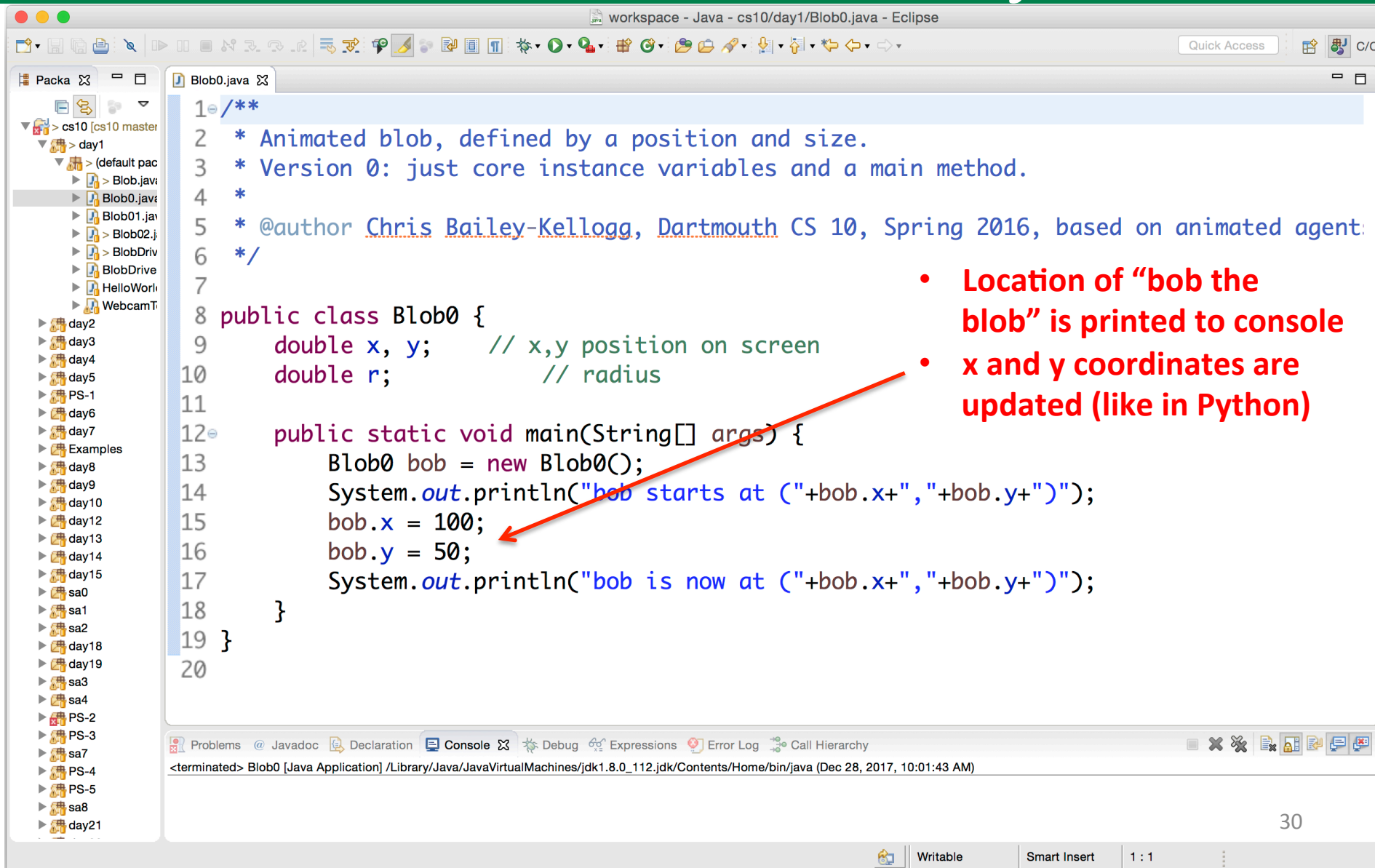
- "Instance" of class Blob0 called "bob" is "instantiated" (created)
- bob's type is Blob0, akin to how x is a double
- Use keyword "new" to create a new object of type Blob0
- Java initializes instance variables x,y, and r to 0

Blob0: Our first "real" class uses instance variables to store data about objects



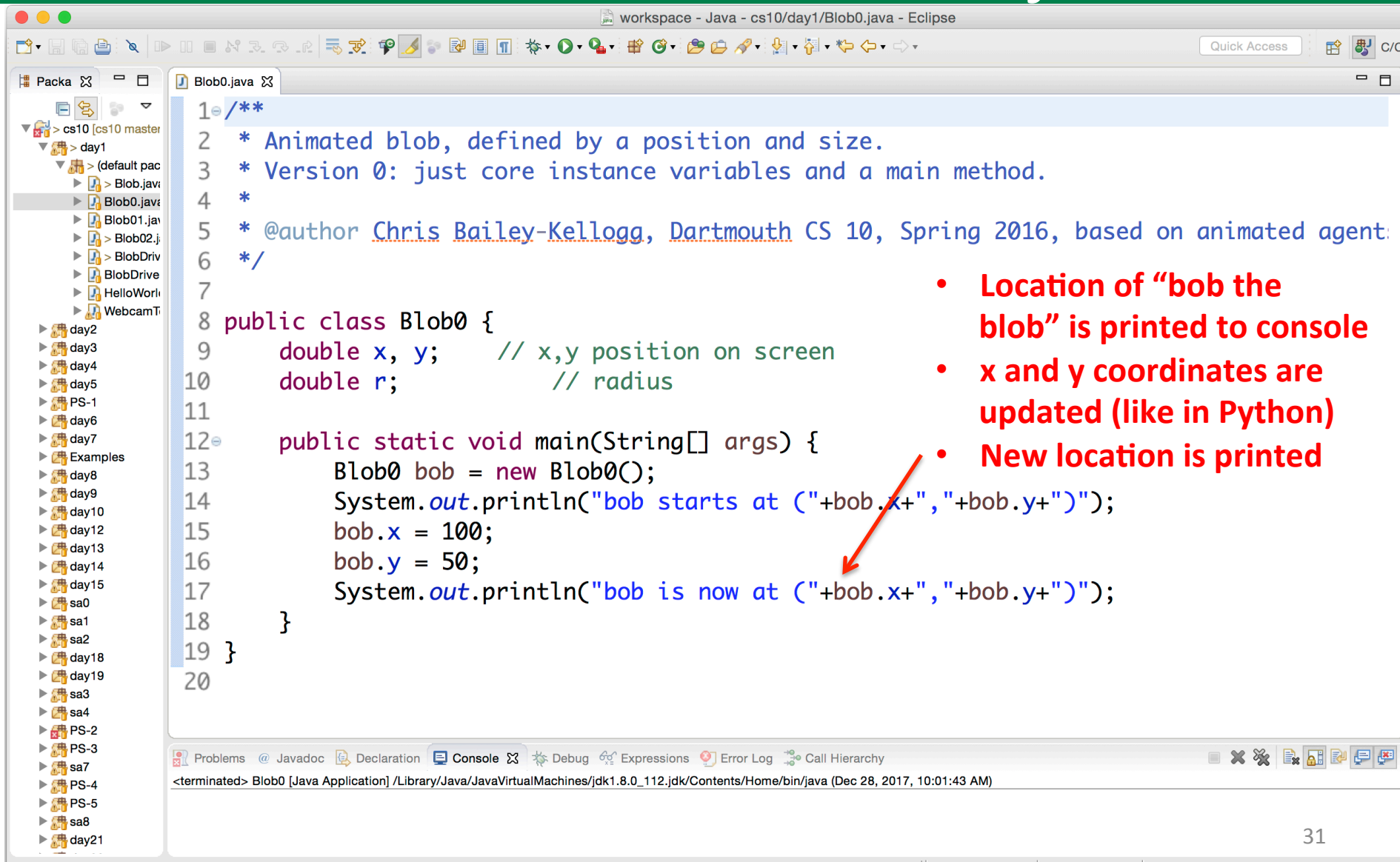
- **Location of "bob the blob" is printed to console**

Blob0: Our first "real" class uses instance variables to store data about objects



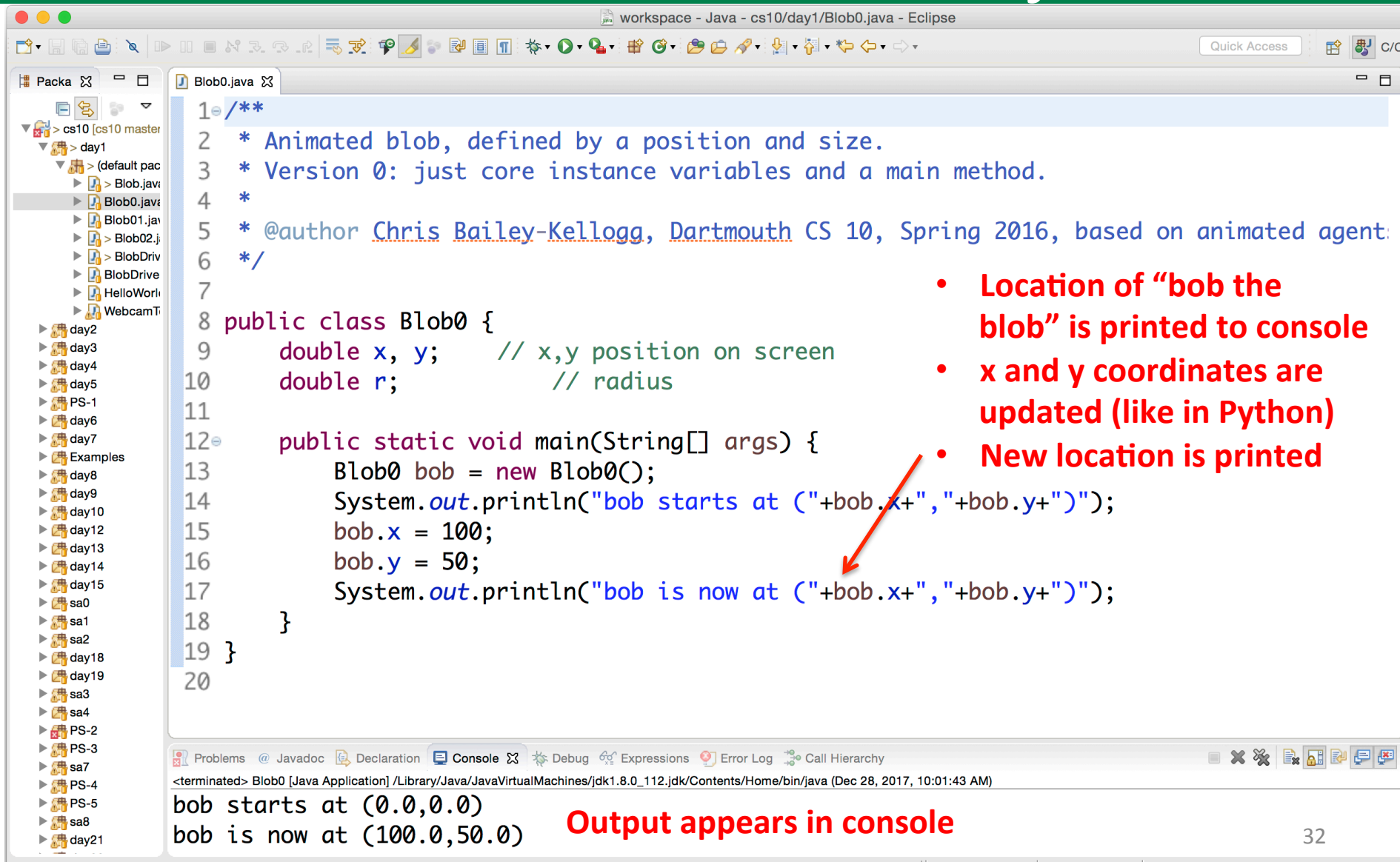
- **Location of "bob the blob" is printed to console**
- **x and y coordinates are updated (like in Python)**

Blob0: Our first "real" class uses instance variables to store data about objects



- Location of "bob the blob" is printed to console
- x and y coordinates are updated (like in Python)
- New location is printed

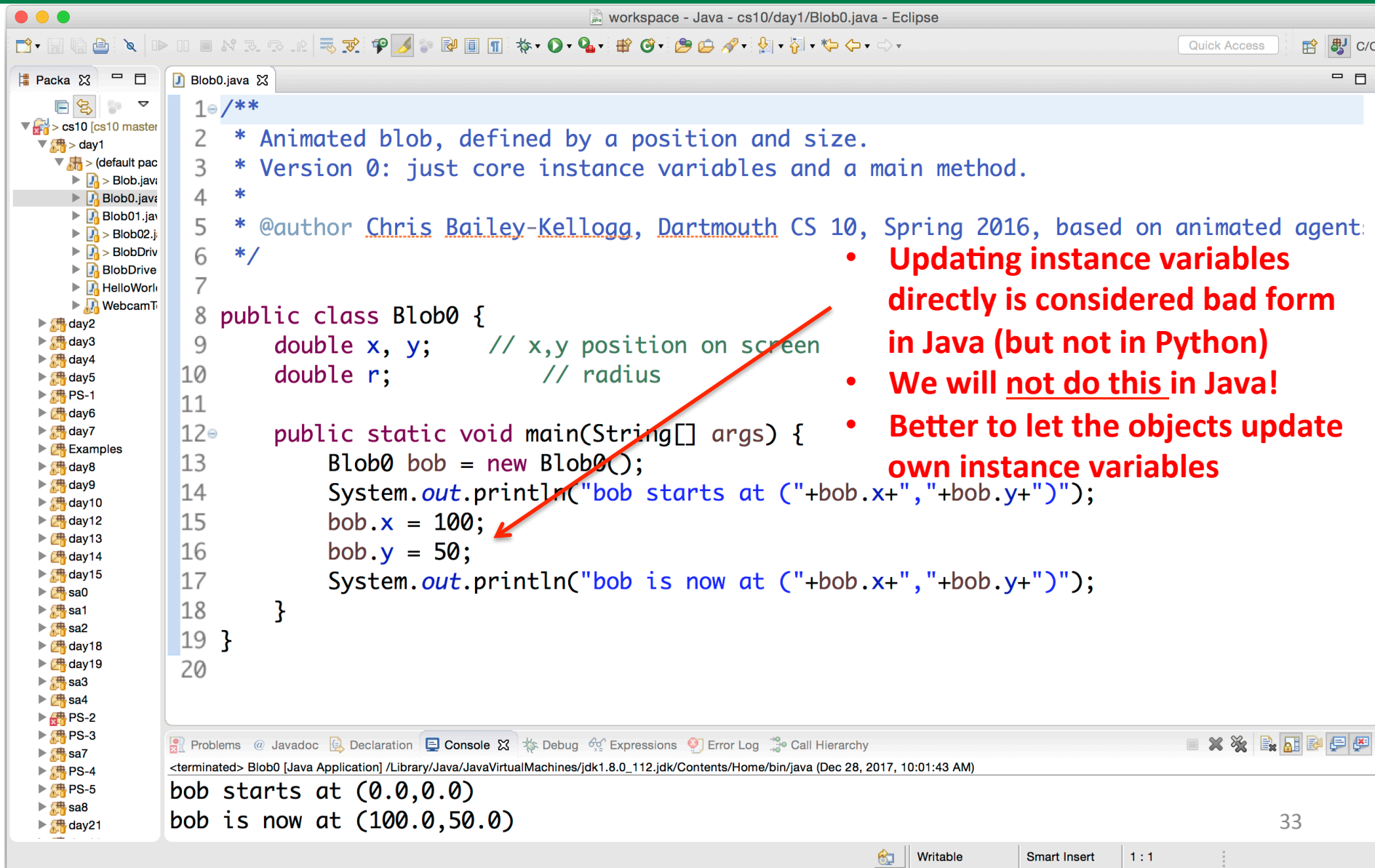
Blob0: Our first "real" class uses instance variables to store data about objects



- **Location of "bob the blob" is printed to console**
- **x and y coordinates are updated (like in Python)**
- **New location is printed**

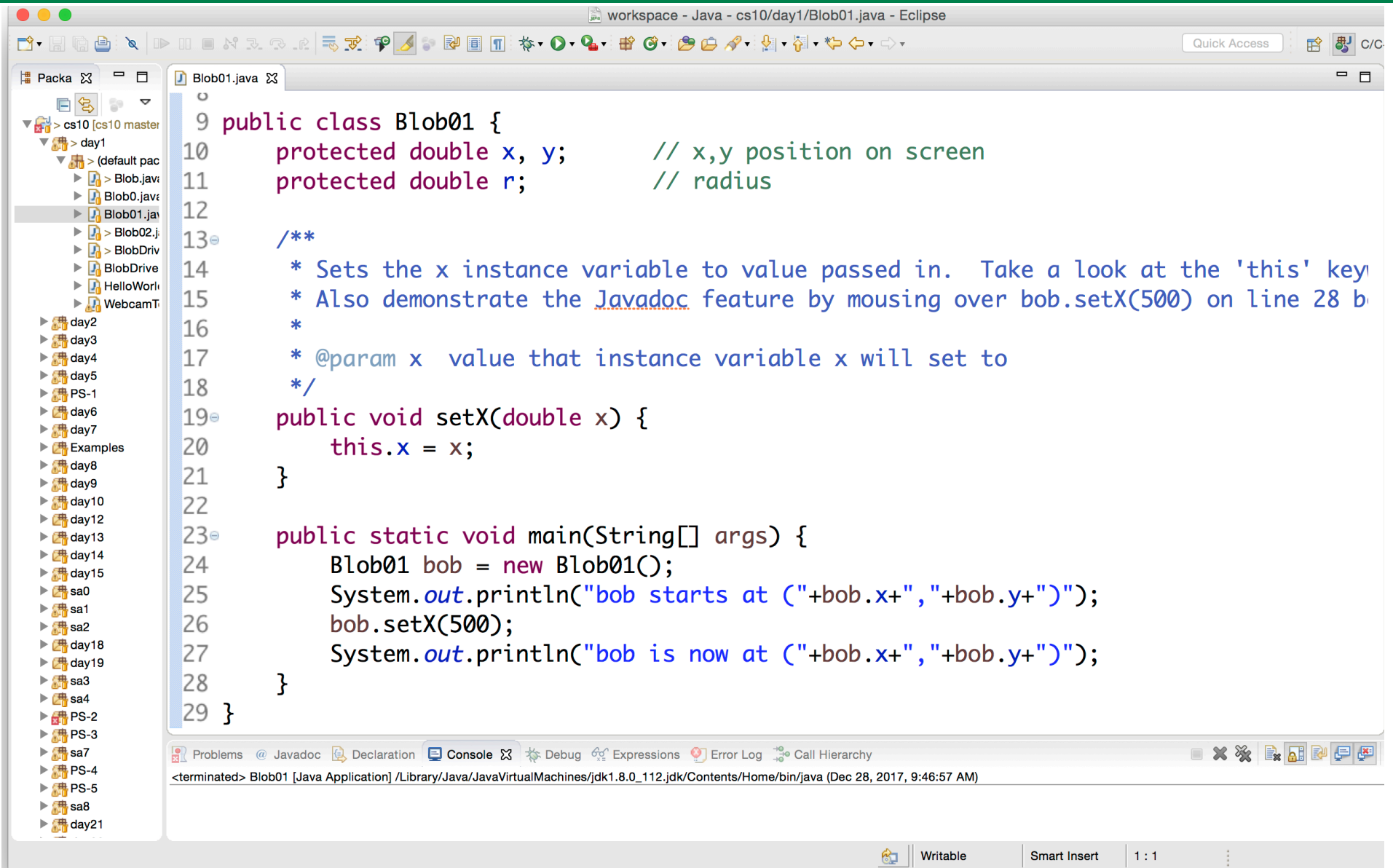
Output appears in console

Blob0: Directly updating instance variables is bad form – we can do better!

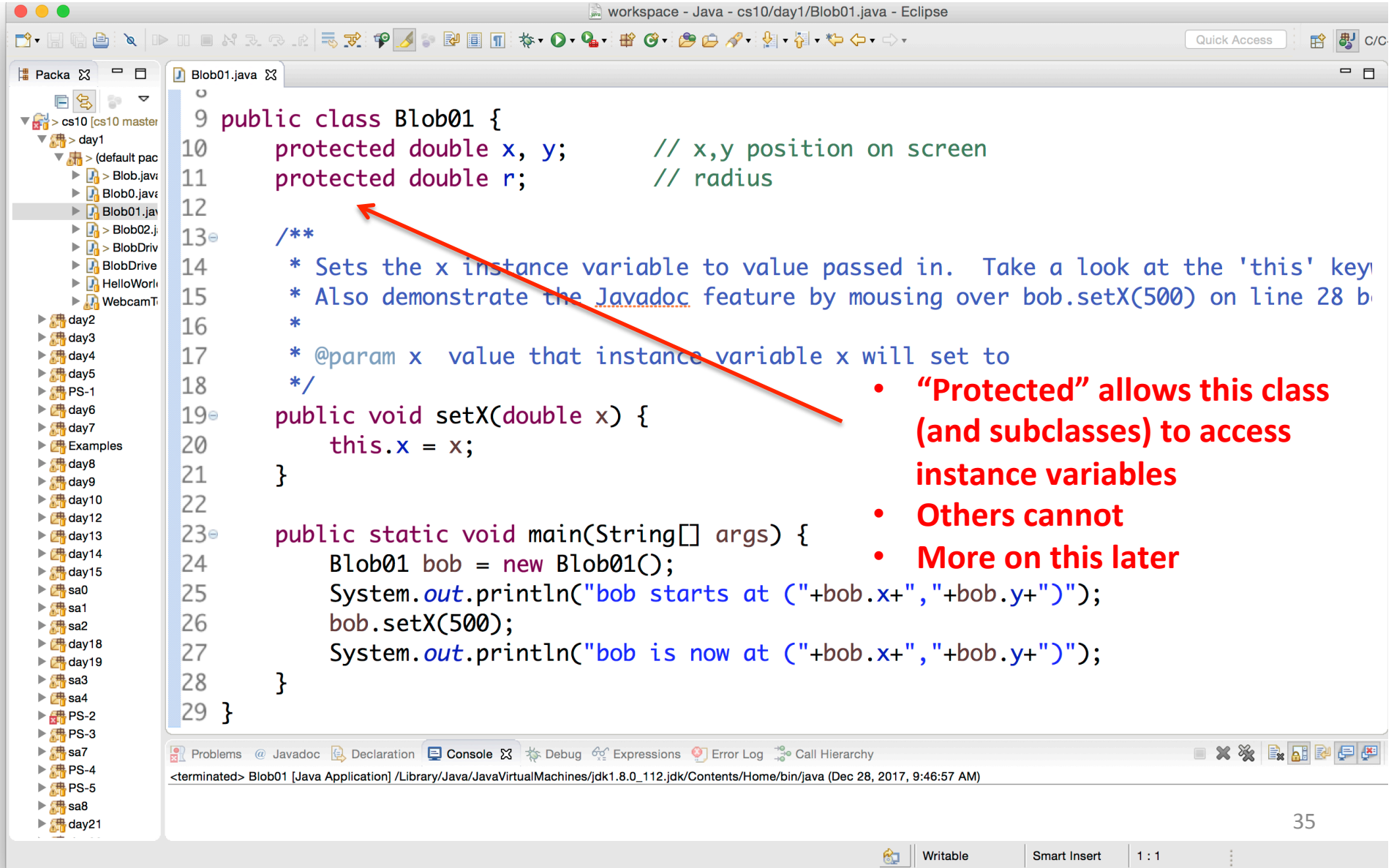


- **Updating instance variables directly is considered bad form in Java (but not in Python)**
- **We will not do this in Java!**
- **Better to let the objects update own instance variables**

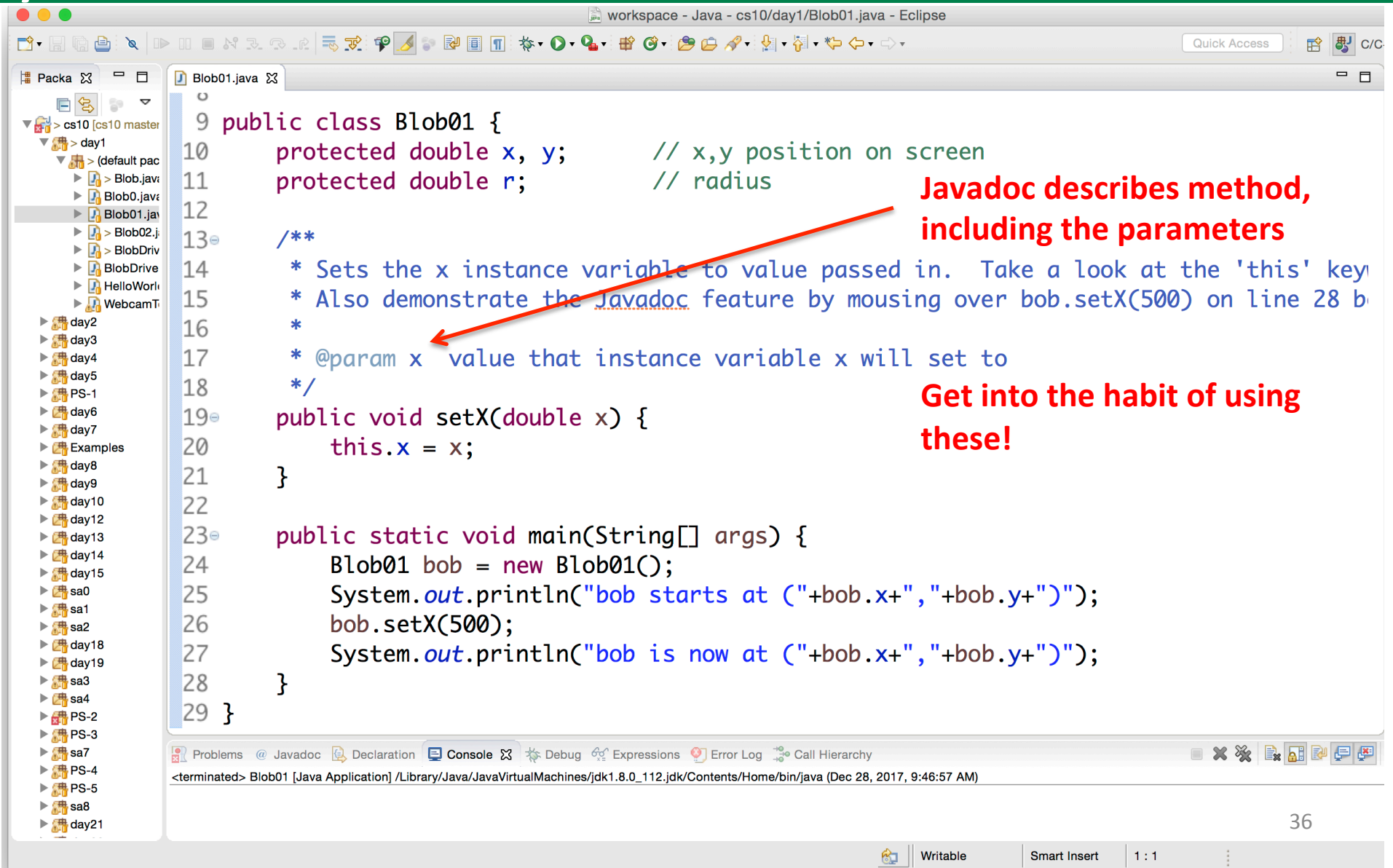
Blob01.java


workspace - Java - cs10/day1/Blob01.java - Eclipse
Quick Access
Packa
cs10 [cs10 master
day1
(default pac
Blob.java
Blob0.java
Blob01.ja
Blob02.j
BlobDriv
BlobDrive
HelloWorl
WebcamT
day2
day3
day4
day5
PS-1
day6
day7
Examples
day8
day9
day10
day12
day13
day14
day15
sa0
sa1
sa2
day18
day19
sa3
sa4
PS-2
PS-3
sa7
PS-4
PS-5
sa8
day21
Blob01.java
9 public class Blob01 {
10 protected double x, y; // x,y position on screen
11 protected double r; // radius
12
13 /**
14 * Sets the x instance variable to value passed in. Take a look at the 'this' key
15 * Also demonstrate the Javadoc feature by mousing over bob.setX(500) on line 28 b
16 *
17 * @param x value that instance variable x will set to
18 */
19 public void setX(double x) {
20 this.x = x;
21 }
22
23 public static void main(String[] args) {
24 Blob01 bob = new Blob01();
25 System.out.println("bob starts at ("+bob.x+","+bob.y+")");
26 bob.setX(500);
27 System.out.println("bob is now at ("+bob.x+","+bob.y+")");
28 }
29 }
Problems
Javadoc
Declaration
Console
Debug
Expressions
Error Log
Call Hierarchy
<terminated> Blob01 [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Dec 28, 2017, 9:46:57 AM)
Writable
Smart Insert
1 : 1

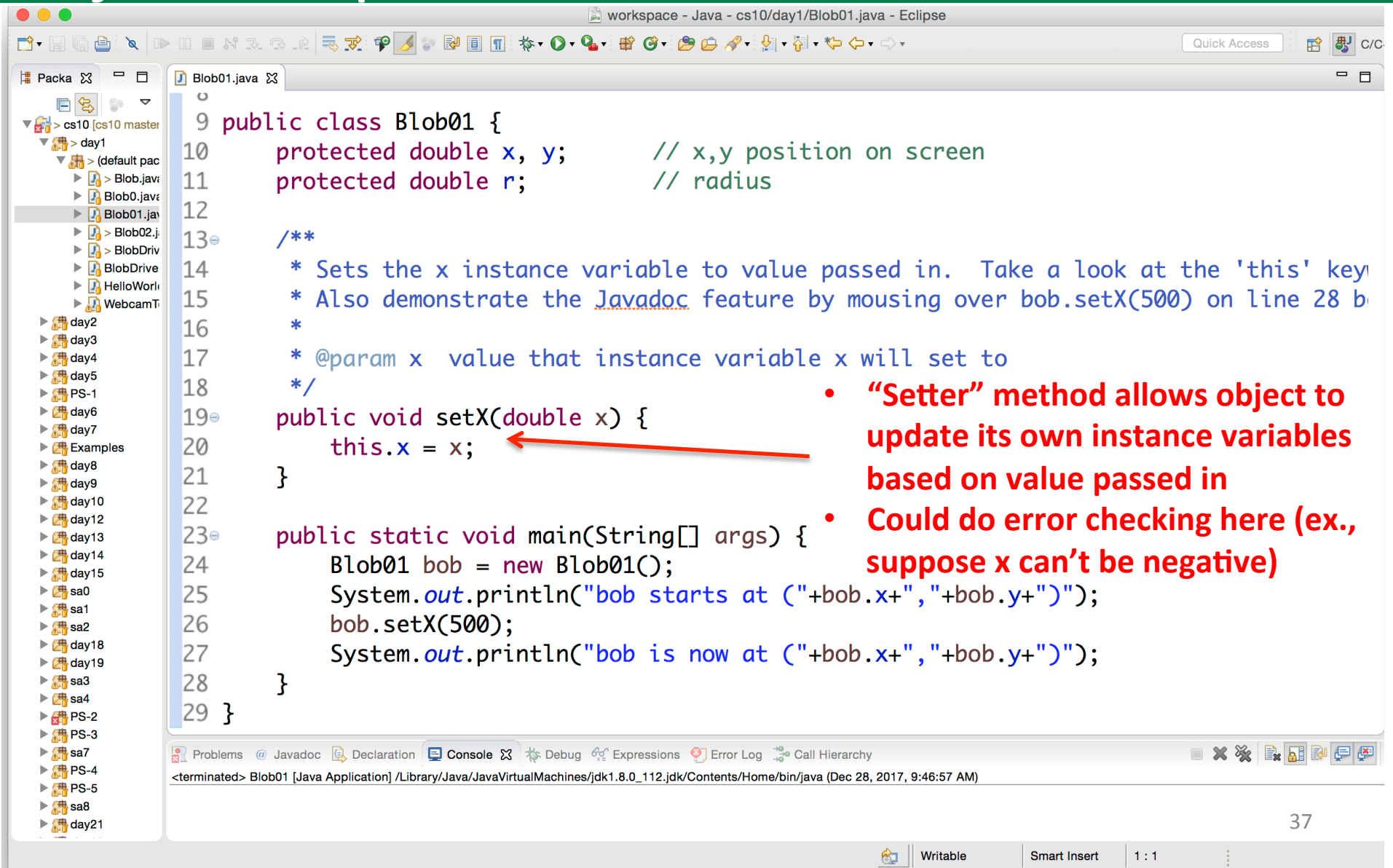
Blob01: Declaring instance variables as "protected" prevents outside modification



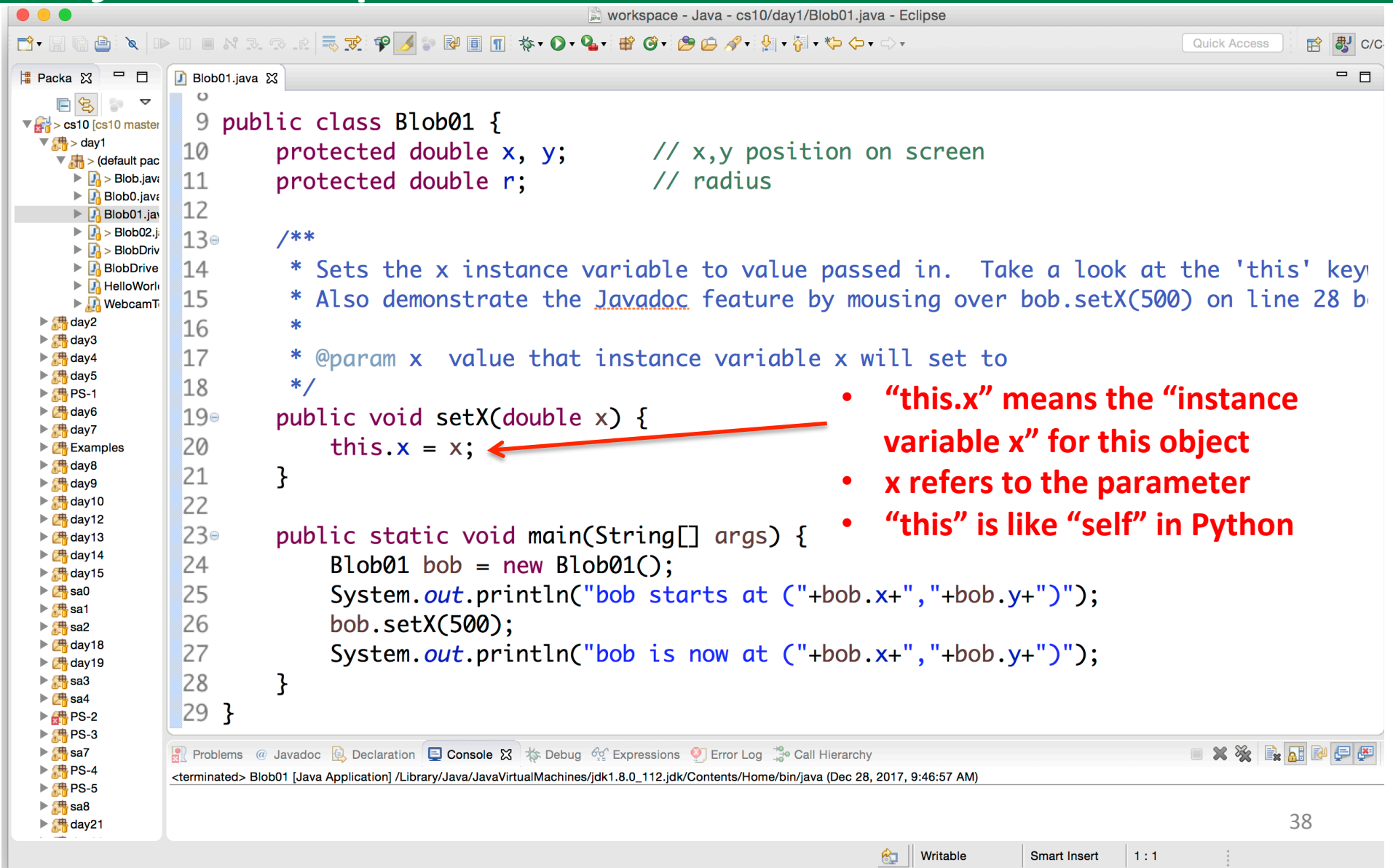
Blob01: Javadoc allows you to document your methods



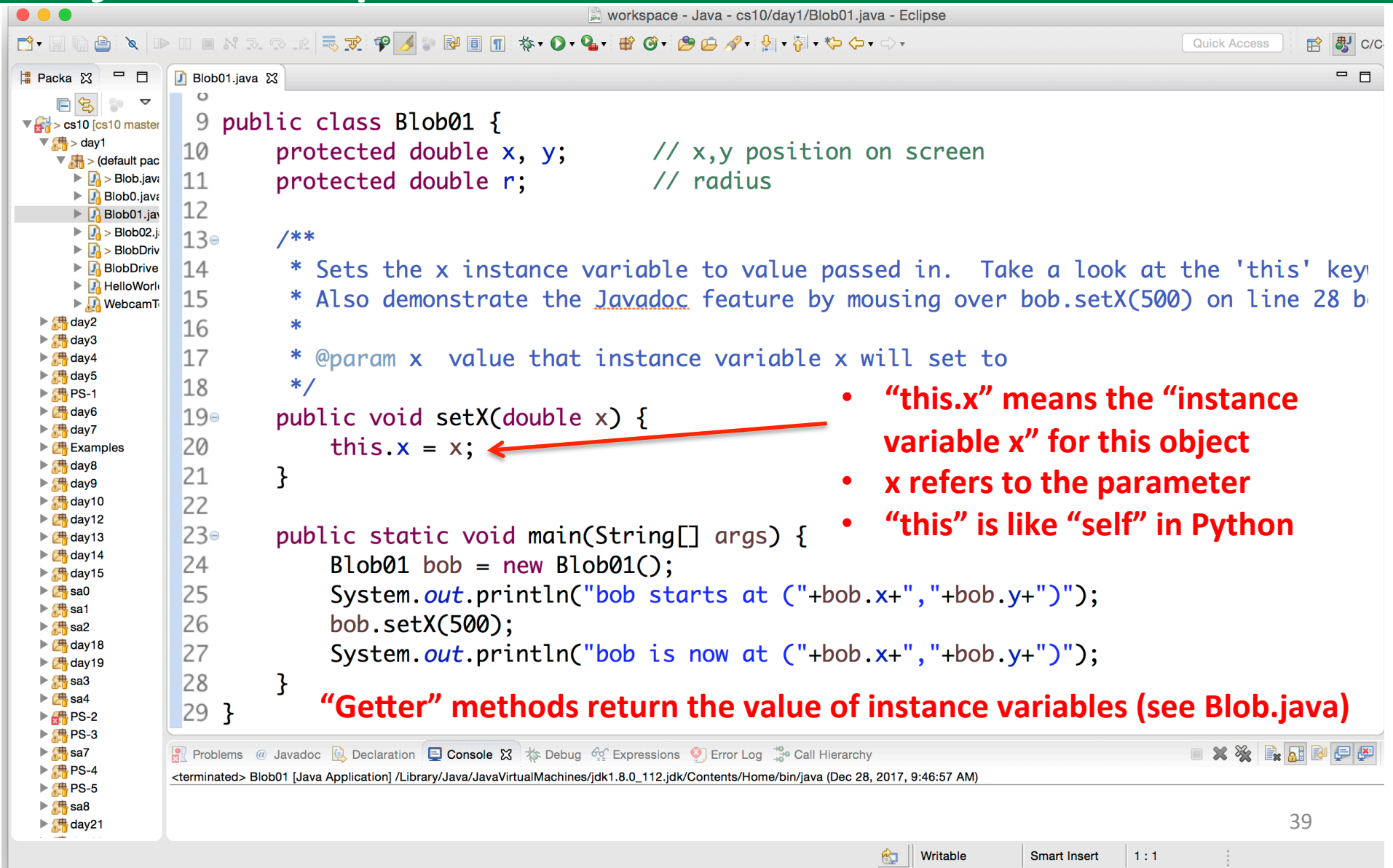
Blob01: Add a "setter" method to allow the object to update its own instance variables



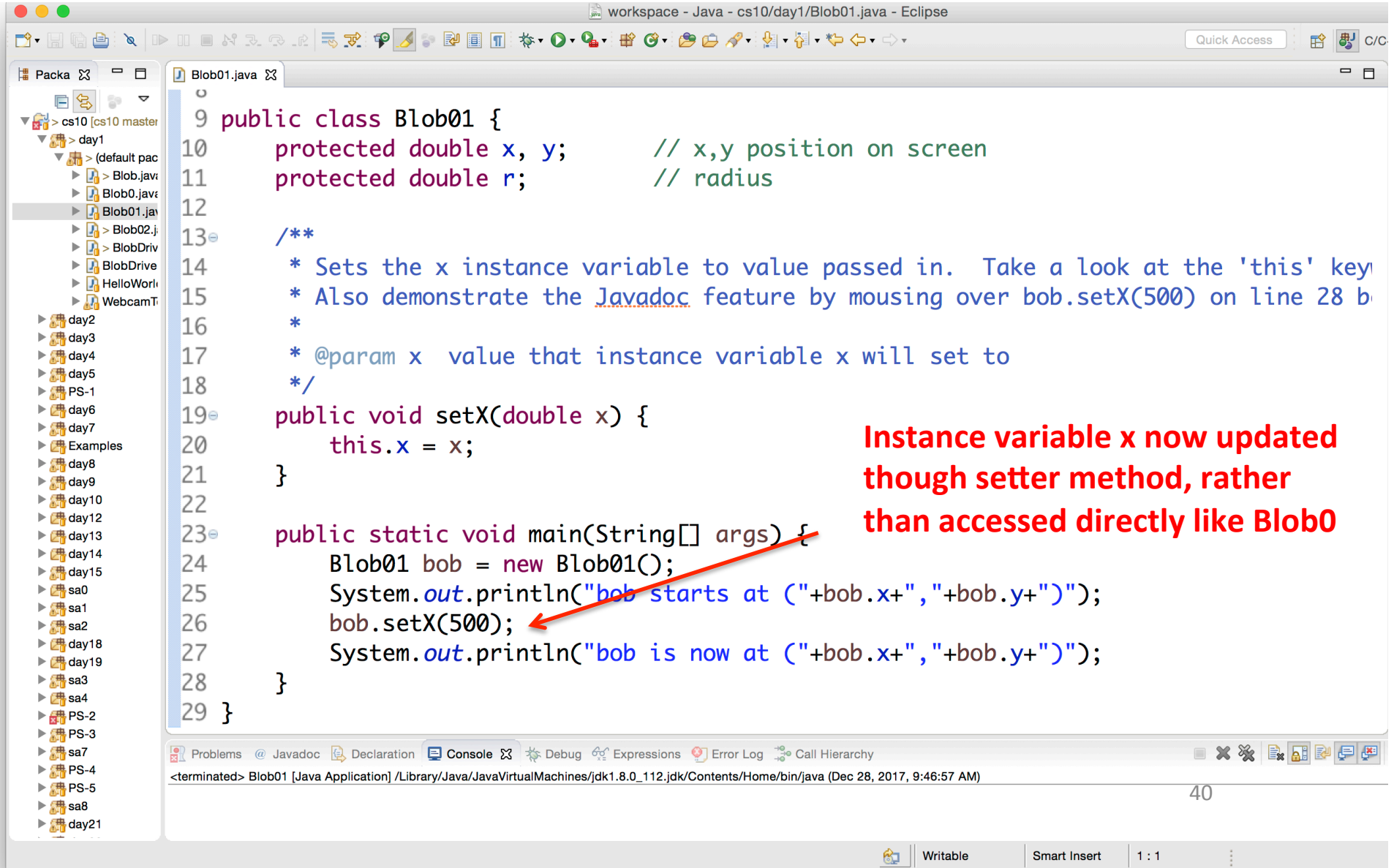
Blob01: Add a "setter" method to allow the object to update its own instance variables



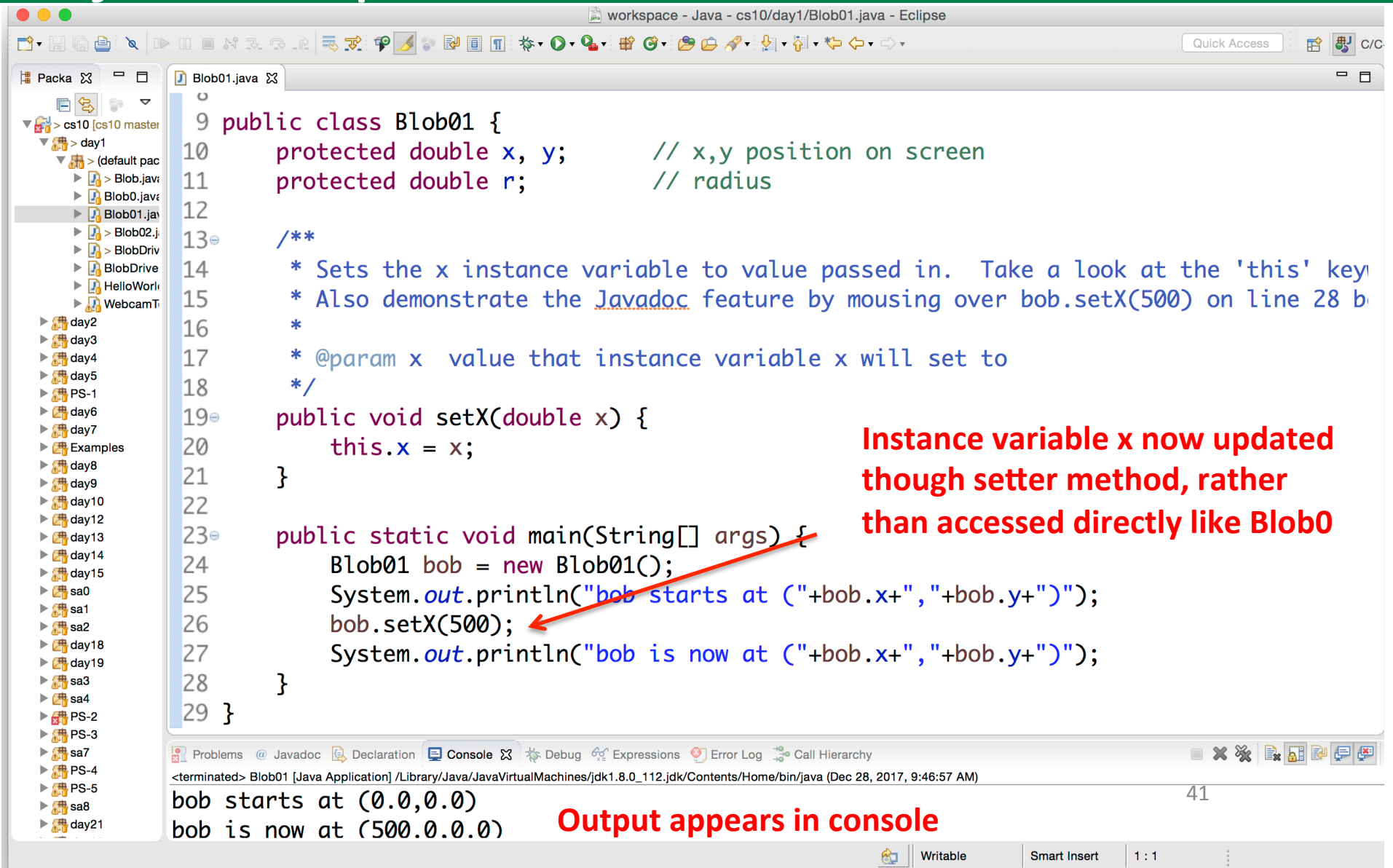
Blob01: Add a "setter" method to allow the object to update its own instance variables



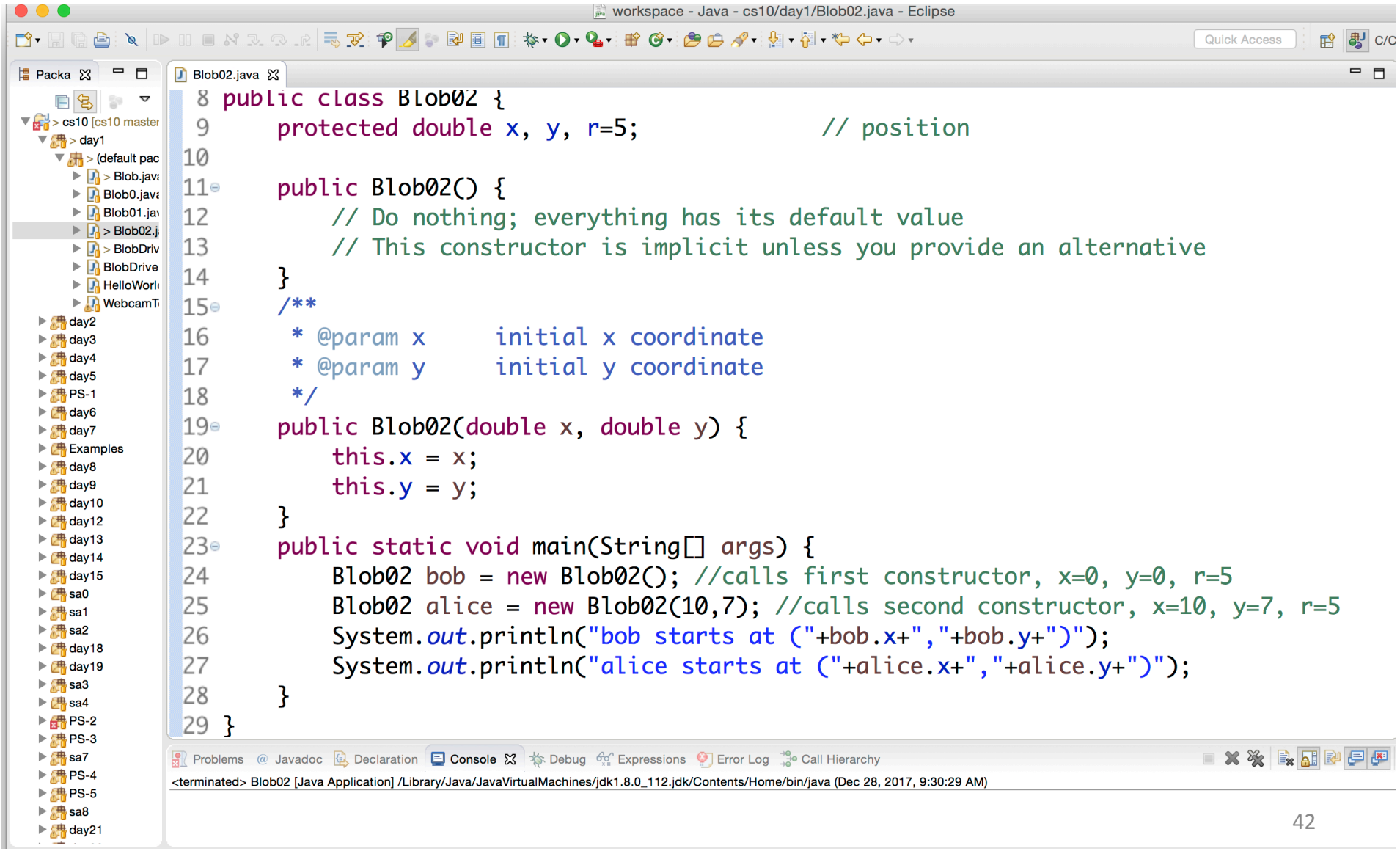
Blob01: Add a "setter" method to allow the object to update its own instance variables



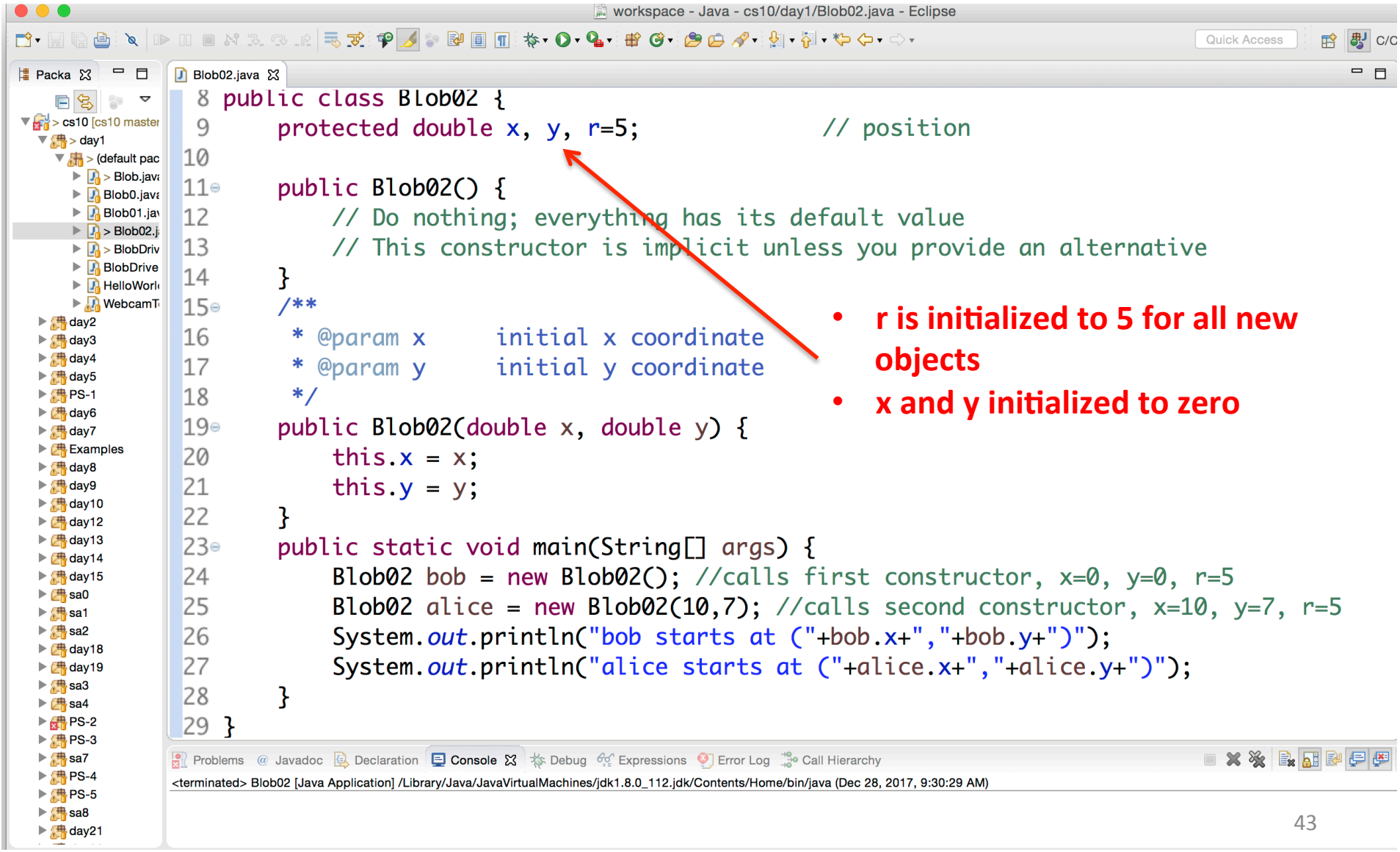
Blob01: Add a "setter" method to allow the object to update its own instance variables



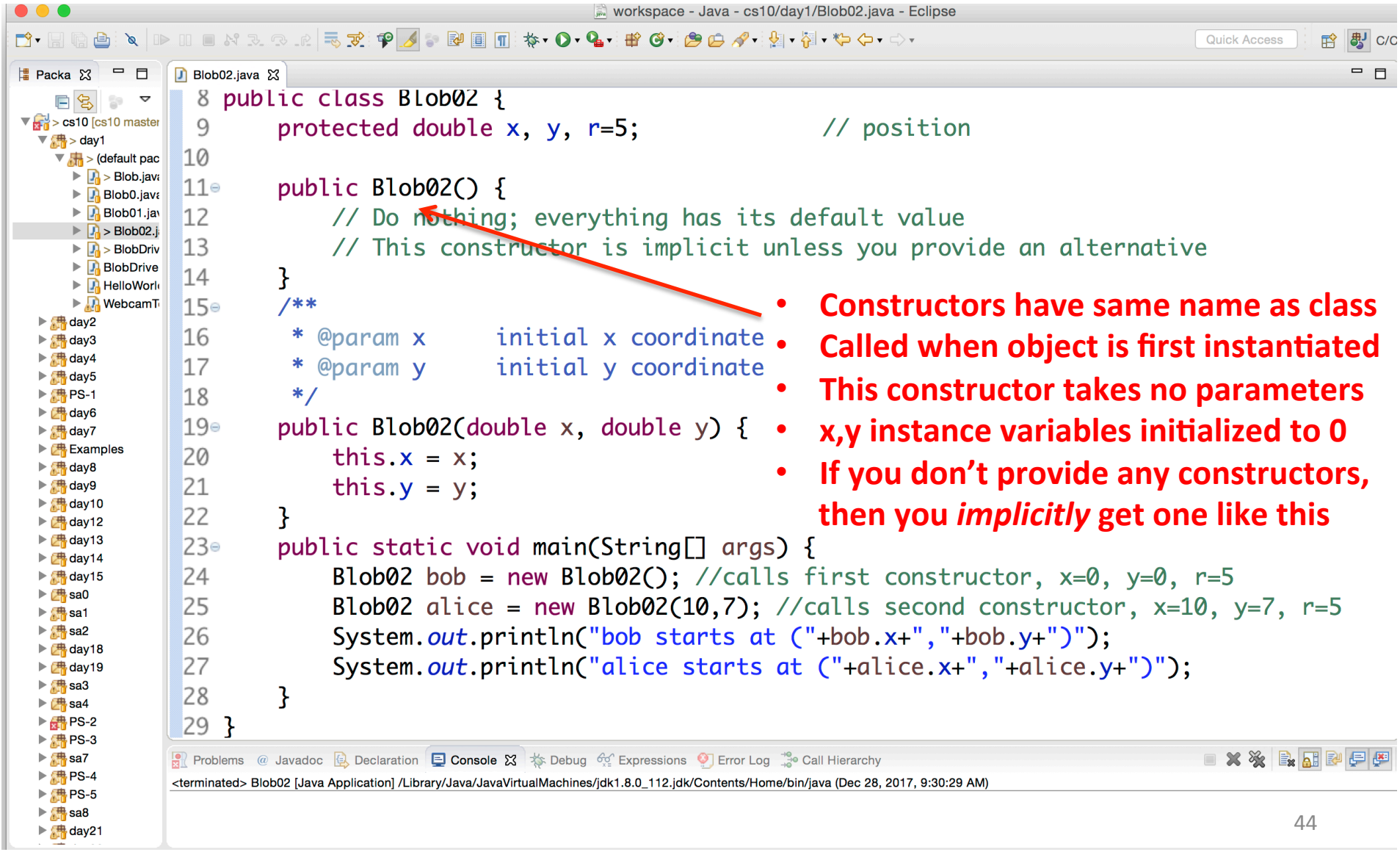
Blob02.java


workspace - Java - cs10/day1/Blob02.java - Eclipse
Quick Access
Packa
Blob02.java
cs10 [cs10 master
day1
(default pac
Blob.java
Blob0.java
Blob01.jav
Blob02.j
BlobDriv
BlobDrive
HelloWorl
WebcamT
day2
day3
day4
day5
PS-1
day6
day7
Examples
day8
day9
day10
day12
day13
day14
day15
sa0
sa1
sa2
day18
day19
sa3
sa4
PS-2
PS-3
sa7
PS-4
PS-5
sa8
day21
8 public class Blob02 {
9 protected double x, y, r=5; // position
10
11 public Blob02() {
12 // Do nothing; everything has its default value
13 // This constructor is implicit unless you provide an alternative
14 }
15 /**
16 * @param x initial x coordinate
17 * @param y initial y coordinate
18 */
19 public Blob02(double x, double y) {
20 this.x = x;
21 this.y = y;
22 }
23 public static void main(String[] args) {
24 Blob02 bob = new Blob02(); //calls first constructor, x=0, y=0, r=5
25 Blob02 alice = new Blob02(10,7); //calls second constructor, x=10, y=7, r=5
26 System.out.println("bob starts at ("+bob.x+","+bob.y+")");
27 System.out.println("alice starts at ("+alice.x+","+alice.y+")");
28 }
29 }
Problems
Javadoc
Declaration
Console
Debug
Expressions
Error Log
Call Hierarchy
<terminated> Blob02 [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Dec 28, 2017, 9:30:29 AM)

Blob02: Instance variables can be initialized to values other than zero

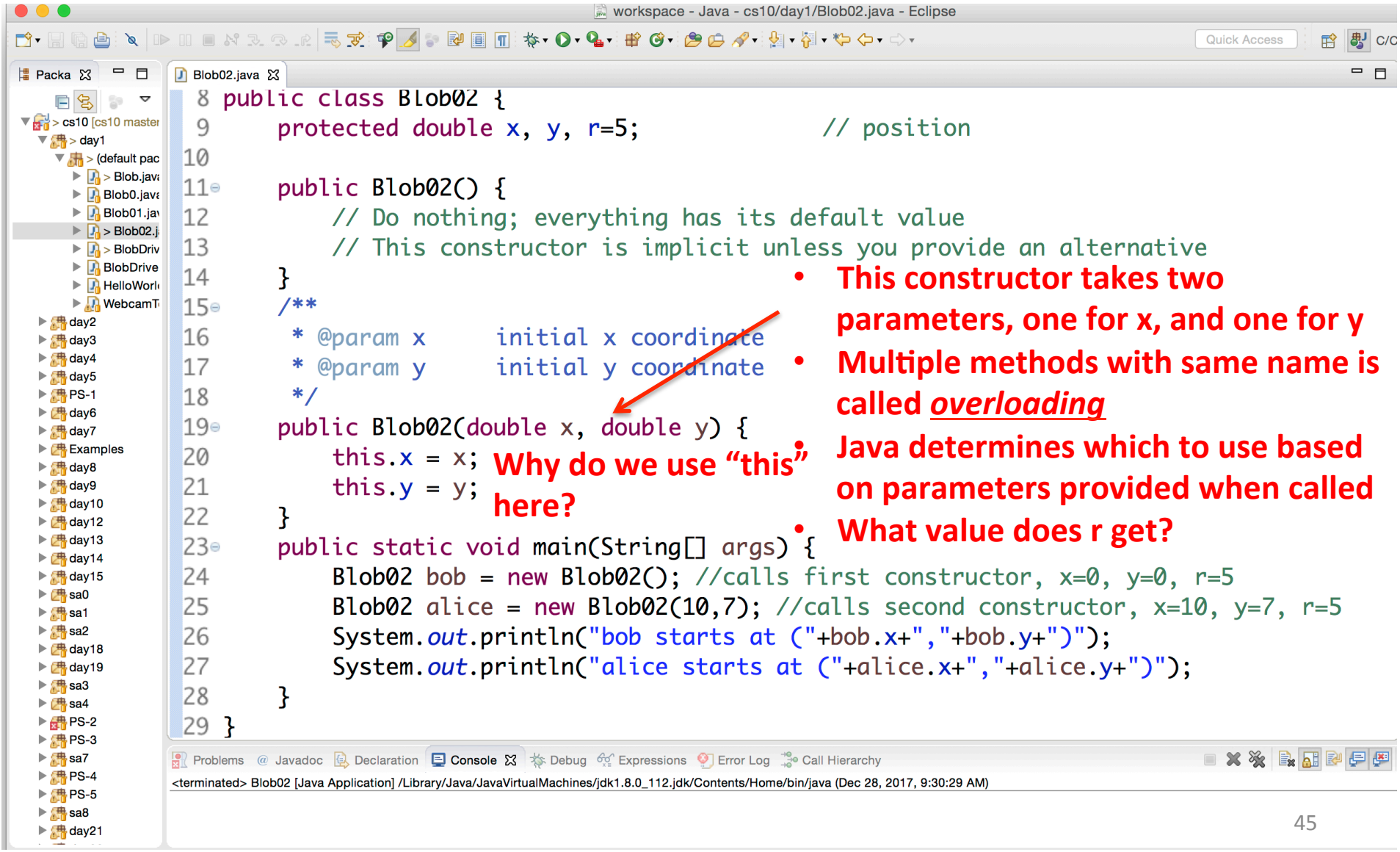


Blob02: Constructors called when an object is first instantiated (run before other code)

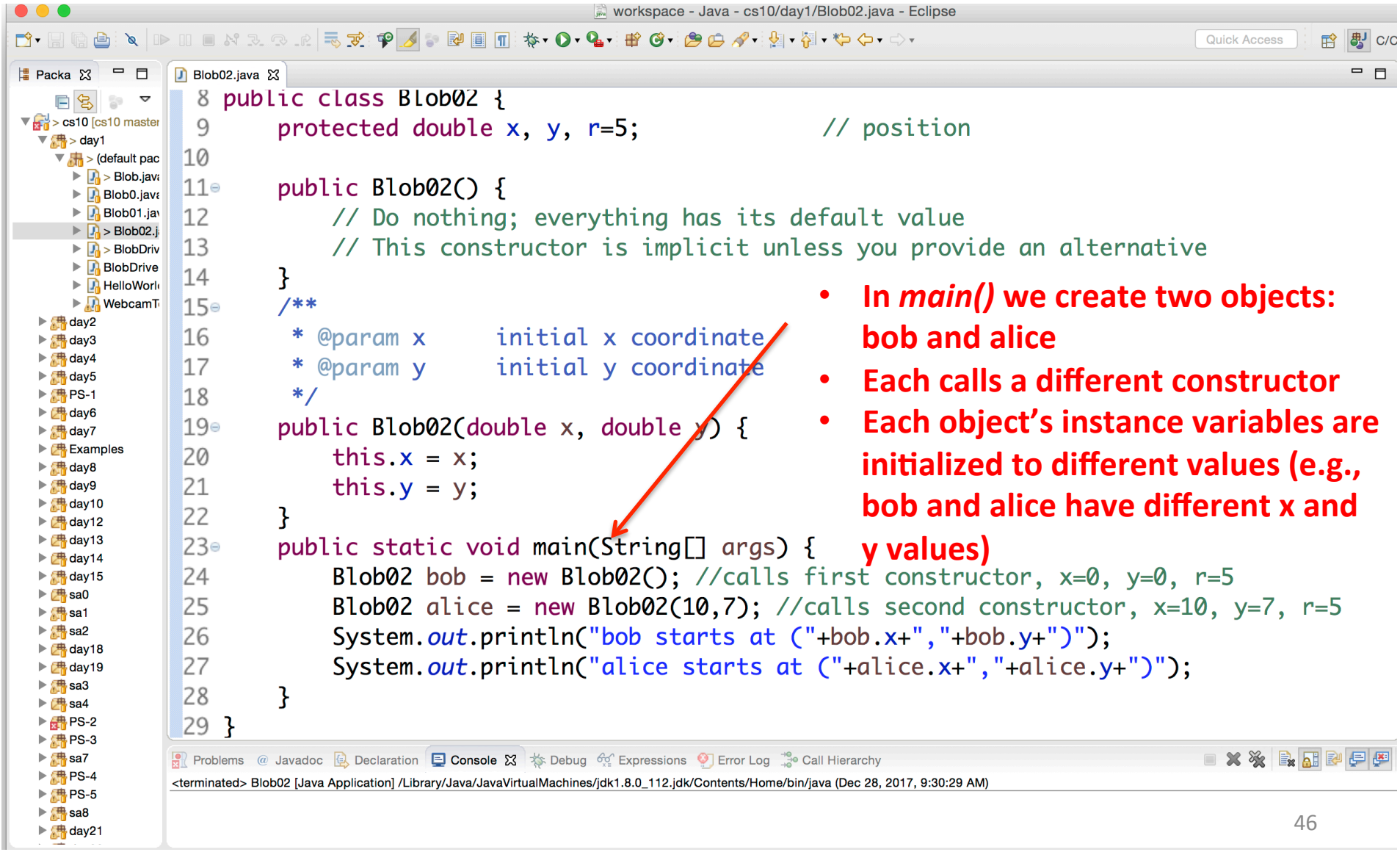


- Constructors have same name as class
- Called when object is first instantiated
- This constructor takes no parameters
- x,y instance variables initialized to 0
- If you don't provide any constructors, then you *implicitly* get one like this

Blob02: Constructors called when an object is first instantiated (run before other code)

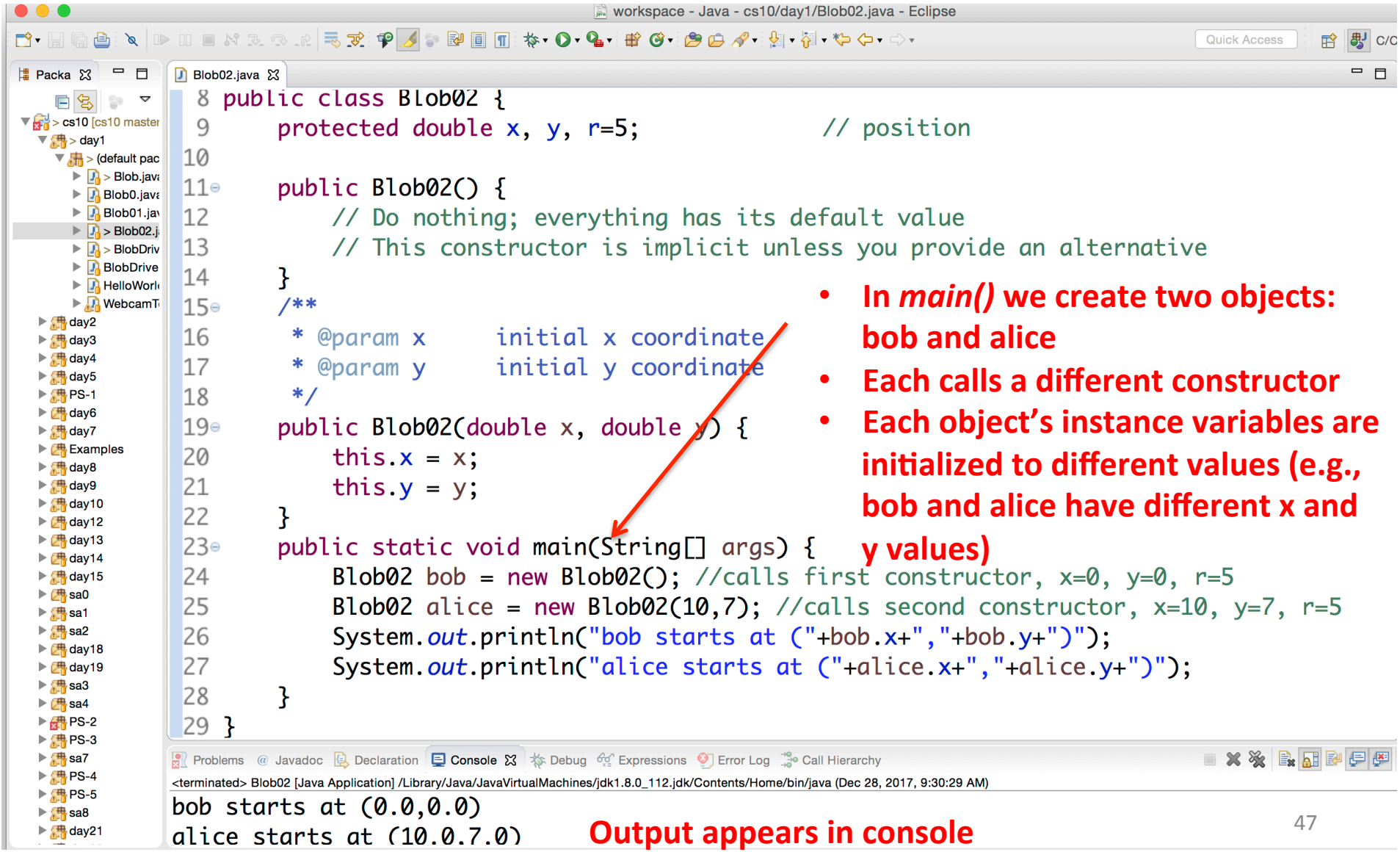


Blob02: Constructors called when an object is first instantiated (run before other code)



- **In *main()* we create two objects: bob and alice**
- **Each calls a different constructor**
- **Each object's instance variables are initialized to different values (e.g., bob and alice have different x and y values)**

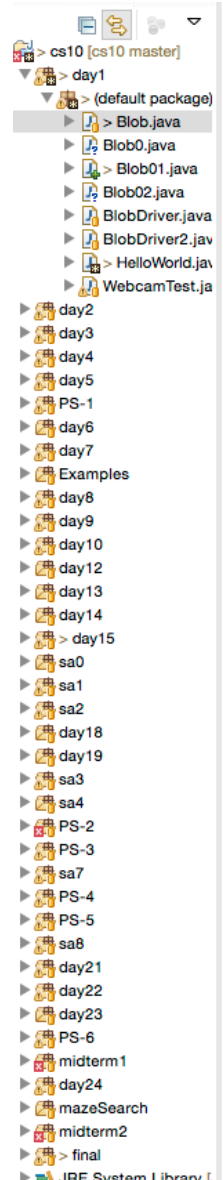
Blob02: Constructors called when an object is first instantiated (run before other code)



- **In *main()* we create two objects: bob and alice**
- **Each calls a different constructor**
- **Each object's instance variables are initialized to different values (e.g., bob and alice have different x and y values)**

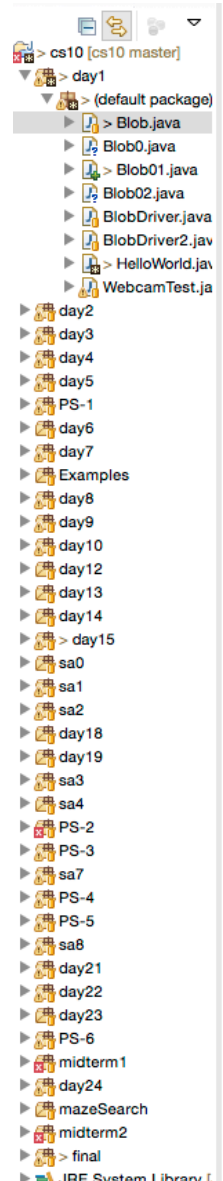
Output appears in console

Blob.java




```
10 public class Blob {
11     protected double x, y;               // position
12     protected double dx=0, dy=0;     // velocity, defaults to none
13     protected double r=5;                // radius
14     protected double dr=0;               // growth step (size and sign), defaults to none
15
16     public Blob() {
17         // Do nothing; everything has its default value
18         // This constructor is implicit unless you provide an alternative
19     }
20
21     /**
22      * @param x      initial x coordinate
23      * @param y      initial y coordinate
24      */
25     public Blob(double x, double y) {
26         this.x = x;
27         this.y = y;
28     }
29
30     /**
31      * @param x      initial x coordinate
32      * @param y      initial y coordinate
33      * @param r      initial radius
34      */
35     public Blob(double x, double y, double r) {
36         this.x = x;
37         this.y = y;
38         this.r = r;
39     }
```

Blob: Small changes to previous versions give a more full featured Blob

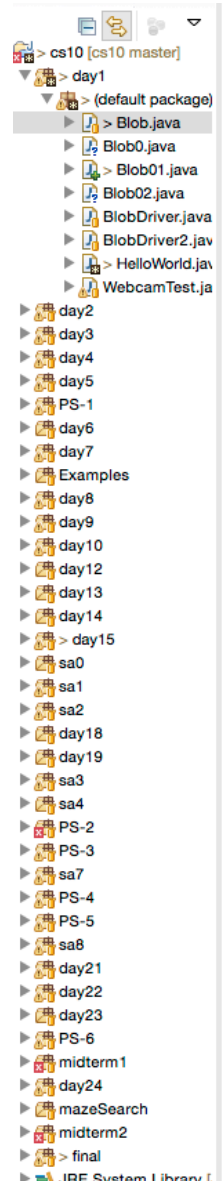



```
10 public class Blob {
11     protected double x, y;                  // position
12     protected double dx=0, dy=0;     // velocity, defaults to none
13     protected double r=5;                   // radius
14     protected double dr=0;                  // growth step (size and sign), defaults to none
15
16     public Blob() {
17         // Do nothing; everything has its default value
18         // This constructor is implicit unless you provide an alternative
19     }
20
21     /**
22      * @param x      initial x coordinate
23      * @param y      initial y coordinate
24      */
25     public Blob(double x, double y) {
26         this.x = x;
27         this.y = y;
28     }
29
30     /**
31      * @param x      initial x coordinate
32      * @param y      initial y coordinate
33      * @param r      initial radius
34      */
35     public Blob(double x, double y, double r) {
36         this.x = x;
37         this.y = y;
38         this.r = r;
39     }
```

New instance variables

- **dx,dy,dr will be the amount each instance variable changes at each time step**

Blob: Small changes to previous versions give a more full featured Blob

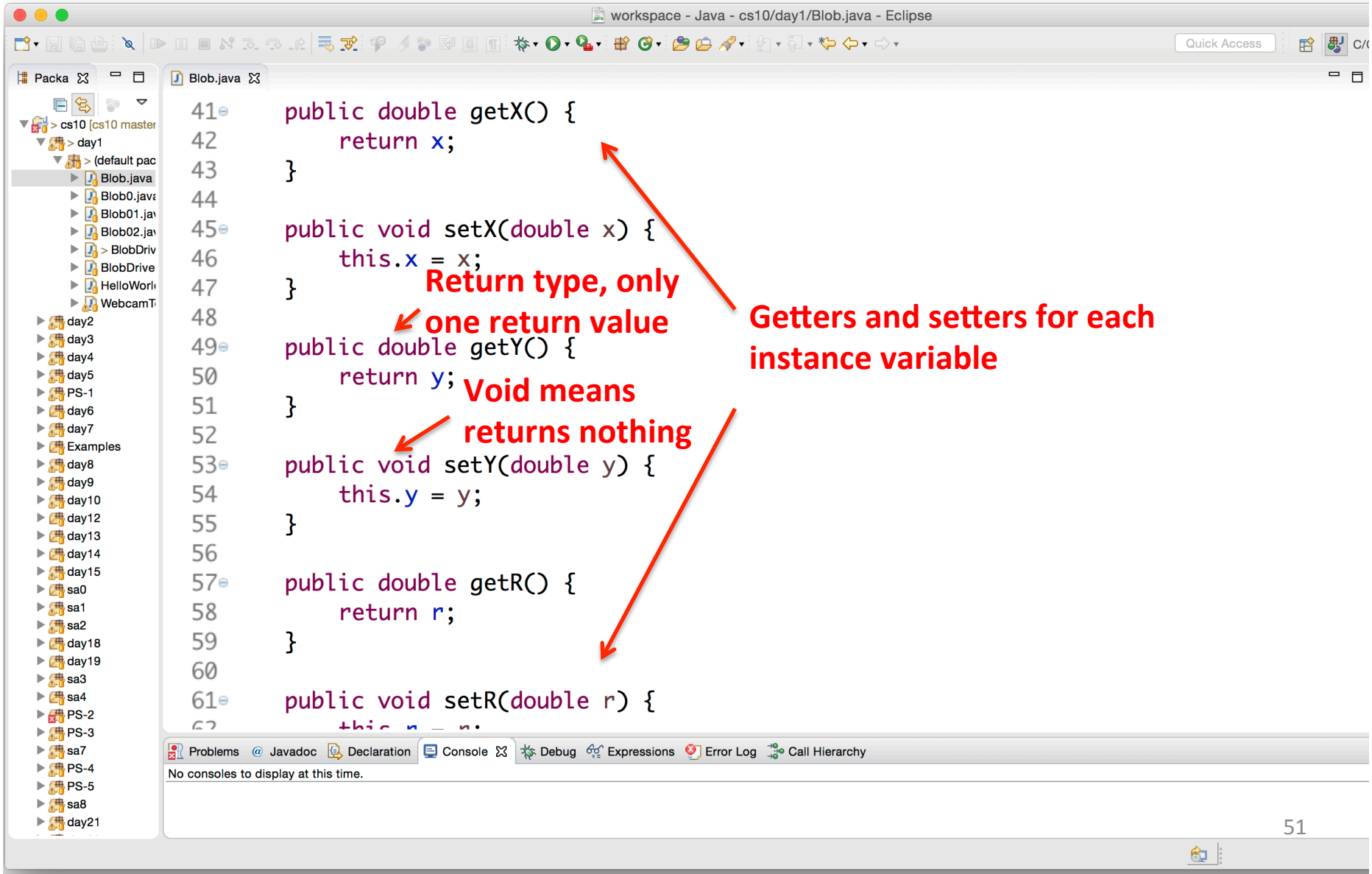



```
10 public class Blob {
11     protected double x, y;                  // position
12     protected double dx=0, dy=0;     // velocity, defaults to none
13     protected double r=5;                   // radius
14     protected double dr=0;                  // growth step (size and sign), defaults to none
15
16     public Blob() {
17         // Do nothing; everything has its default value
18         // This constructor is implicit unless you provide an alternative
19     }
20
21     /**
22      * @param x      initial x coordinate
23      * @param y      initial y coordinate
24      */
25     public Blob(double x, double y) {
26         this.x = x;
27         this.y = y;
28     }
29
30     /**
31      * @param x      initial x coordinate
32      * @param y      initial y coordinate
33      * @param r      initial radius
34      */
35     public Blob(double x, double y, double r) {
36         this.x = x;
37         this.y = y;
38         this.r = r;
39     }
```

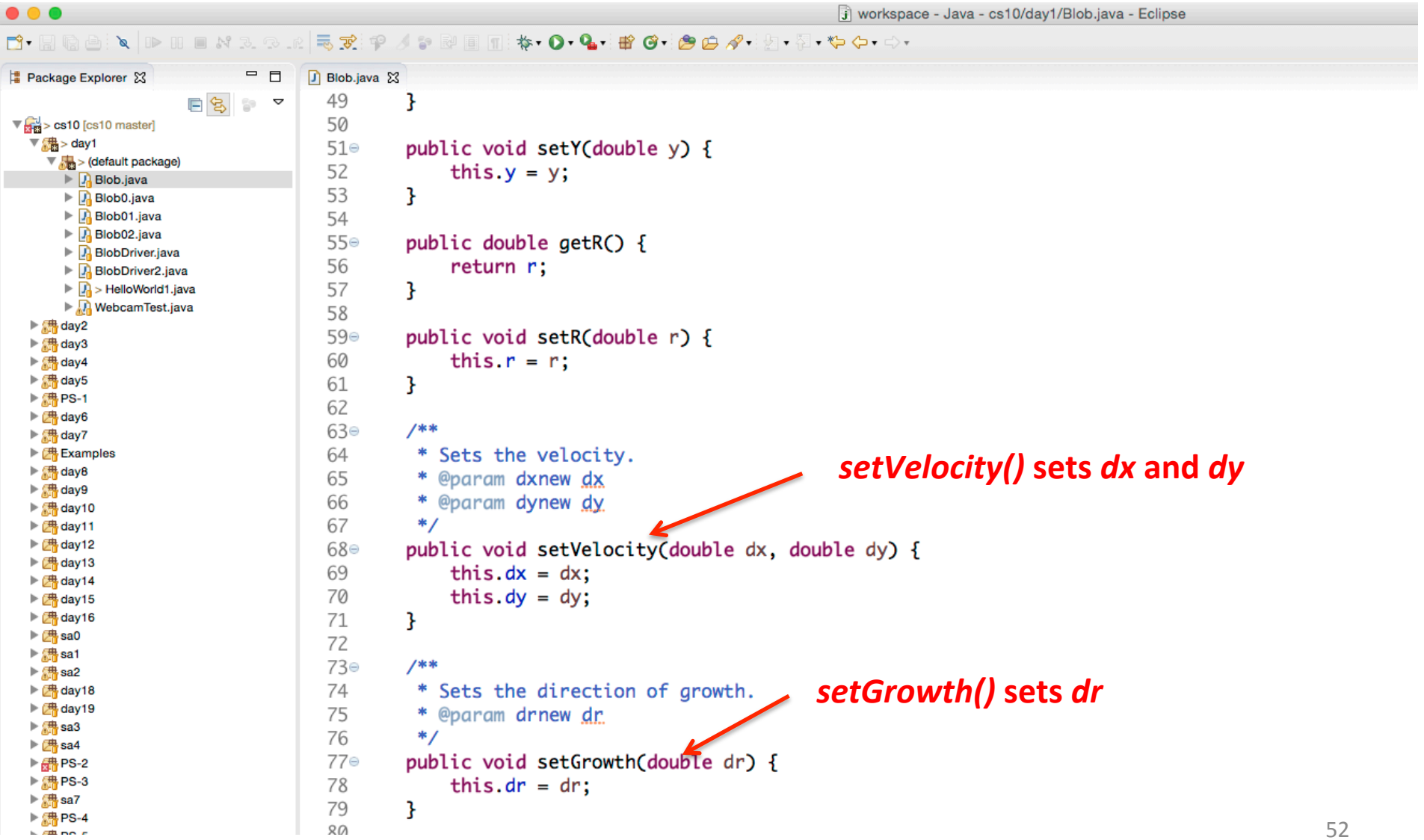
Three (overloaded) constructors

- **Like previous constructors**
- **Now allow for 0, 2, or 3 parameters**

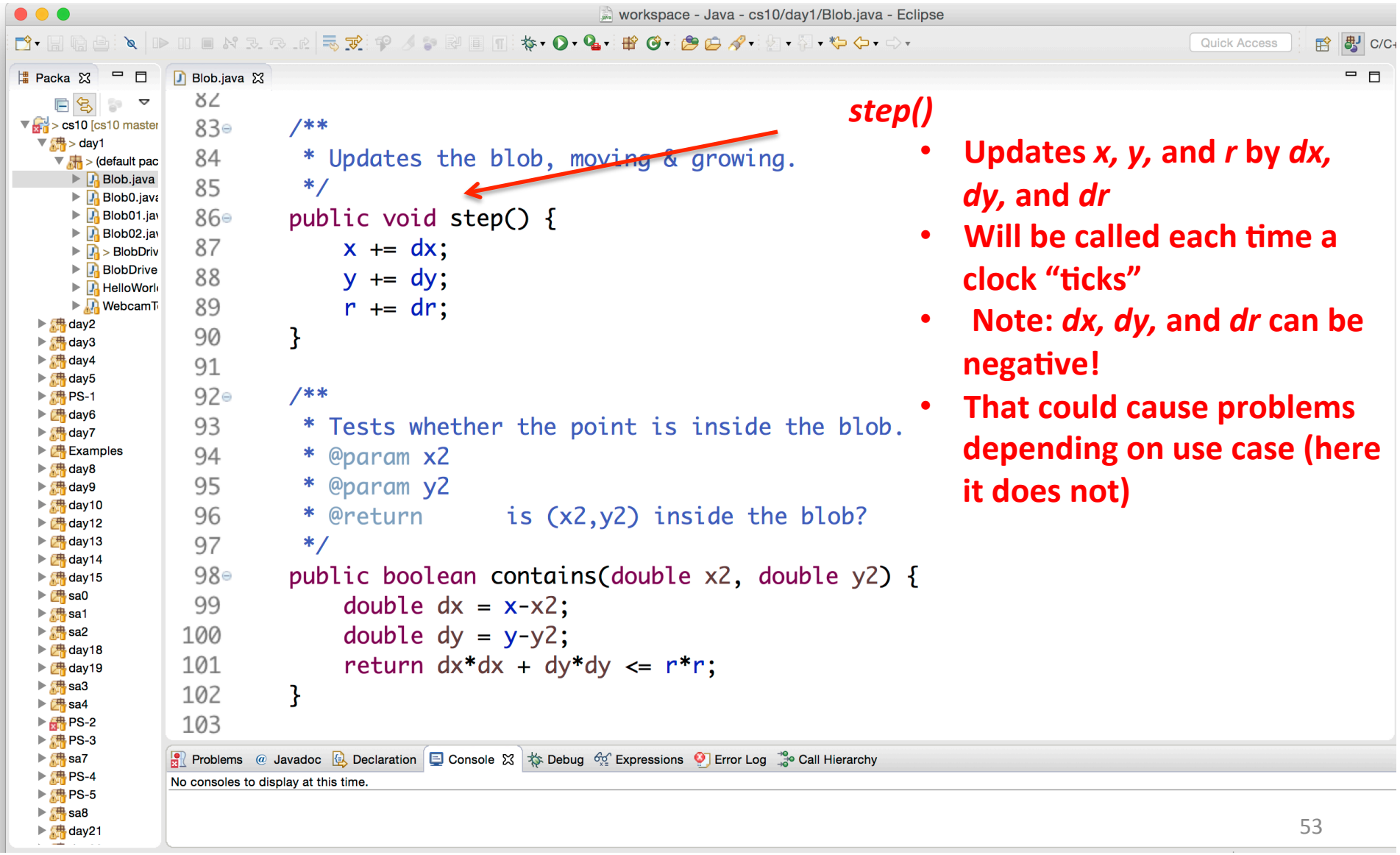
Blob: Small changes to previous versions give a more full featured Blob



Blob: Small changes to previous versions give a more full featured Blob



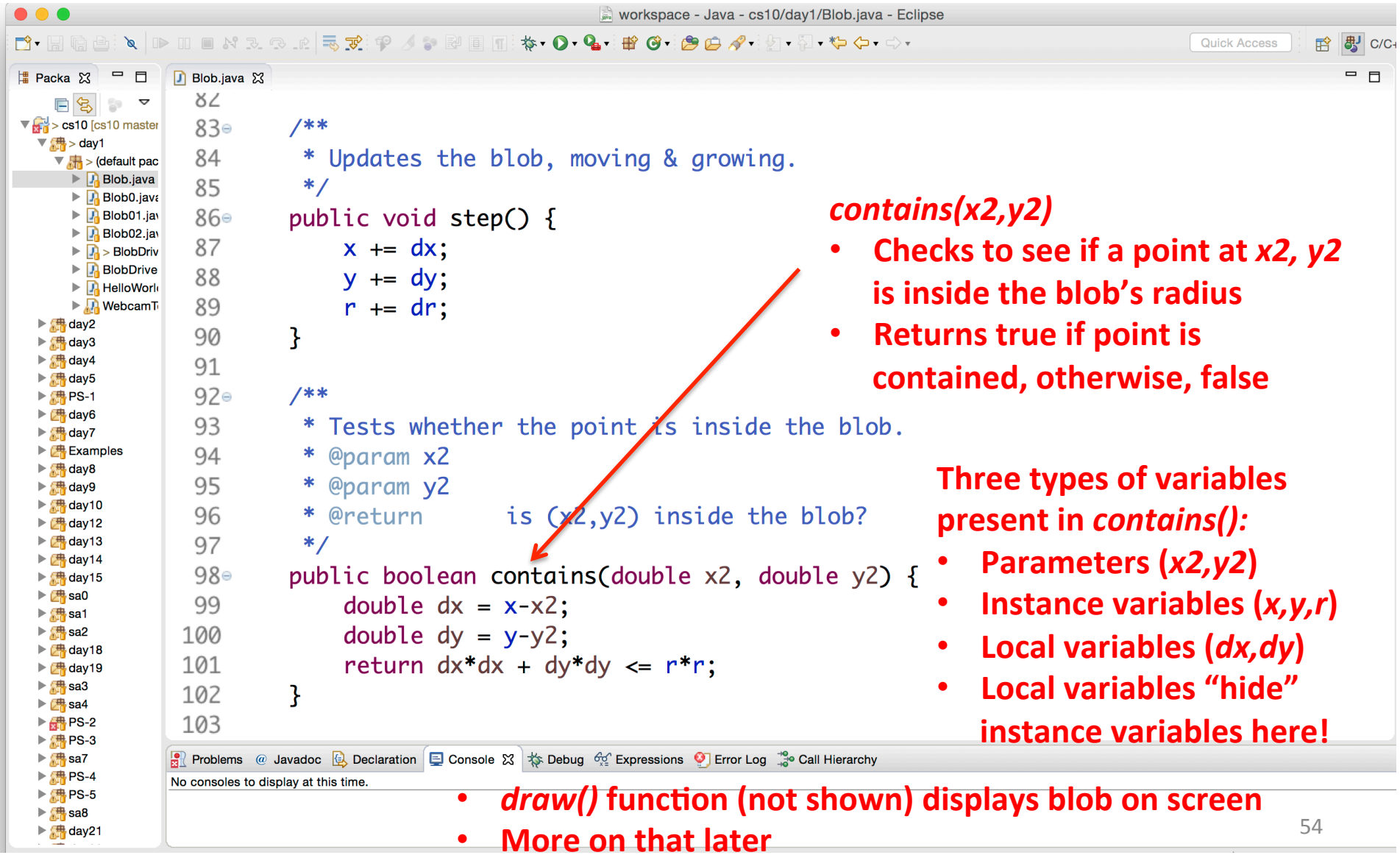
Blob: New methods to control how the blob behaves



step()

- **Updates *x*, *y*, and *r* by *dx*, *dy*, and *dr***
- **Will be called each time a clock "ticks"**
- **Note: *dx*, *dy*, and *dr* can be negative!**
- **That could cause problems depending on use case (here it does not)**

Blob: New methods to control how the blob behaves



contains(x2,y2)

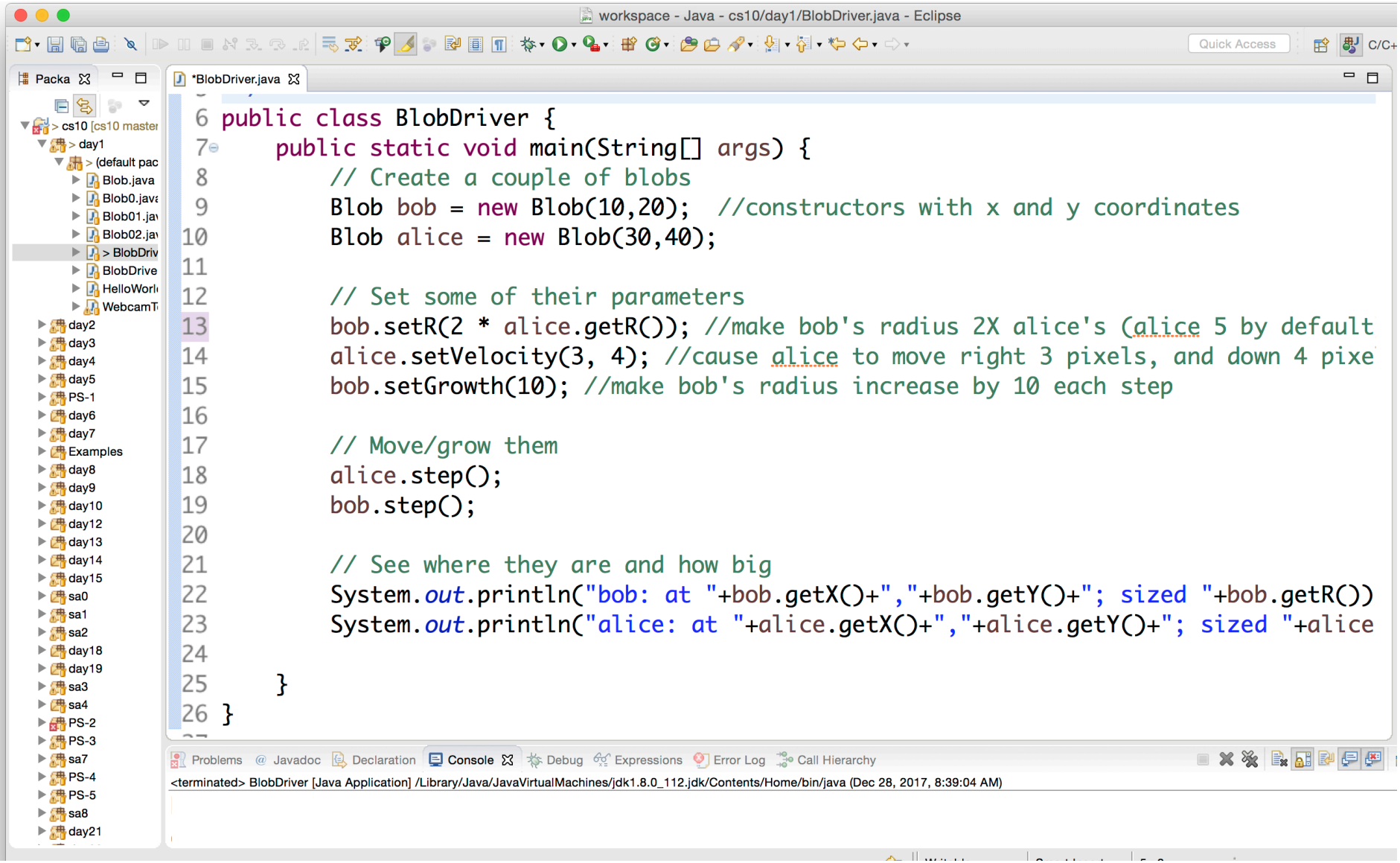
- **Checks to see if a point at *x2, y2* is inside the blob's radius**
- **Returns true if point is contained, otherwise, false**

Three types of variables present in *contains():*

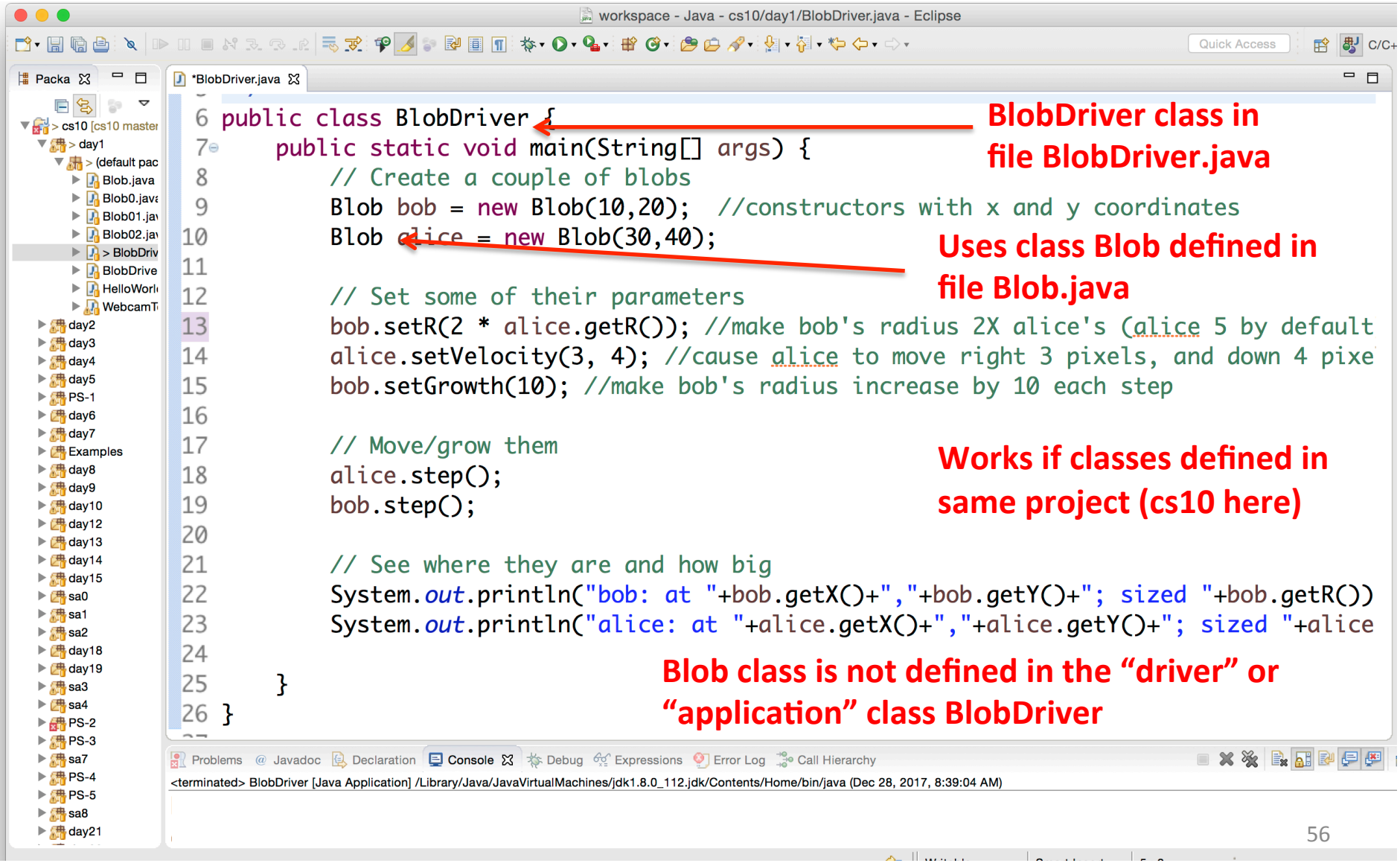
- **Parameters (*x2,y2*)**
- **Instance variables (*x,y,r*)**
- **Local variables (*dx,dy*)**
- **Local variables "hide" instance variables here!**

- ***draw()* function (not shown) displays blob on screen**
- **More on that later**

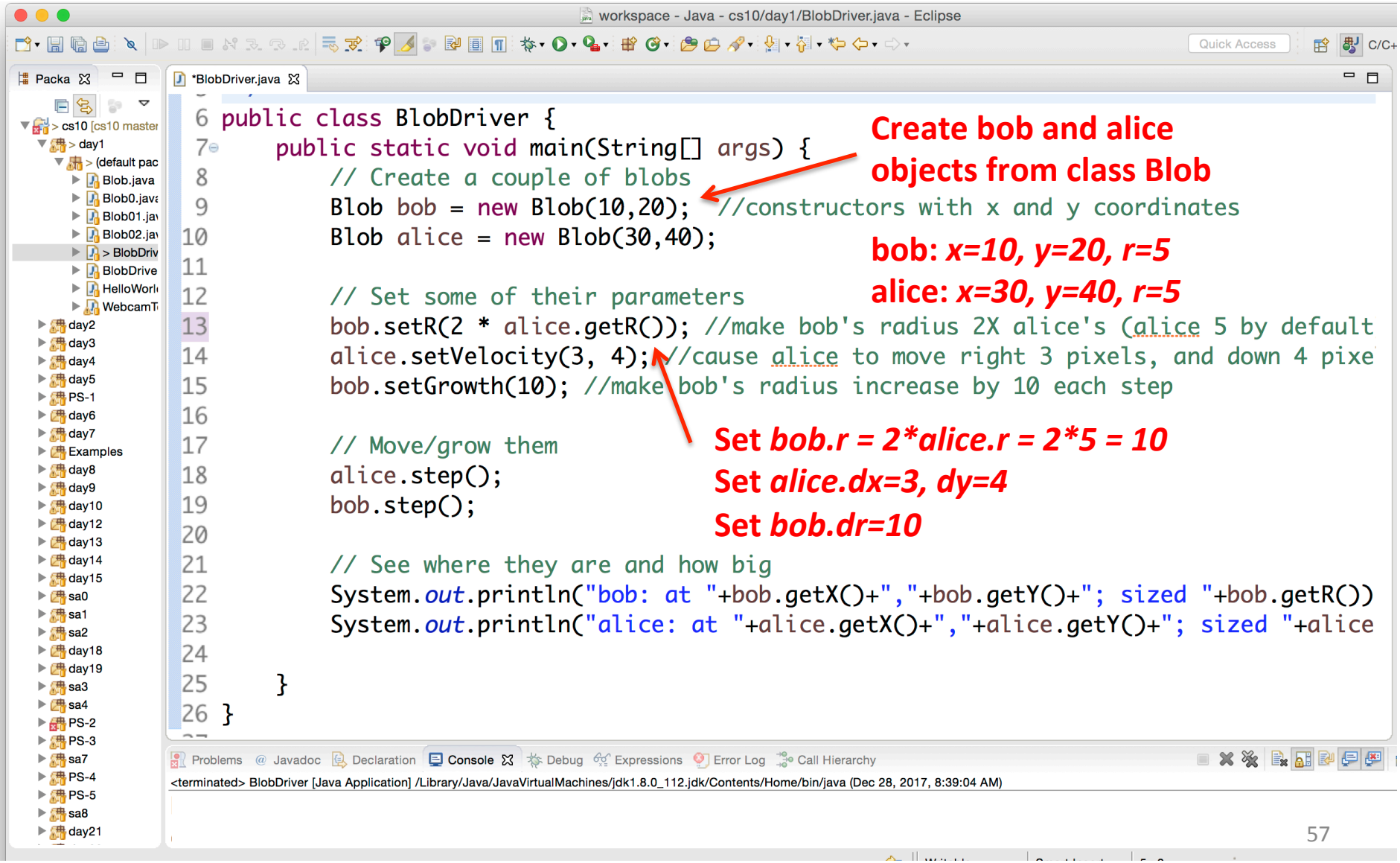
BlobDriver.java: Create a "driver" program separate from class definitions



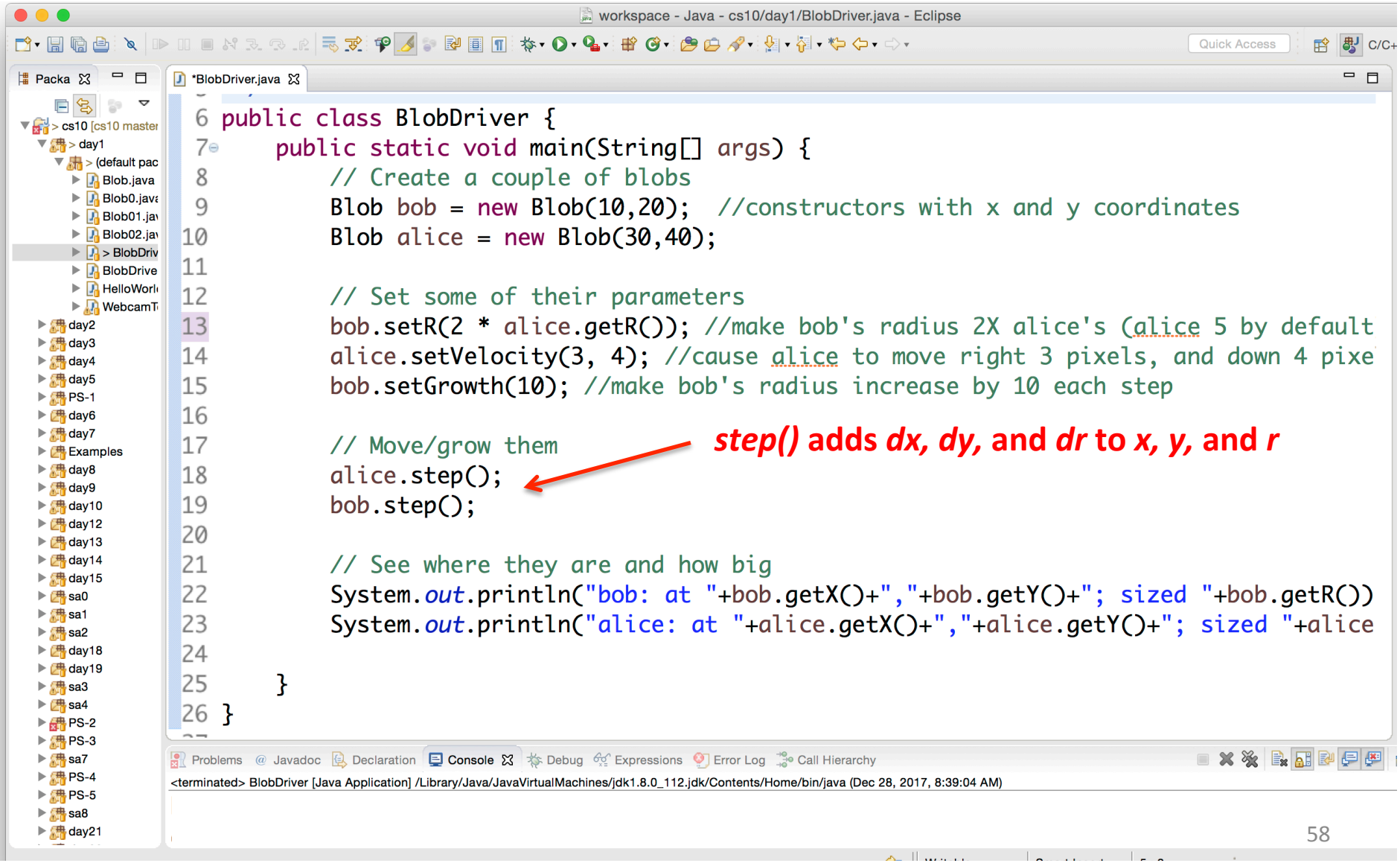
BlobDriver: Uses Blob class to create blob objects and then move/grow them



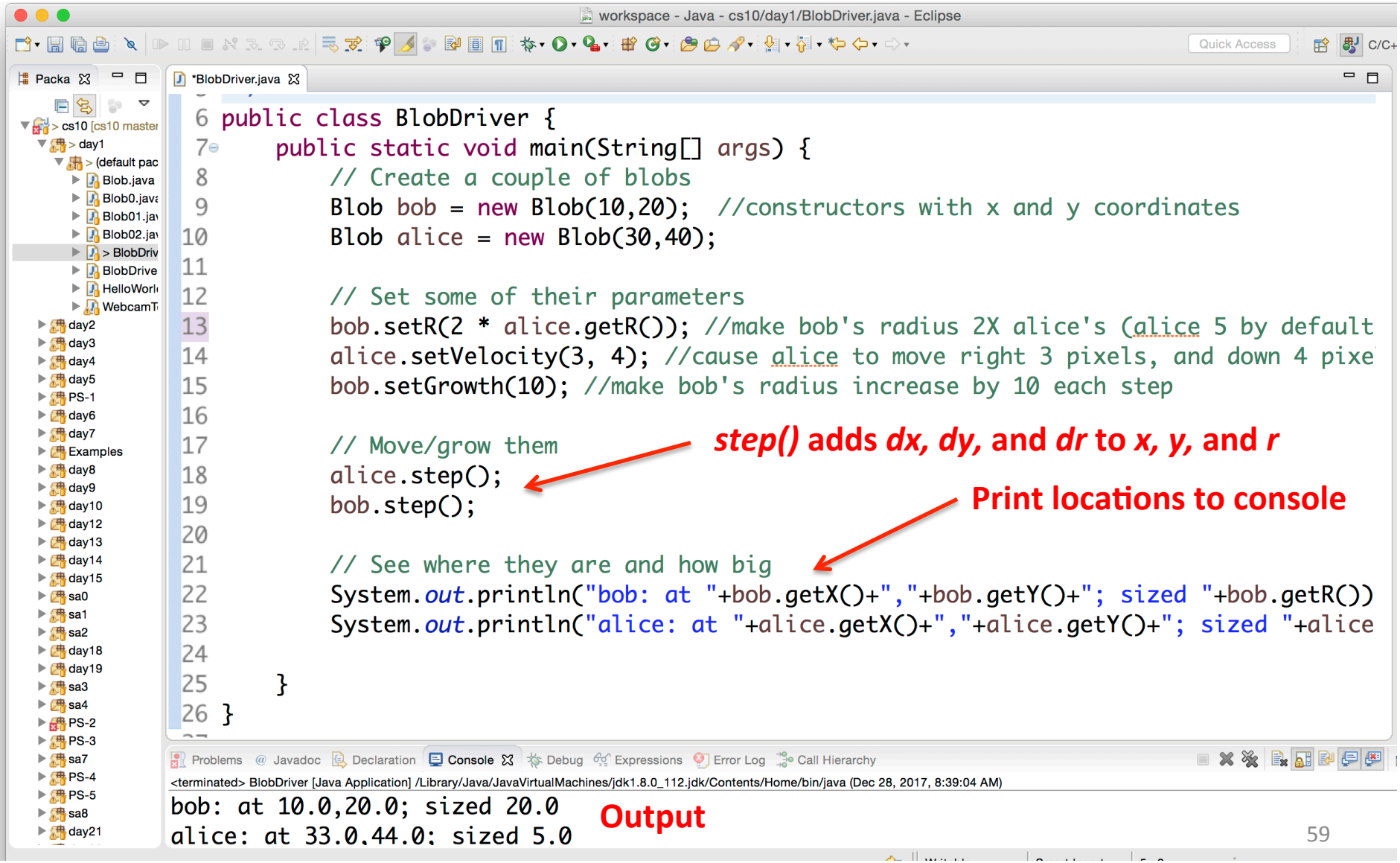
BlobDriver: Uses Blob class to create blob objects and then move/grow them



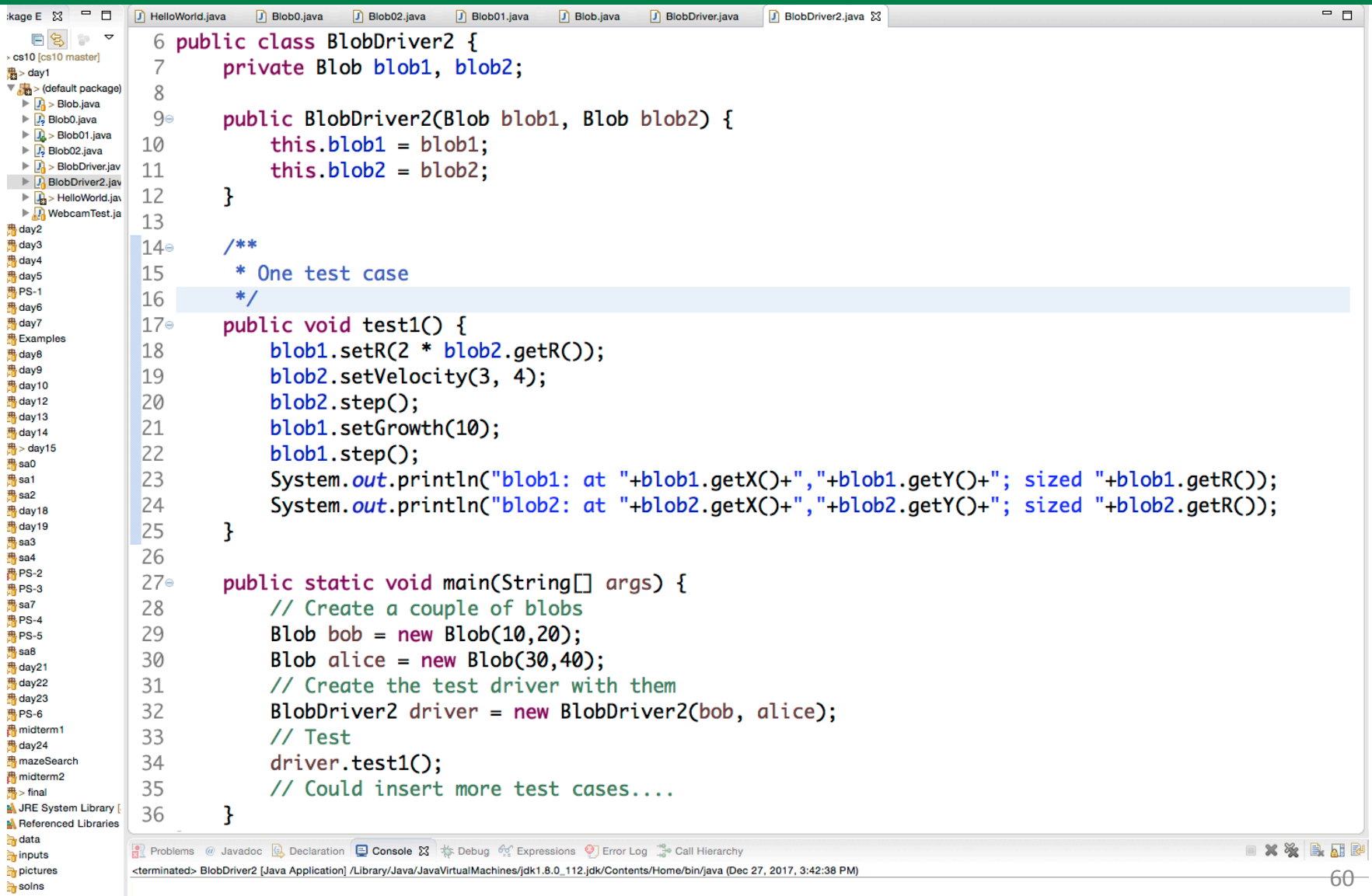
BlobDriver: Uses Blob class to create blob objects and then move/grow them



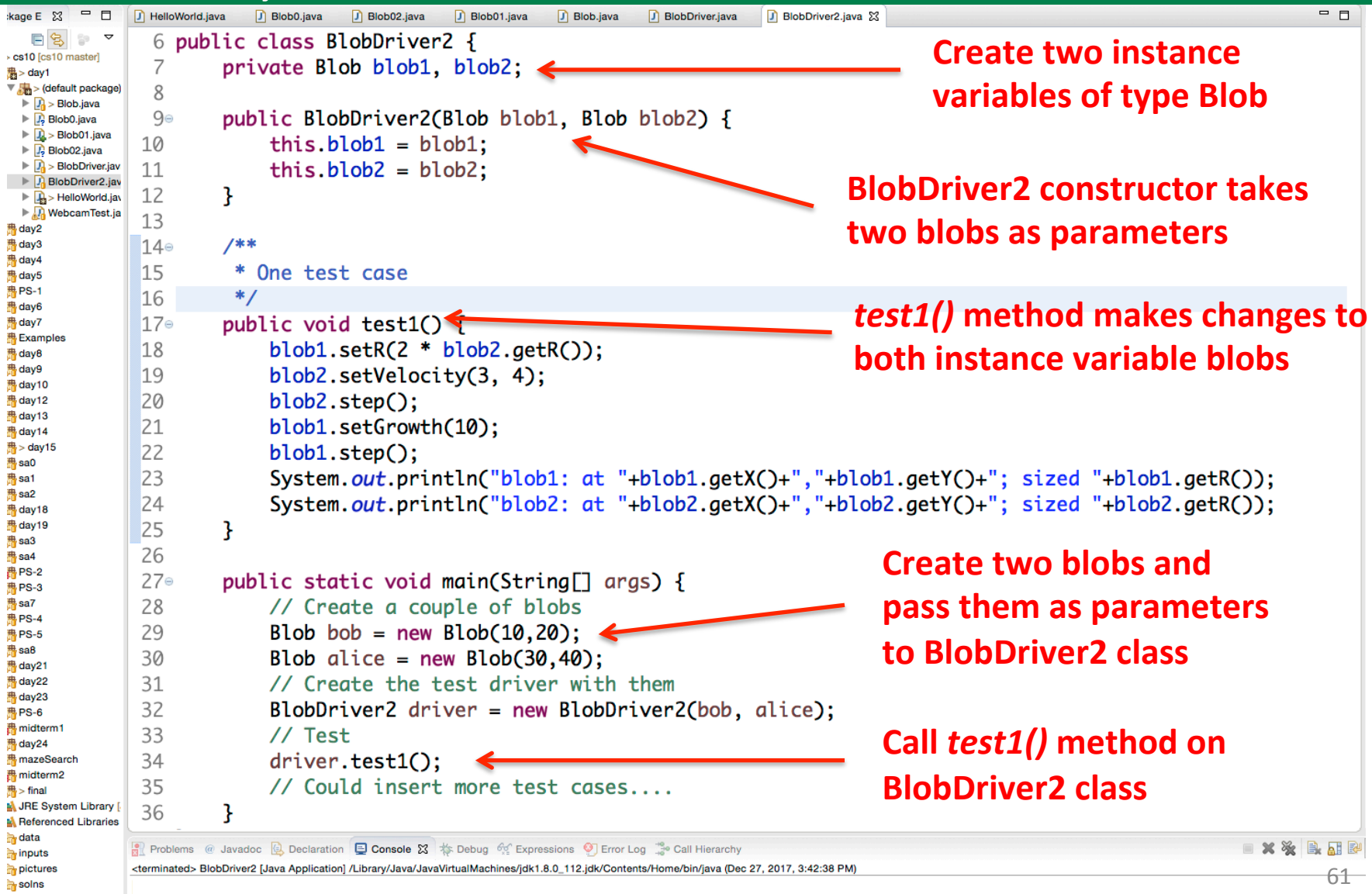
BlobDriver: Uses Blob class to create blob objects and then move/grow them



BlobDriver2.java



BlobDriver2: A driver program can, however, create classes it uses itself



Create two instance variables of type Blob

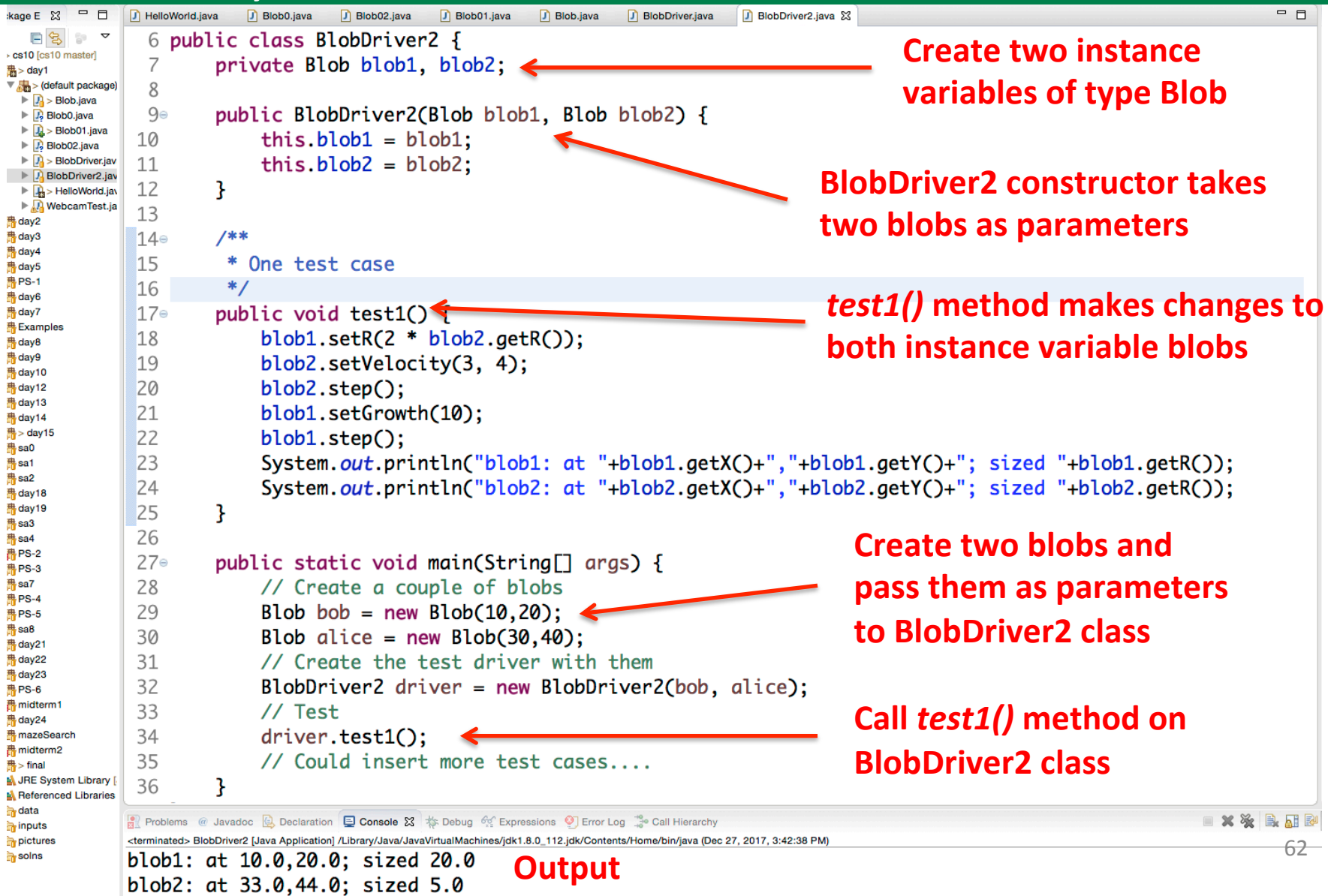
BlobDriver2 constructor takes two blobs as parameters

***test1()* method makes changes to both instance variable blobs**

Create two blobs and pass them as parameters to BlobDriver2 class

Call *test1()* method on BlobDriver2 class

BlobDriver2: A driver program can, however, create classes it uses itself



Short Assignment 0 (SA-0) is out, complete it before noon tomorrow

SA-0

- Find it on Canvas
- Take course survey to understand your background and assign you to a section
- Set up development environment (instructions and screen shots provided; I'm using Eclipse Neon)
- Create your first Java class
- Read course policies and honor code
- Please **complete survey before noon** tomorrow (or risk getting assigned to inconvenient section time!)