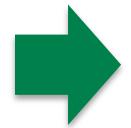


# CS 10: Problem solving via Object Oriented Programming

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, 6<sup>th</sup> 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

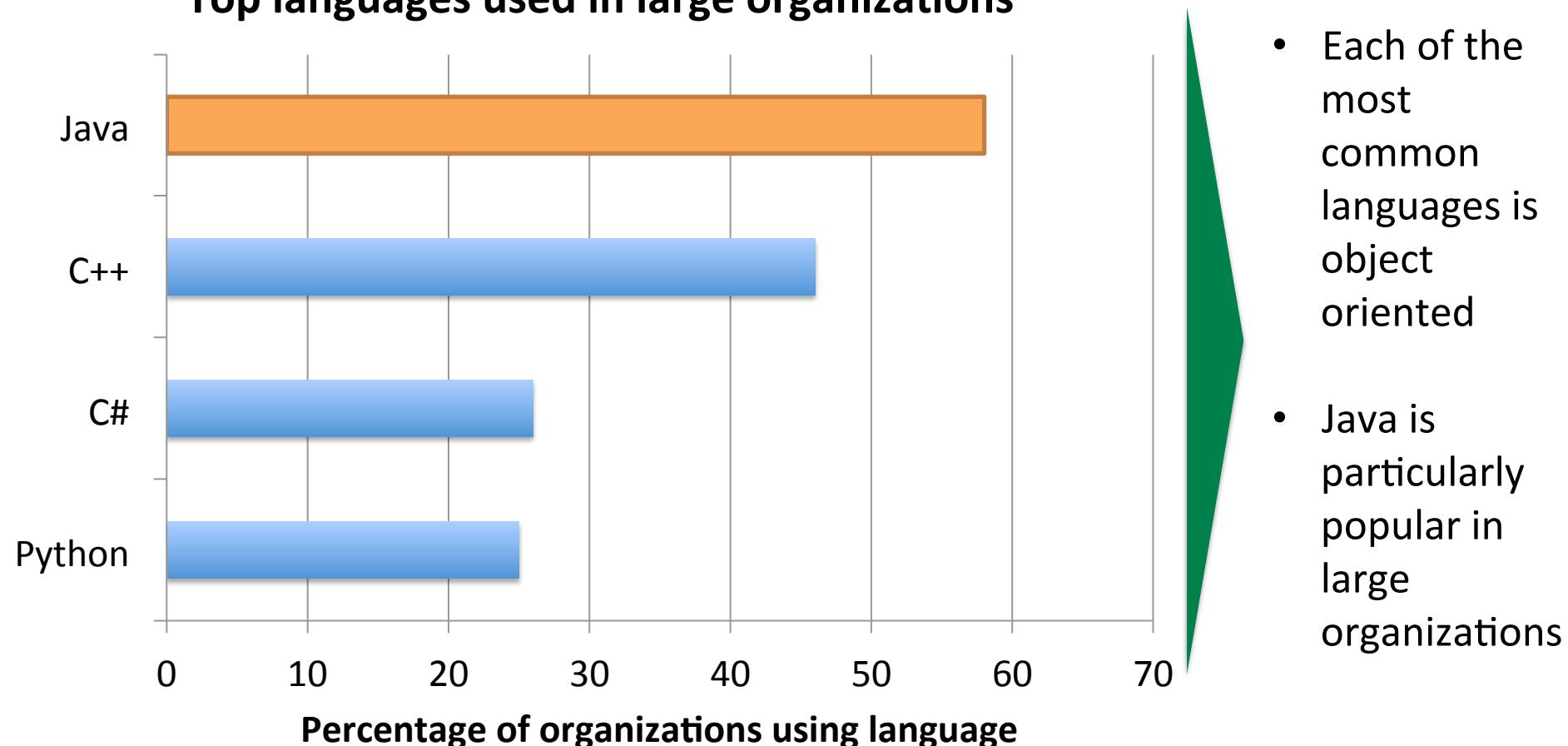
- Create “specialty” versions of objects that “inherit” functionality of parent, then customize behavior (more next class); reduces code redundancy

### 4. Polymorphism

- Same name, multiple meanings (more next class)

# OOP is popular, especially in large organizations

**Top languages used in large organizations**



# 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!**



# 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

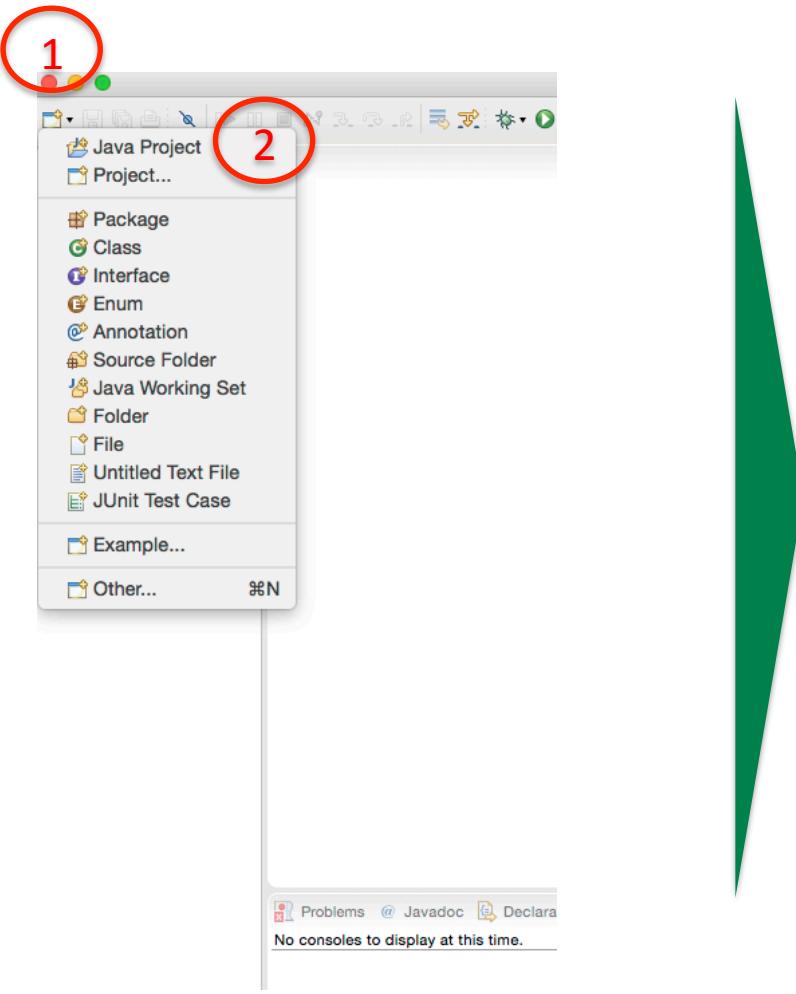
Add `System.out.println("Hello World")` to output to the console

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

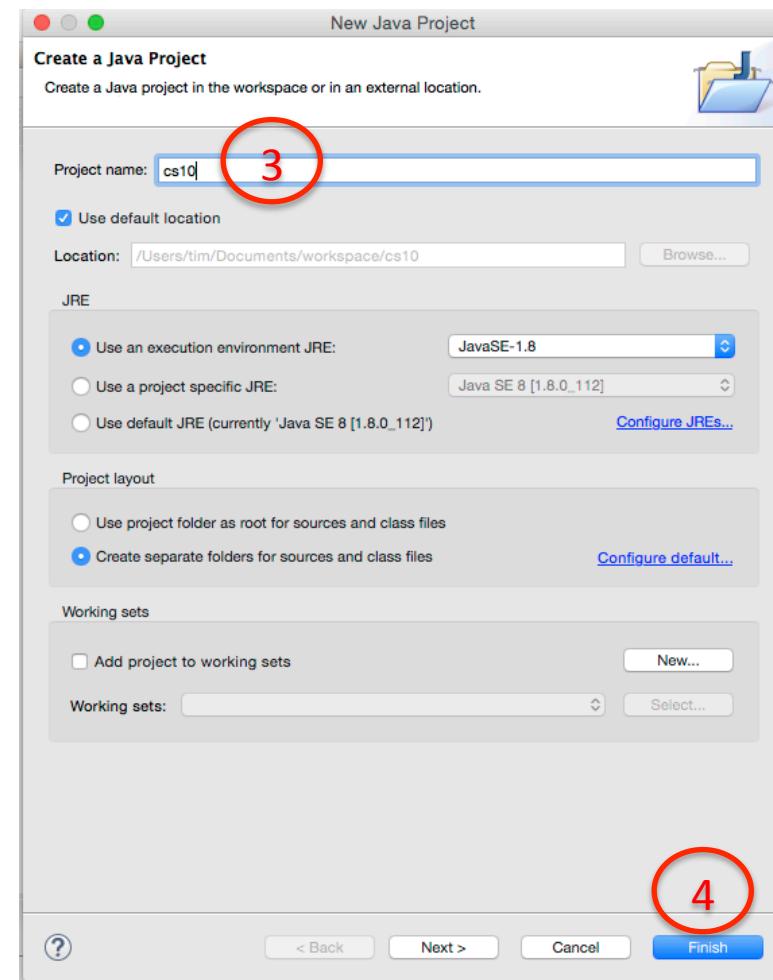
Can run using “Debug” button

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

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

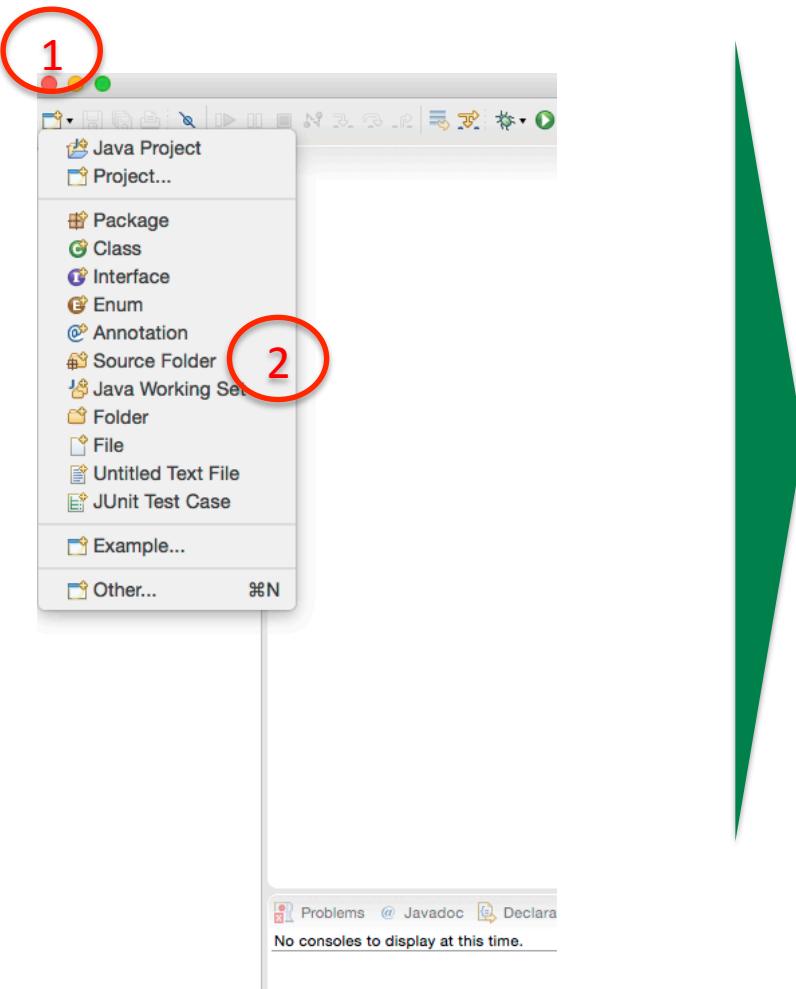


3. Enter “cs10” as Project name
4. Click “Finish”

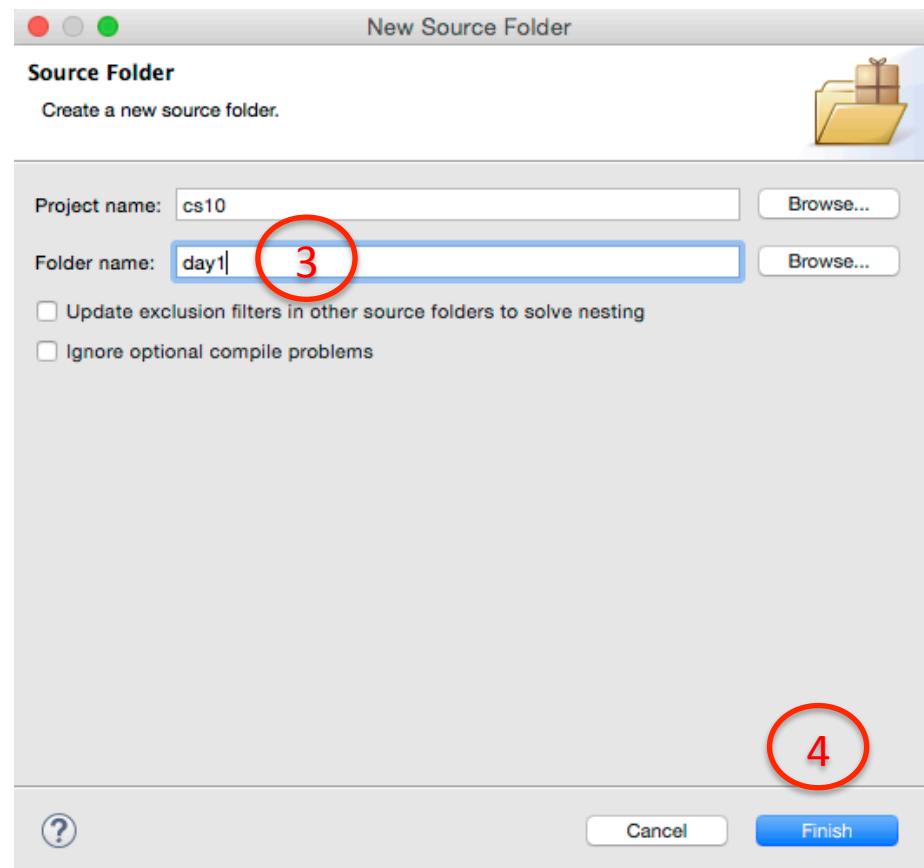


# 2. Create “day1” Source folder to hold your source code for day one of class

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

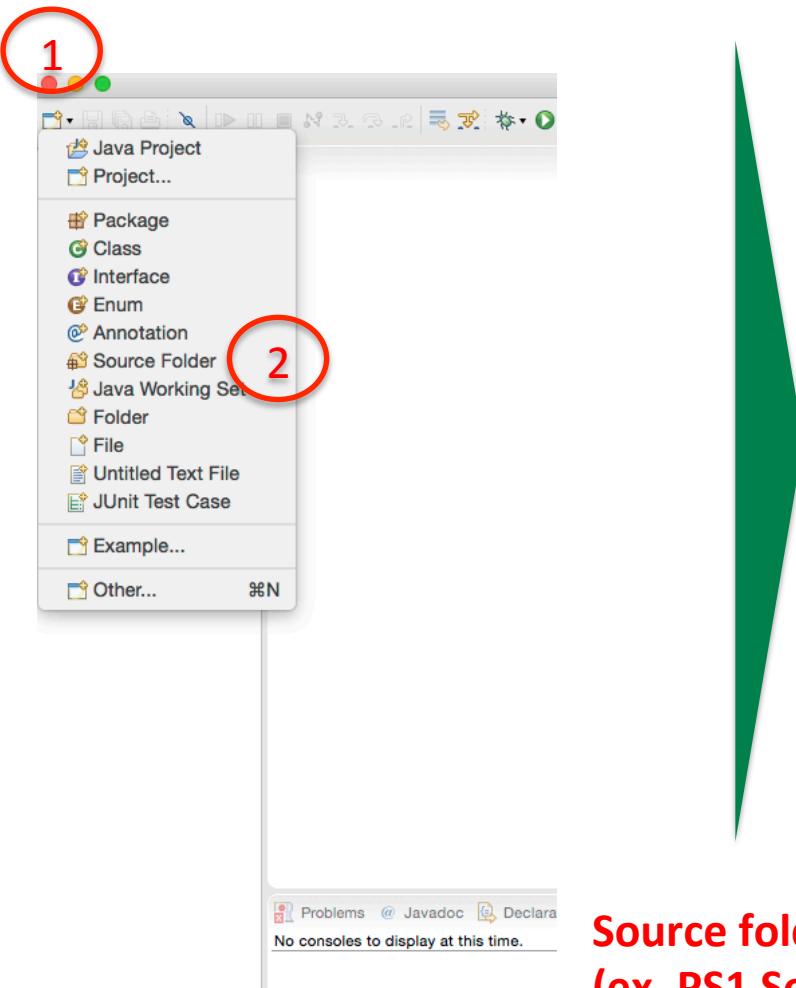


3. Enter Folder name (“day1”)
4. Click “Finish”

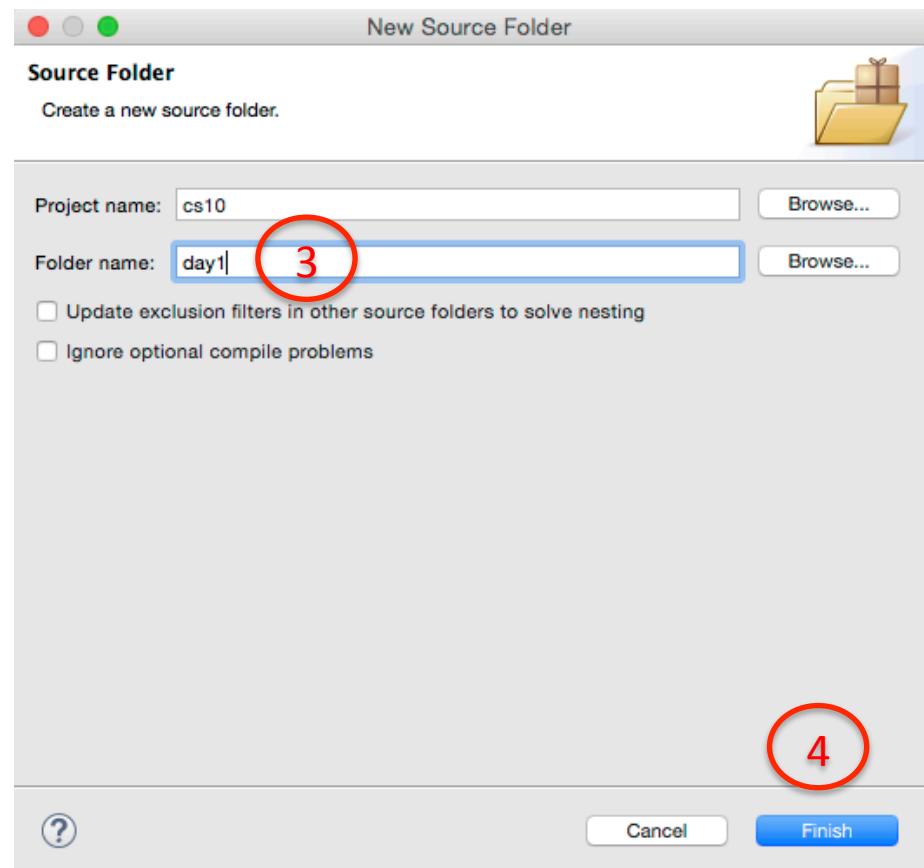


# 2. Create “day1” Source folder to hold your source code for day one of class

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



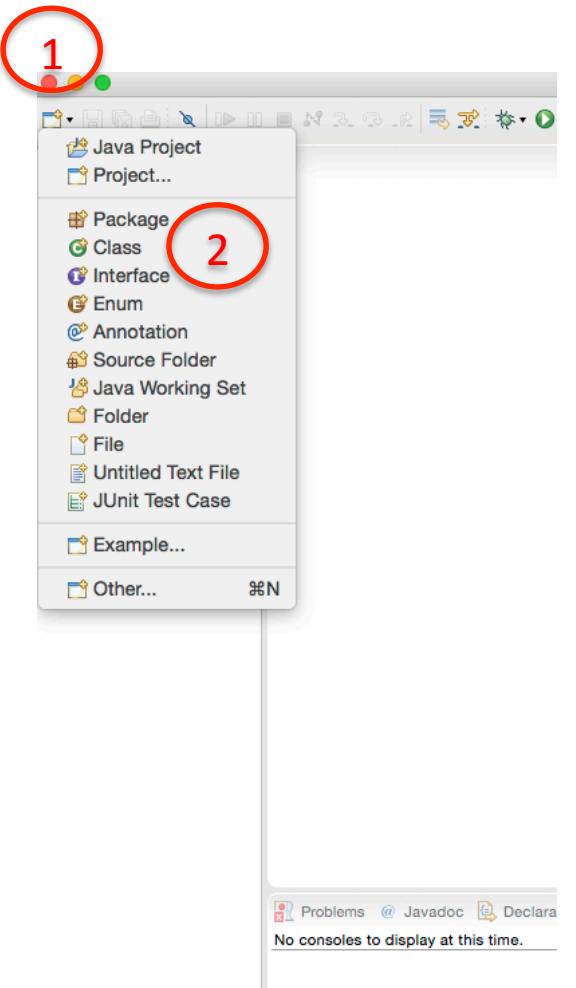
3. Enter Folder name (“day1”)
4. Click “Finish”



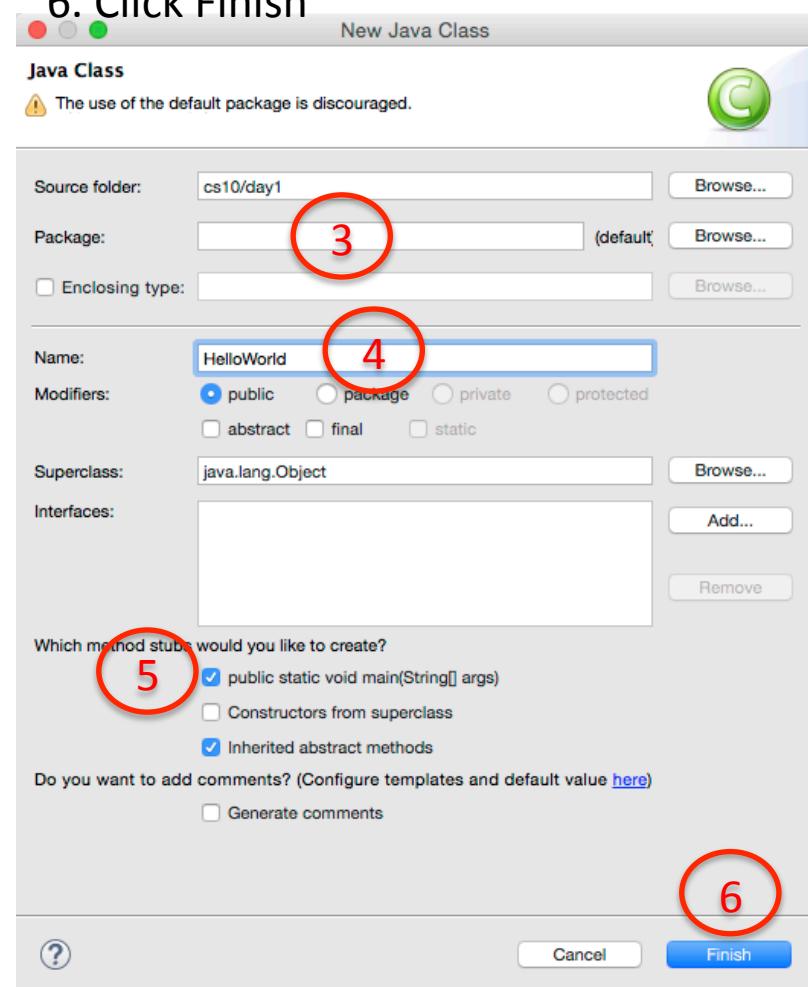
Source folders are a useful way to organize your code  
(ex. PS1 Source folder contains all code for Problem Set 1) 16

# 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

File on disk is `HelloWorld.java`

```
1 |
2 public class HelloWorld {
3
4     public static void main(String[] args) {
5         // TODO Auto-generated method stub
6     }
7 }
8
9 }
10 }
```

Class is named `HelloWorld`

- `main()` is where a Java program starts running
- Eclipse “stubs it out” for us if we check the box on previous slide

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

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer (left):** Shows a project named "cs10 [cs10 master]" containing several Java files like Blob.java, Blob0.java, etc., and some configuration files.
- Editor (center):** Displays the content of `HelloWorld.java`. The code includes Javadoc comments and a `main` method that prints "Hello World!" to the console.
- Bottom Bar:** Includes tabs for Problems, Javadoc, Declaration, Console, Debug, Expressions, Error Log, and Call Hierarchy. The Console tab is active, showing the output: <terminated> HelloWorld (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0\_112.jdk/Contents/Home/bin/java (Dec 28, 2017, 10:18:45 AM)

**Javadoc**

- Describes program (or method)
- Begins with “`/**`” ends with “`*/`”

Add tags such as  
“`@author`” or “`@param`”

- “`sys0`” + ctrl + space is shortcut to add `println()` command
- `println()` prints a line to the console

# Running the program prints “Hello World!” to console

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer (left):** Shows a Java project named "cs10" with several source files: Blob.java, Blob0.java, Blob01.java, Blob02.java, BlobDrive, BlobDrive, HelloWorld.java, and WebcamT.
- Editor Area (center):** Displays the content of `HelloWorld.java`. The code is a standard "Hello World" program.

```
1  /**
2  * Standard 'Hello World' first program
3  *
4  * @author Everyone who has ever written about programming and Tim Pierson too,
5  * Dartmouth CS 10, Winter 2018
6  */
7
8 public class HelloWorld {
9
10    public static void main(String[] args) {
11        System.out.println("Hello World!");
12    }
13
14 }
15
```
- Toolbar (top):** Contains various icons for file operations, code navigation, and toolbars.
- Run Configuration Buttons (top right):** Includes icons for Run, Stop, and Debug.
- Text in Red:** Two annotations are present:
  - "Run program by pressing Run button" with a red arrow pointing to the Run icon in the toolbar.
  - "Can also use ‘Debug’ to run" with a red arrow pointing to the Debug icon in the toolbar.
- Bottom Status Bar:** Shows tabs for Problems, Javadoc, Declaration, Console, Debug, Expressions, Error Log, and Call Hierarchy. The "Console" tab is active.
- Console Output (bottom):** Displays the output: "`<terminated> HelloWorld (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Dec 28, 2017, 10:18:45 AM)`".
- Page Number:** The bottom right corner shows the page number "20".

# Running the program prints “Hello World!” to console

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer (left):** Shows a project named "cs10 [cs10 master]" containing several Java files like Blob.java, Blob0.java, etc., and some configuration files.
- Code Editor (center):** Displays the content of `HelloWorld.java`. The code is a standard "Hello World" program with a copyright notice for Tim Pierson and Dartmouth CS 10, Winter 2018.
- Toolbar (top):** Contains various icons for file operations, code navigation, and toolbars.
- Run Configuration Buttons (top right):** Includes a "Run" button (green triangle) and a "Debug" button (green triangle with a dot).
- Annotations (red text):**
  - "Run program by pressing Run button" points to the green "Run" button.
  - "Can also use “Debug” to run" points to the green "Debug" button.
- Console (bottom):** Shows the output "Hello World!" and the status "- Bottom Status Bar:** Shows "Writable", "Smart Insert", "1 : 1", and other status indicators.

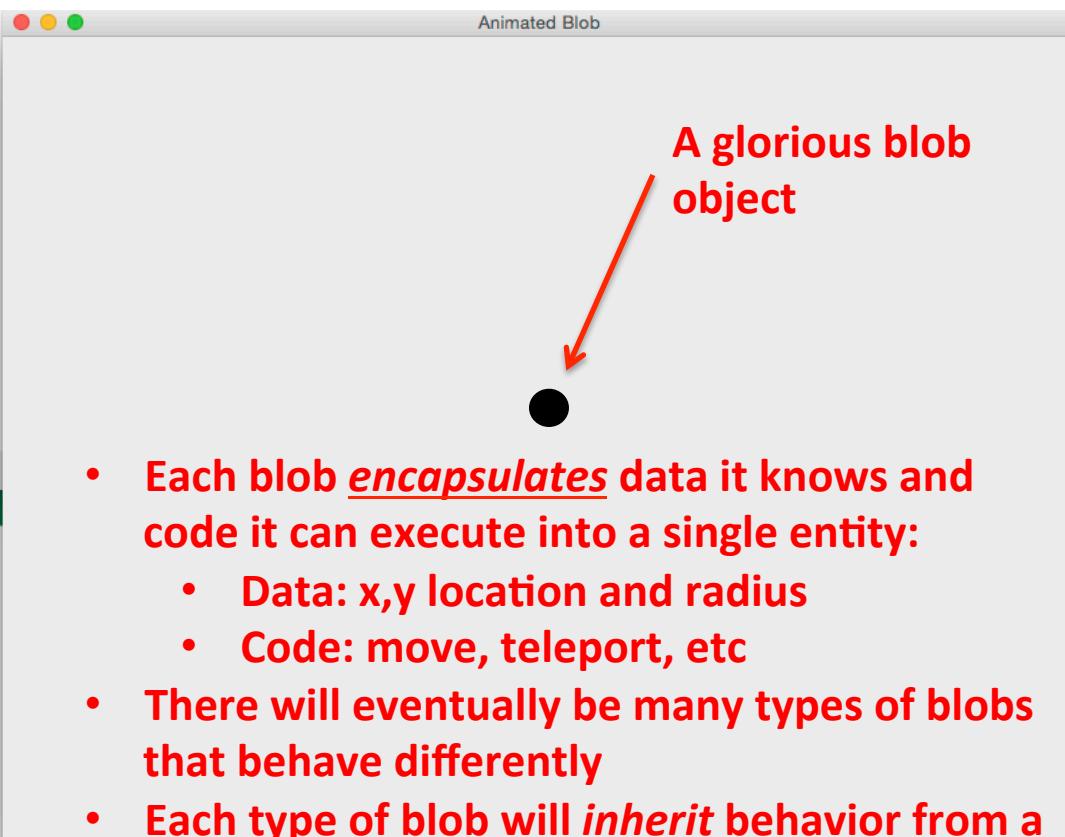
# Today we will focus on encapsulation

## 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**

A close-up photograph of an older man with white hair, wearing a red button-down shirt. He is smiling and pointing his right index finger directly at the viewer. In the background, there is a blurred interior setting with what appears to be a bar or restaurant environment.

# **SHOW US HOW IT WORKS**

[memegenerator.net](http://memegenerator.net)

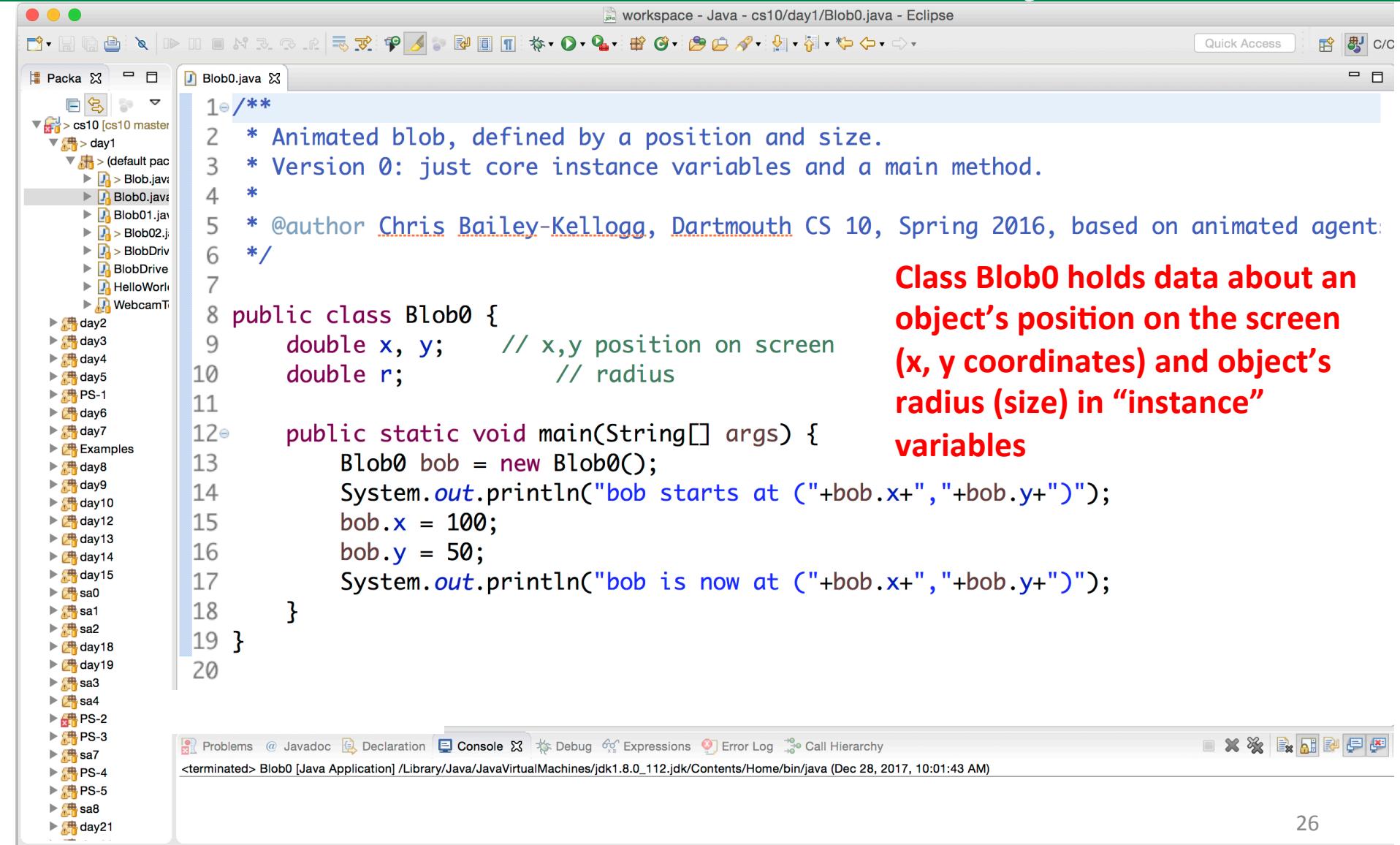
# Blob0.java

The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** workspace - Java - cs10/day1/Blob0.java - Eclipse
- Toolbar:** Standard Eclipse toolbar with various icons for file operations, search, and navigation.
- Left Sidebar:** Package Explorer view showing the project structure. The current file, `Blob0.java`, is selected in the list.
- Central Area:** Editor view displaying the `Blob0.java` code. The code defines a class `Blob0` with a constructor and a main method. It uses `System.out.println` to output the blob's position.
- Bottom Navigation:** Standard Eclipse navigation bar with tabs for Problems, Javadoc, Declaration, Console, Debug, Expressions, Error Log, and Call Hierarchy.
- Status Bar:** Shows the status "Writable" and the line number "1 : 1".

```
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
```

# Blob0: Our first “real” class uses instance variables to store data about objects



The screenshot shows the Eclipse IDE interface with the following details:

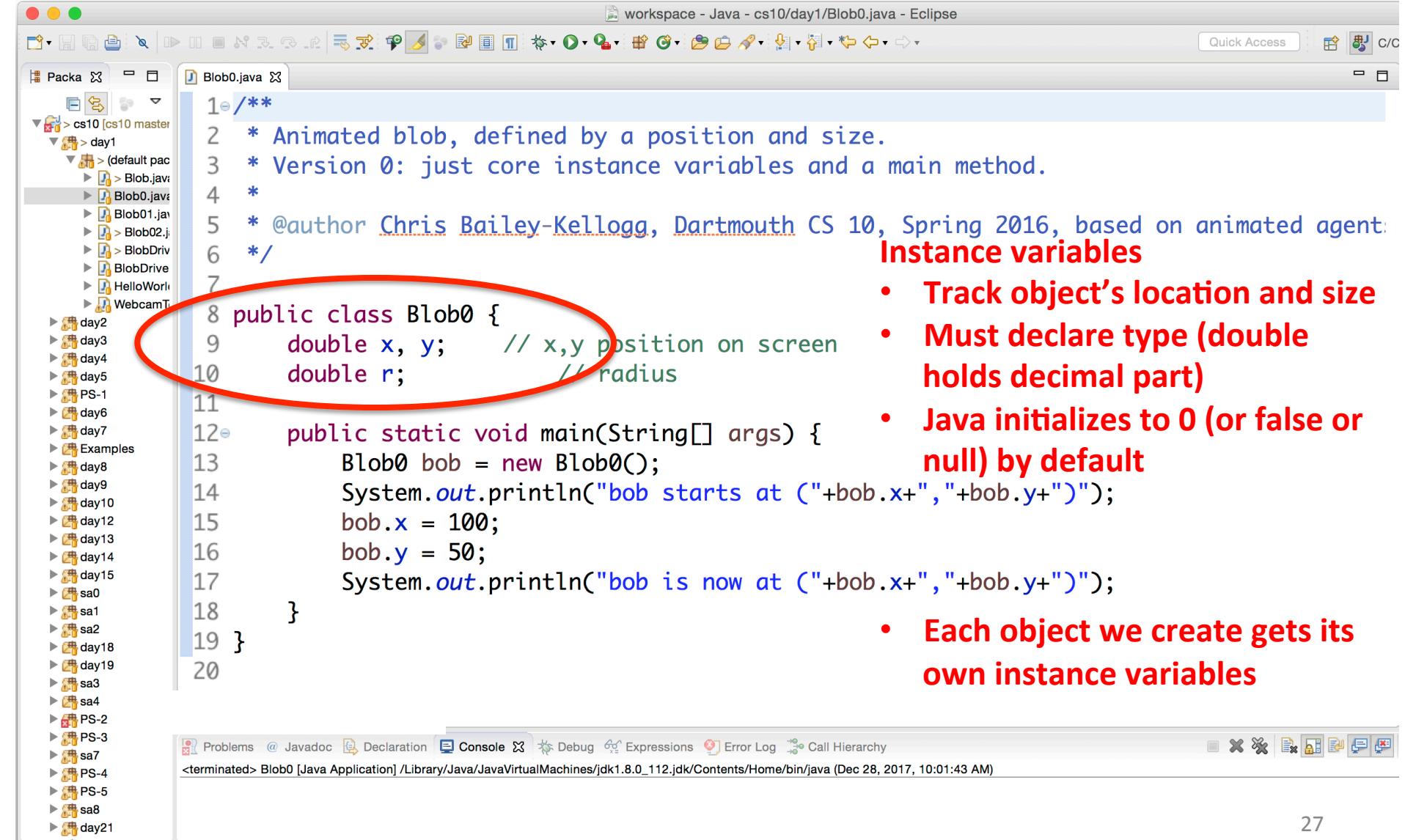
- Project Explorer (left):** Shows a project named "cs10 [cs10 master]" containing several Java files: Blob.java, Blob0.java, Blob01.java, Blob02.java, BlobDrive, BlobDrive, HelloWorld, and WebcamTest.
- Code Editor (center):** Displays the content of `Blob0.java`. The code defines a class `Blob0` with instance variables `x`, `y`, and `r`. It includes a `main` method that creates a `Blob0` object, prints its initial position, changes its `x` coordinate to 100, and then prints its new position.
- Console (bottom):** Shows the output of the `main` method:

```
bob starts at (100.0,50.0)
bob is now at (100.0,50.0)
```
- Status Bar (bottom right):** Shows the status "Writable" and "Smart Insert".

**Text on the right side of the slide:**

**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



The screenshot shows the Eclipse IDE interface with the following details:

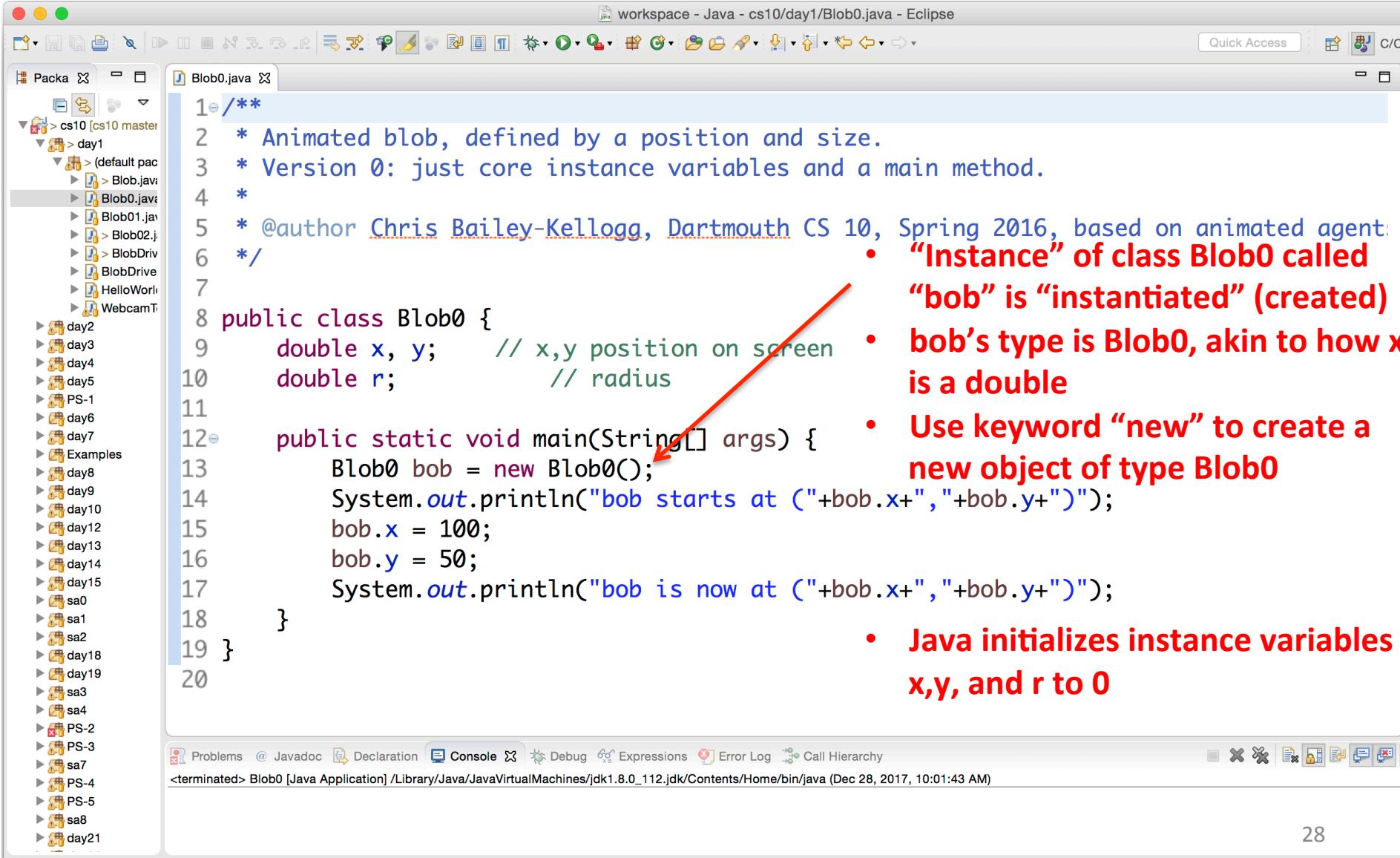
- Project Explorer (left):** Shows the project structure under "cs10 [cs10 master]". The "day1" folder contains several Java files: Blob.java, Blob0.java, Blob01.java, Blob02.java, BlobDrv, BlobDrive, HelloWorld, and WebcamT.
- Code Editor (center):** Displays the content of `Blob0.java`. A red oval highlights the class definition and its instance variables (`x`, `y`, `r`).

```
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
```
- Console (bottom):** Shows the output of the application: "bob starts at (100,50)" followed by "bob is now at (100,50)".

**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



The screenshot shows the Eclipse IDE interface with the following details:

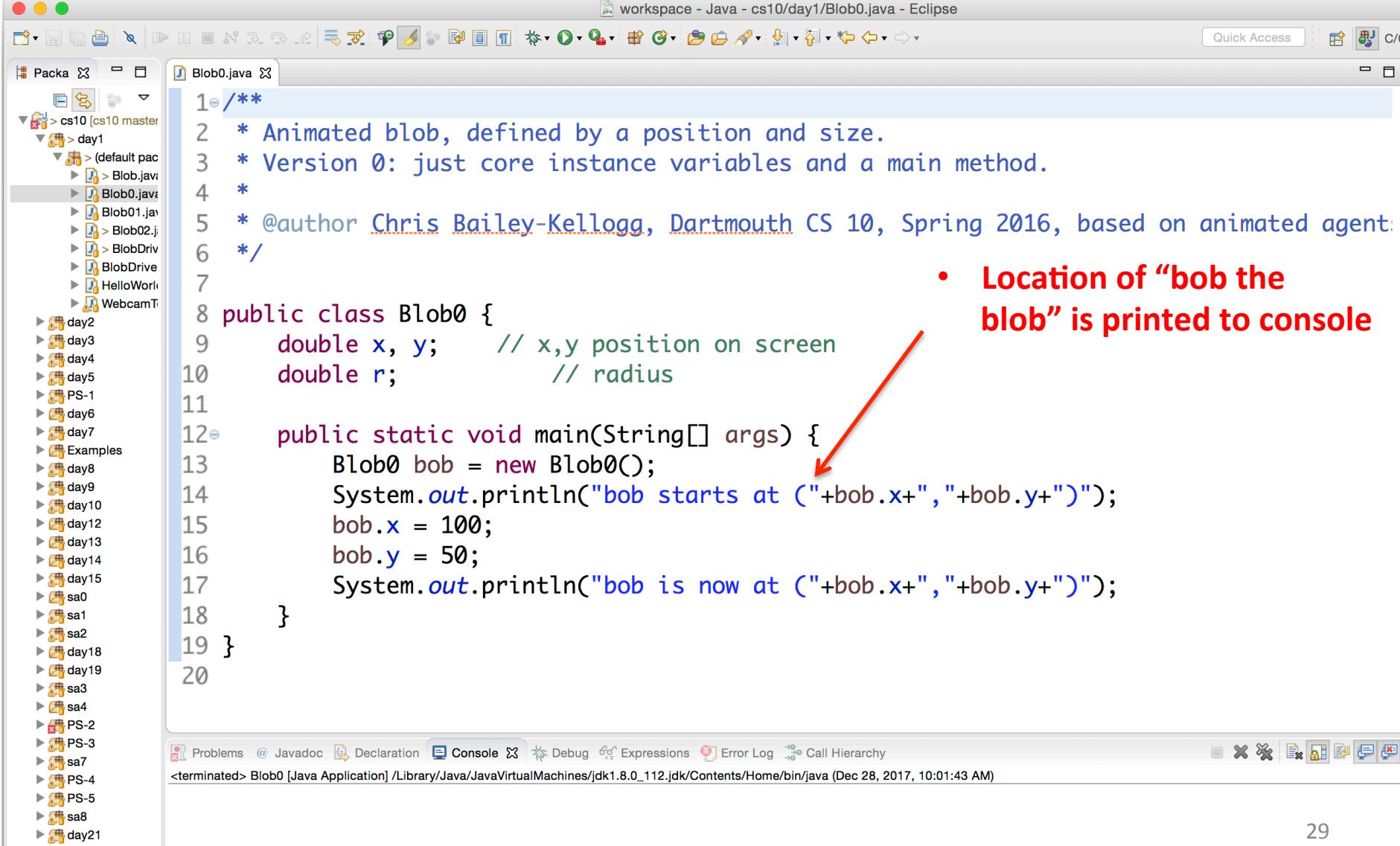
- Project Explorer (left):** Shows a project named "cs10 [cs10 master]" containing several Java files: Blob.java, Blob0.java, Blob01.java, Blob02.java, BlobDrive, BlobDrive, HelloWorld, and WebcamTest.
- Code Editor (center):** Displays the content of `Blob0.java`. The code defines a class `Blob0` with instance variables `x`, `y`, and `r`, and a `main` method that creates a `Blob0` object named `bob`.
- Console (bottom):** Shows the output of the application: "blob starts at (100,50)" followed by "blob is now at (100,50)".

A red arrow points from the text "Java initializes instance variables `x,y`, and `r` to 0" to the line of code where `bob` is initialized.

**Annotations (red text):**

- “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

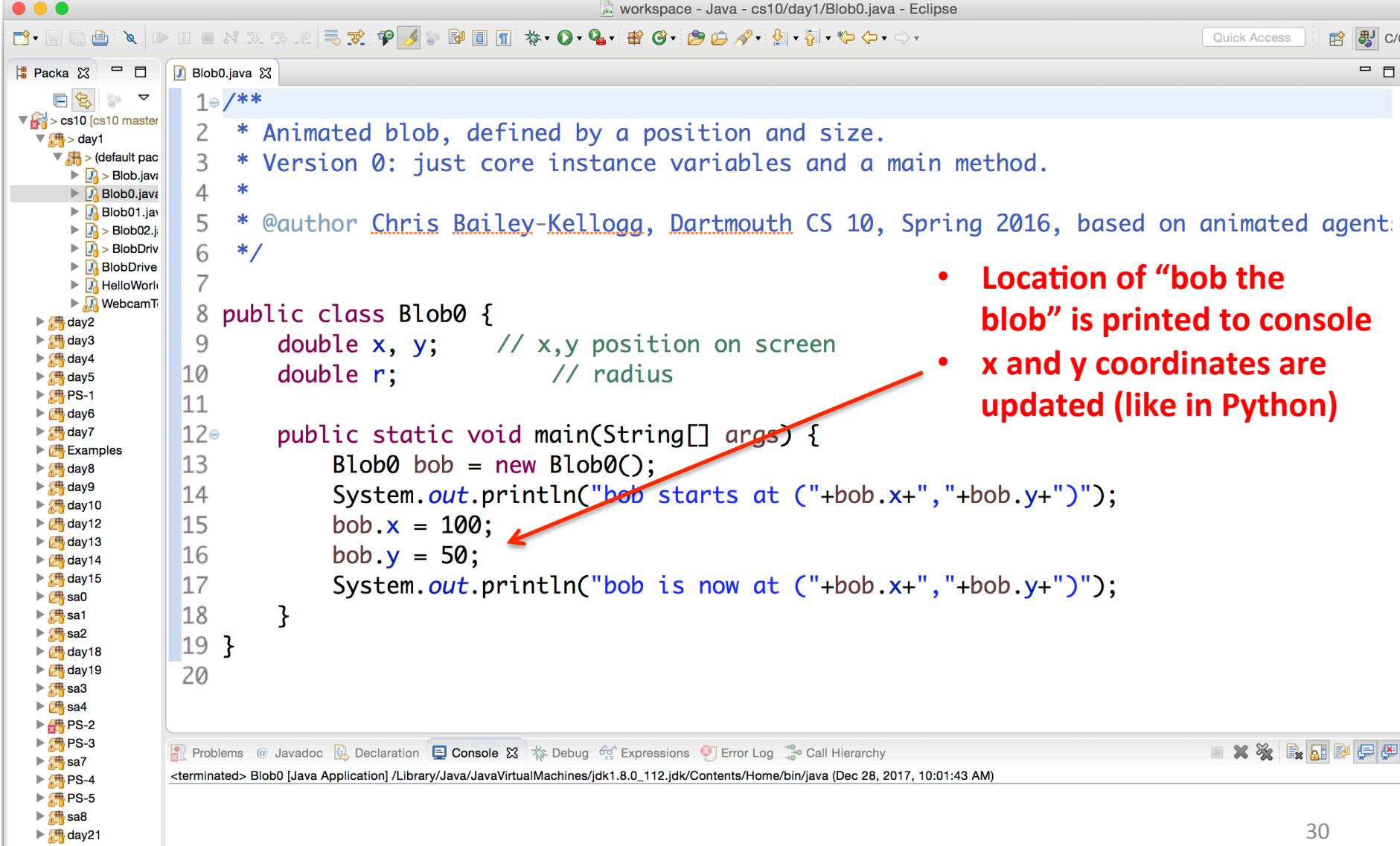


The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer (left):** Shows the workspace structure under "cs10 [cs10 master]". The "day1" folder contains several Java files: Blob.java, Blob0.java, Blob01.java, Blob02.java, BlobDrive, BlobDrive, HelloWorld, and WebcamTest.
- Code Editor (center):** Displays the content of `Blob0.java`. The code defines a class `Blob0` with instance variables `x`, `y`, and `r`. It includes a `main` method that creates a `Blob0` object, prints its initial position, changes its position, and prints its new position.
- Console (bottom):** Shows the output of the application's execution, indicating it was terminated at `blob0 [Java Application]` with the timestamp `(Dec 28, 2017, 10:01:43 AM)`.
- Annotations (right):** A red arrow points from the text "Location of ‘bob the blob’ is printed to console" to the line `System.out.println("bob starts at ("+bob.x+","+bob.y+");")` in the code editor.
- Text on the right:** The text "• Location of ‘bob the blob’ is printed to console" is displayed in red.

```
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
```

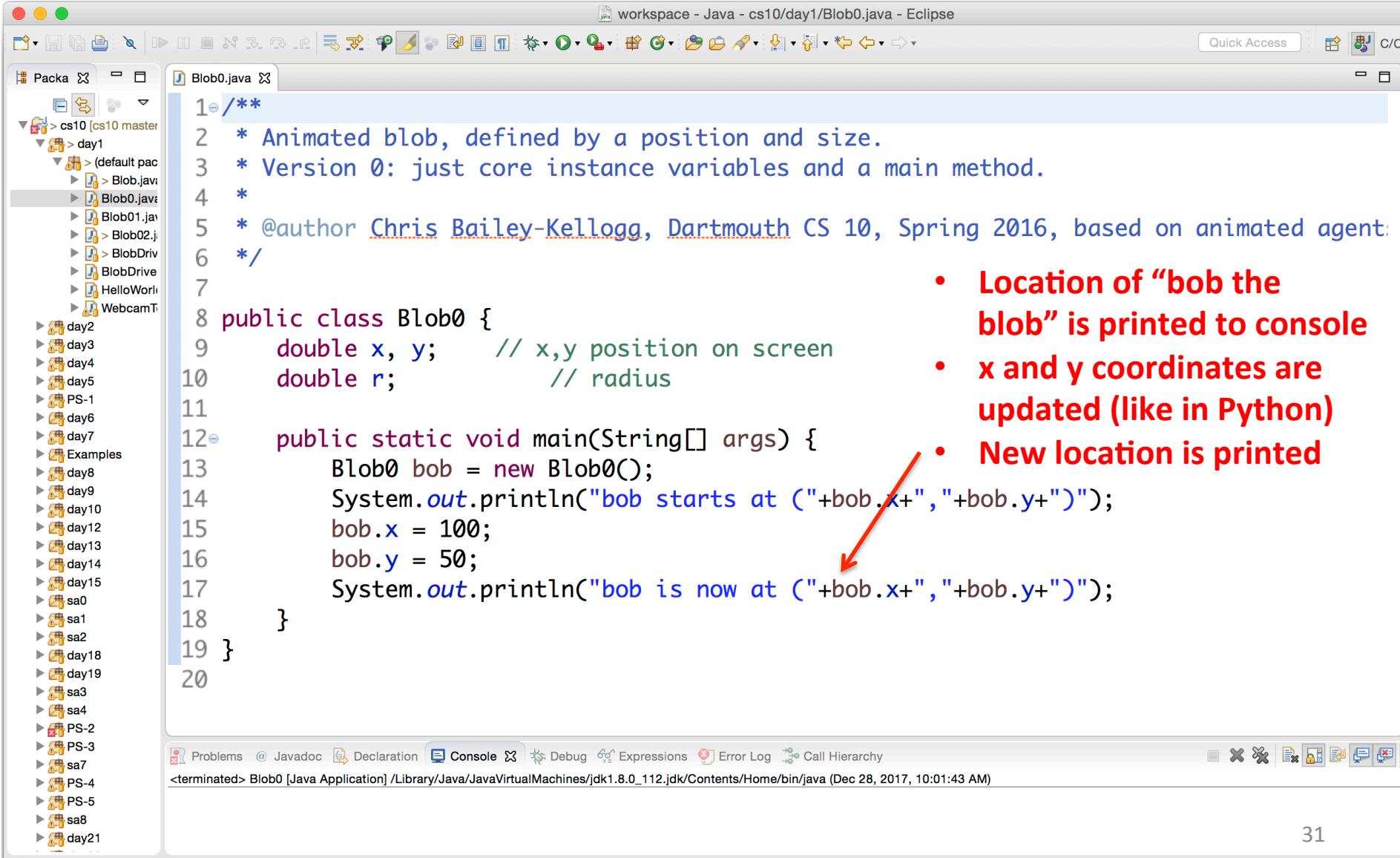
# Blob0: Our first “real” class uses instance variables to store data about objects



The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer (left):** Shows the project structure under "cs10 [cs10 master]". The "day1" folder contains several Java files: Blob.java, Blob0.java, Blob01.java, Blob02.java, BlobDrive, BlobDrive, HelloWorld, and WebcamTest.
- Code Editor (center):** Displays the content of `Blob0.java`. The code defines a class `Blob0` with instance variables `x` and `y`, and a main method that prints the blob's position and updates its coordinates to (100, 50).
- Console (bottom):** Shows the output of the application: "bob starts at (0,0)" followed by "bob is now at (100,50)".
- Annotations (right):** A red arrow points from the text "x and y coordinates are updated (like in Python)" to the line `bob.y = 50;` in the code editor.
- List-Group (red text, right):**
  - 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



The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer (left):** Shows the project structure under "cs10 [cs10 master]". The "day1" folder contains several Java files: Blob.java, Blob0.java, Blob01.java, Blob02.java, BlobDrive, BlobDrive, HelloWorld, and WebcamTest.
- Code Editor (center):** Displays the content of `Blob0.java`. The code defines a class `Blob0` with instance variables `x` and `y`, and a main method that prints the blob's position and updates it to (100, 50).
- Console (bottom):** Shows the output of the application: "bob starts at (0,0)" followed by "bob is now at (100,50)".
- Right side (list):** A red arrow points from the text "New location is printed" to the second `System.out.println` statement in the code editor.

```
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
```

- 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

The screenshot shows the Eclipse IDE interface with the following details:

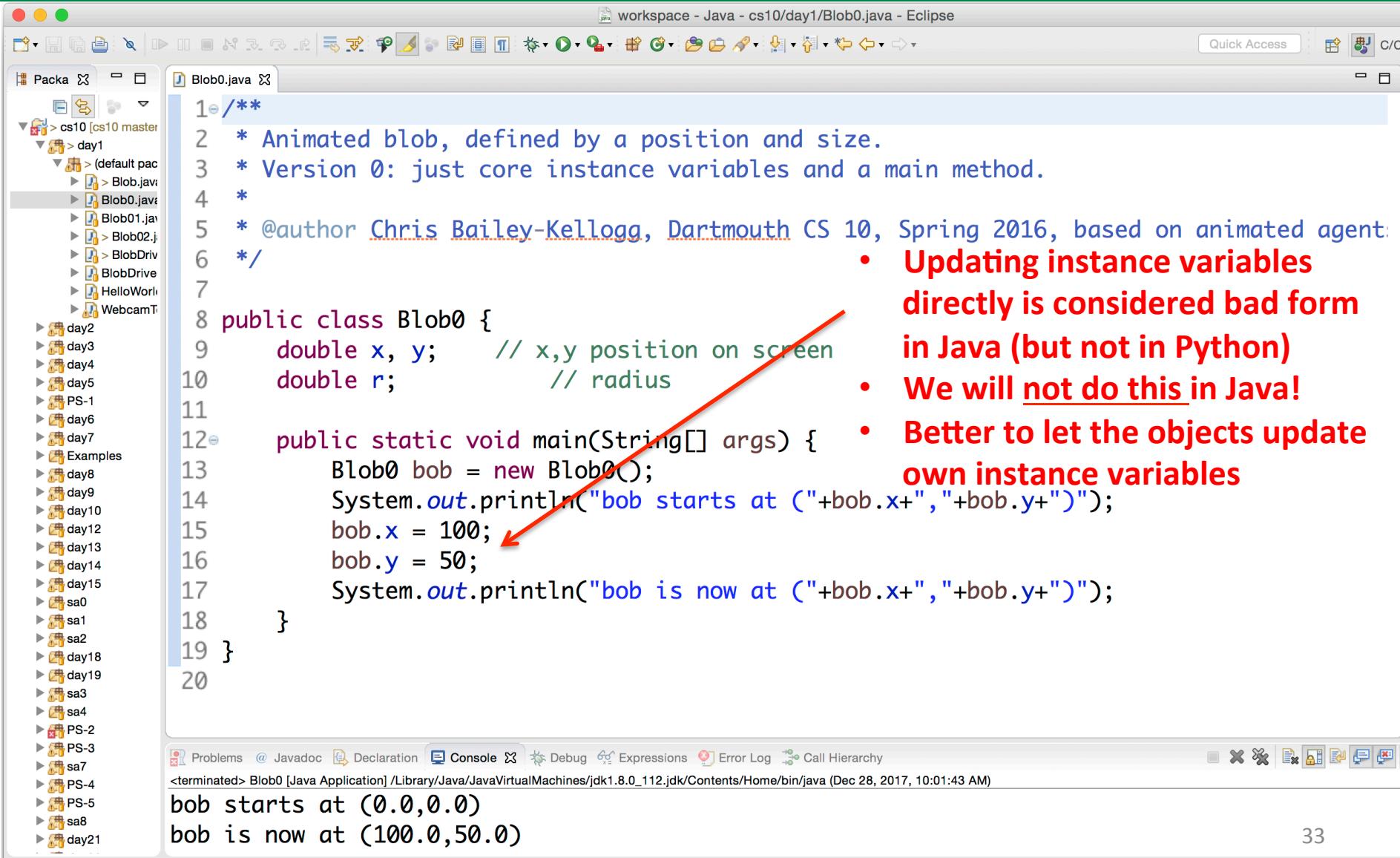
- Project Explorer (left):** Shows the project structure under "cs10 [cs10 master]". The "day1" folder contains several Java files: Blob.java, Blob0.java, Blob01.java, Blob02.java, BlobDrive, BlobDrive, HelloWorld, and WebcamTest.
- Code Editor (center):** Displays the content of `Blob0.java`. The code defines a class `Blob0` with instance variables `x` and `r`, and a `main` method that creates a `Blob0` object, prints its initial position, changes its coordinates, and prints its new position.
- Console (bottom):** Shows the terminal output of the application. It prints two lines:

```
bob starts at (0.0,0.0)
bob is now at (100.0,50.0)
```
- Annotations (right):** A red arrow points from the text "New location is printed" to the second `System.out.println` statement in the code editor.
- Text Overlay (bottom right):** The text "Output appears in console" is displayed in red.

- 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!



The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer (left):** Shows a project named "cs10" with several Java files: Blob.java, Blob0.java, Blob01.java, Blob02.java, BlobDrive, BlobDrive, HelloWorld, and WebcamTest.
- Code Editor (center):** Displays the content of `Blob0.java`. The code defines a class `Blob0` with instance variables `x` and `y`, and a main method that prints the blob's position and then updates its `x` and `y` values directly.
- Console (bottom):** Shows the output of the program: "bob starts at (0.0,0.0)" followed by "bob is now at (100.0,50.0)".
- Annotations (right side):** A red arrow points from the text "Updating instance variables directly is considered bad form in Java (but not in Python)" to the line `bob.x = 100;` in the code editor.
- List of bullet points (right side):**
  - 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

The screenshot shows the Eclipse IDE interface with the following details:

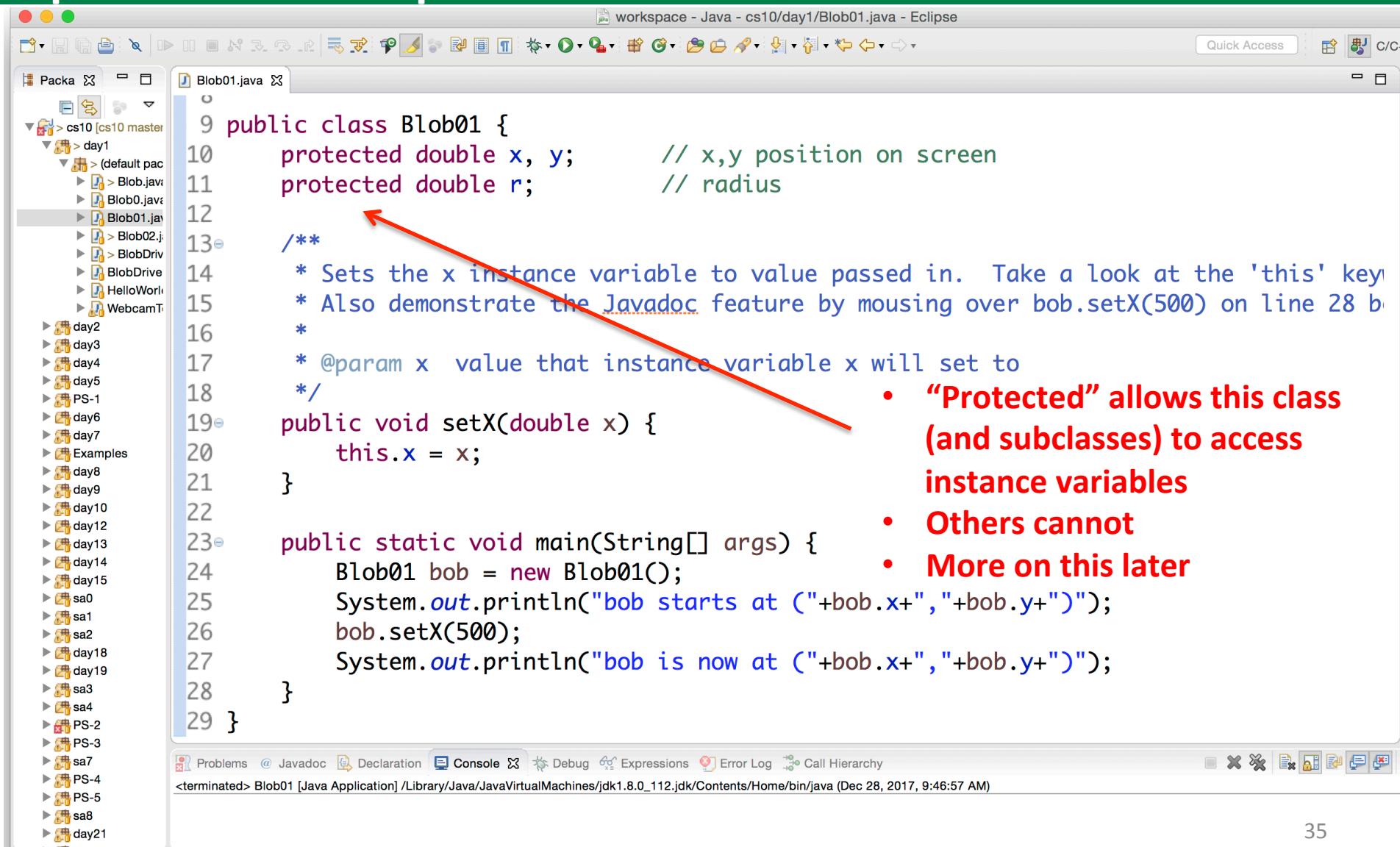
- Title Bar:** workspace - Java - cs10/day1/Blob01.java - Eclipse
- Toolbar:** Standard Eclipse toolbar with various icons for file operations, search, and navigation.
- Left Sidebar (Package Explorer):** Shows the project structure under "cs10 [cs10 master]". The "day1" folder contains several Java files: Blob.java, Blob0.java, Blob01.java (selected), Blob02.java, BlobDrive, BlobDrive, HelloWorld, and WebcamT.
- Central Editor Area:** Displays the content of the Blob01.java file. The code defines a class Blob01 with instance variables x and y (protected) and radius (protected). It includes a setX method with Javadoc comments and a main method that creates a Blob01 object, sets its x position to 500, and prints its current position twice.
- Bottom Status Bar:** Shows tabs for Problems, Javadoc, Declaration, Console (selected), Debug, Expressions, Error Log, and Call Hierarchy. The status bar also indicates the application is Writable and shows the line number 1:1.
- Bottom Status Bar (Details):** Shows the terminal output: <terminated> Blob01 [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0\_112.jdk/Contents/Home/bin/java (Dec 28, 2017, 9:46:57 AM)

```
public class Blob01 {
    protected double x, y;          // x,y position on screen
    protected double r;             // radius

    /**
     * Sets the x instance variable to value passed in. Take a look at the 'this' keyword
     * Also demonstrate the Javadoc feature by mousing over bob.setX(500) on line 28 below
     *
     * @param x  value that instance variable x will set to
     */
    public void setX(double x) {
        this.x = x;
    }

    public static void main(String[] args) {
        Blob01 bob = new Blob01();
        System.out.println("bob starts at ("+bob.x+","+bob.y+")");
        bob.setX(500);
        System.out.println("bob is now at ("+bob.x+","+bob.y+")");
    }
}
```

# Blob01: Declaring instance variables as “protected” prevents outside modification



The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer (left):** Shows the project structure under "cs10 [cs10 master]". The "day1" folder contains files: Blob.java, Blob0.java, Blob01.java (selected), Blob02.java, BlobDrive.java, HelloWorld.java, and WebcamTest.java.
- Code Editor (center):** Displays the content of `Blob01.java`.

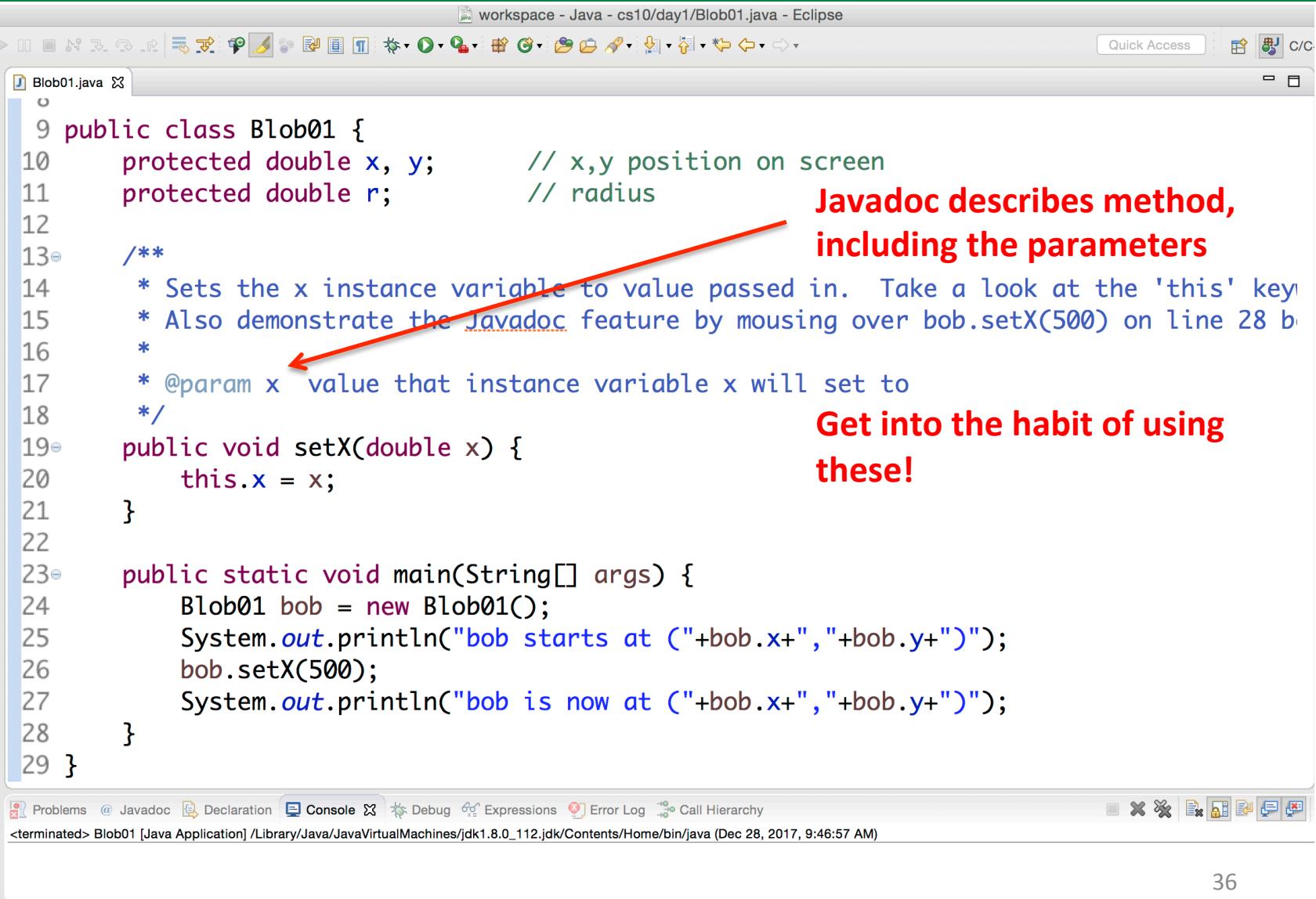
```
public class Blob01 {  
    protected double x, y;          // x,y position on screen  
    protected double r;            // radius  
  
    /**  
     * Sets the x instance variable to value passed in. Take a look at the 'this' keyword.  
     * Also demonstrate the Javadoc feature by mousing over bob.setX(500) on line 28 below.  
     *  
     * @param x  value that instance variable x will set to  
     */  
    public void setX(double x) {  
        this.x = x;  
    }  
  
    public static void main(String[] args) {  
        Blob01 bob = new Blob01();  
        System.out.println("bob starts at ("+bob.x+","+bob.y+")");  
        bob.setX(500);  
        System.out.println("bob is now at ("+bob.x+","+bob.y+")");  
    }  
}
```

A red arrow points from the explanatory text in the Javadoc comment to the `this.x = x;` line.
- Bottom Status Bar:** Shows the status bar with tabs like Problems, Javadoc, Declaration, Console, Debug, Expressions, Error Log, and Call Hierarchy. It also displays the terminal output: <terminated> Blob01 [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0\_112.jdk/Contents/Home/bin/java (Dec 28, 2017, 9:46:57 AM).
- Bottom Right:** Includes icons for Writable, Smart Insert, and a 1:1 ratio button.

**List of Key Points (red text):**

- “Protected” allows this class (and subclasses) to access instance variables
- Others cannot
- More on this later

# Blob01: Javadoc allows you to document your methods



```
workspace - Java - cs10/day1/Blob01.java - Eclipse
Quick Access C/C

public class Blob01 {
    protected double x, y;          // x,y position on screen
    protected double r;             // radius

    /**
     * Sets the x instance variable to value passed in. Take a look at the 'this' keyword
     * Also demonstrate the Javadoc feature by mousing over bob.setX(500) on line 28 below
     *
     * @param x value that instance variable x will set to
     */
    public void setX(double x) {
        this.x = x;
    }

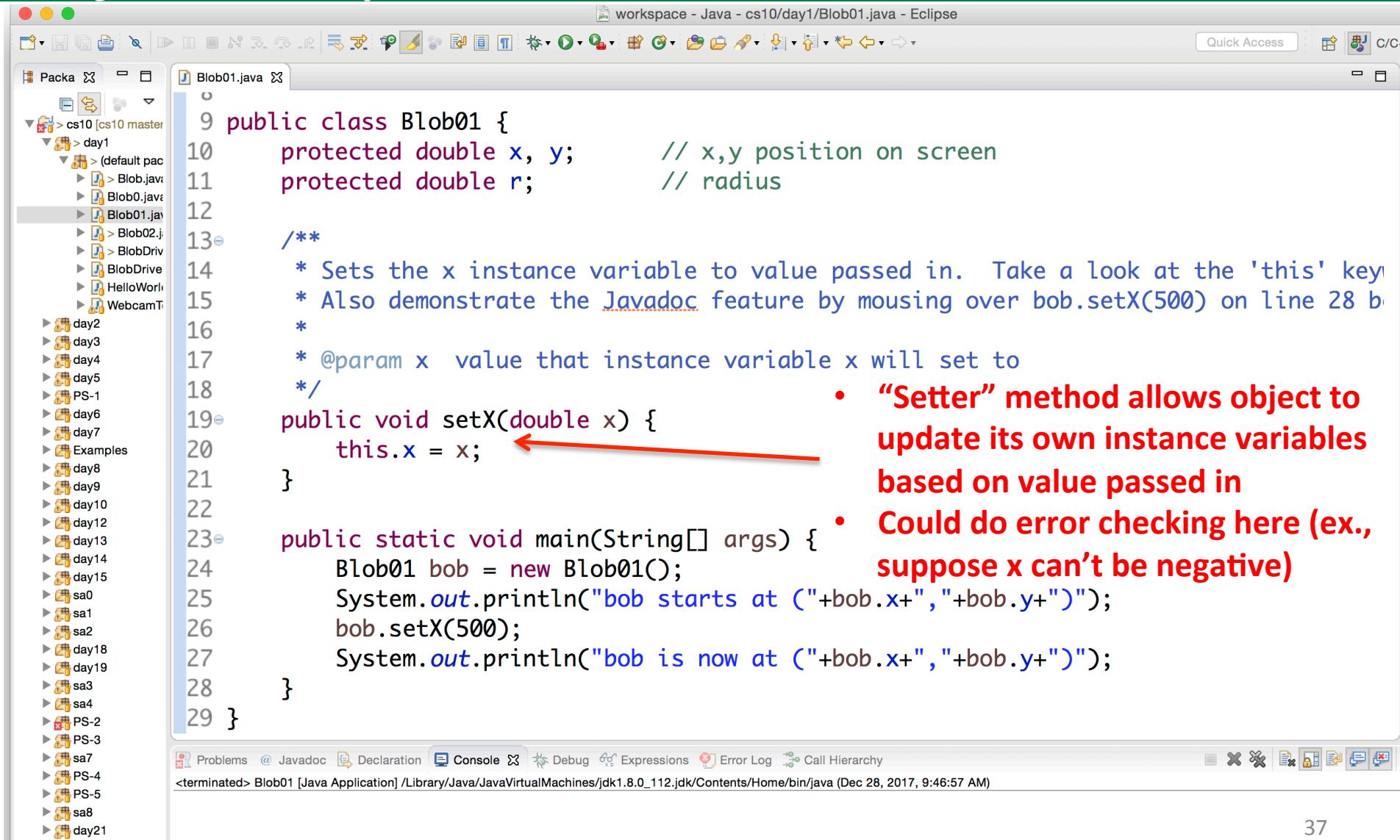
    public static void main(String[] args) {
        Blob01 bob = new Blob01();
        System.out.println("bob starts at ("+bob.x+","+bob.y+")");
        bob.setX(500);
        System.out.println("bob is now at ("+bob.x+","+bob.y+")");
    }
}
```

**Javadoc describes method, including the parameters**

**Get into the habit of using these!**

36

# Blob01: Add a “setter” method to allow the object to update its own instance variables



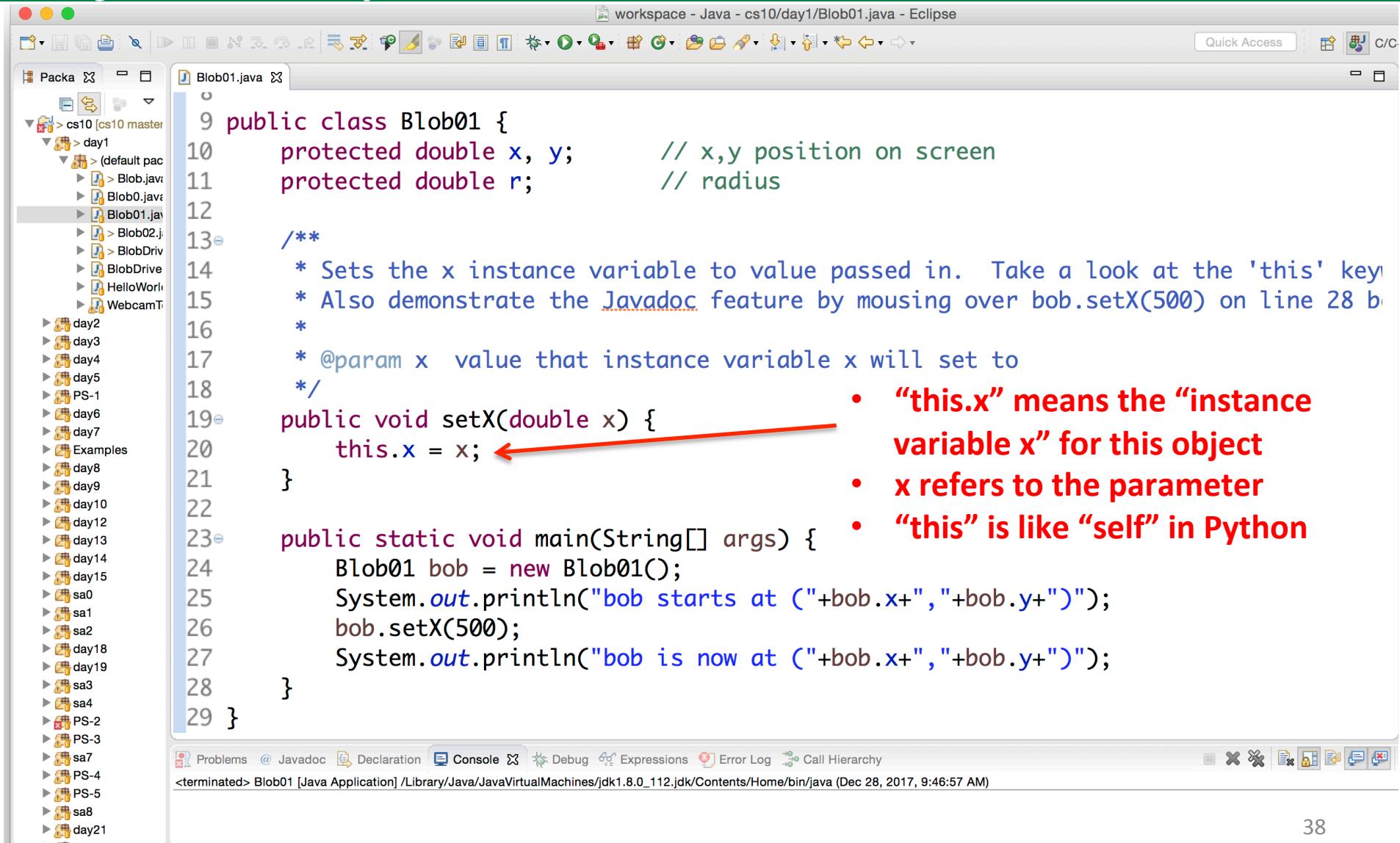
The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer (left):** Shows the project structure under "cs10 [cs10 master]". The "day1" folder contains files: Blob.java, Blob0.java, Blob01.java (selected), Blob02.java, BlobDrive, BlobDrive, HelloWorld, and WebcamT.
- Code Editor (center):** Displays the content of `Blob01.java`. The code defines a class `Blob01` with instance variables `x`, `y`, and `r`. It includes a Javadoc comment for the `setX` method and a `main` method demonstrating its use. A red arrow points to the `this.x = x;` line in the `setX` method.
- Bottom Status Bar:** Shows tabs for Problems, Javadoc, Declaration, Console, Debug, Expressions, Error Log, and Call Hierarchy. The status bar also indicates the application is terminated and provides the path to the Java VM and date/time.

```
public class Blob01 {  
    protected double x, y;          // x,y position on screen  
    protected double r;            // radius  
  
    /**  
     * Sets the x instance variable to value passed in. Take a look at the 'this' keyword.  
     * Also demonstrate the Javadoc feature by mousing over bob.setX(500) on line 28 below.  
     *  
     * @param x  value that instance variable x will set to  
     */  
    public void setX(double x) {  
        this.x = x;                ←  
    }  
  
    public static void main(String[] args) {  
        Blob01 bob = new Blob01();  
        System.out.println("bob starts at ("+bob.x+","+bob.y+")");  
        bob.setX(500);  
        System.out.println("bob is now at ("+bob.x+","+bob.y+")");  
    }  
}
```

- Listed Points (right):**
  - “Setter” method allows object to update its own instance variables based on value passed in
  - Could do error checking here (ex., suppose x can't be negative)

# Blob01: Add a “setter” method to allow the object to update its own instance variables



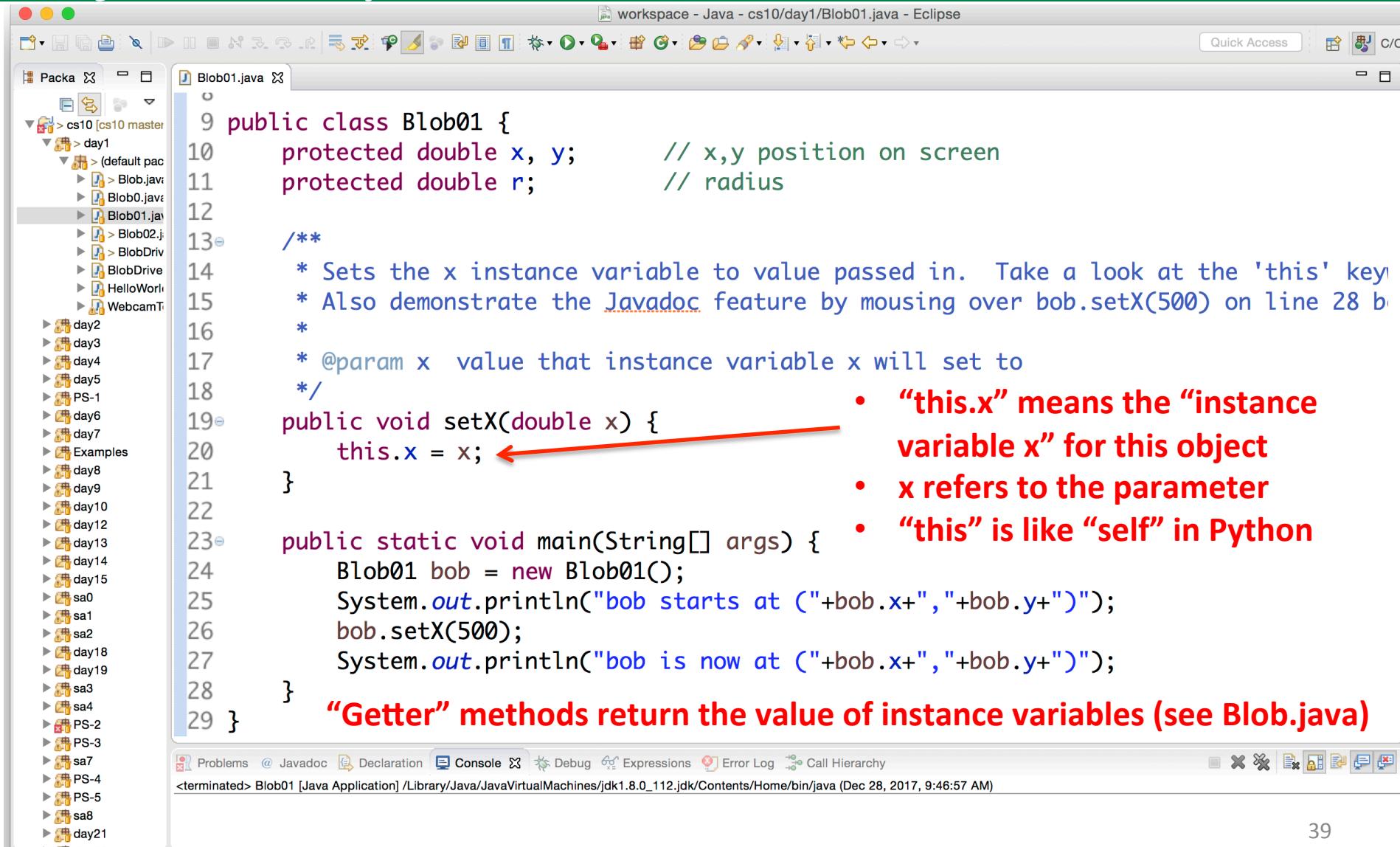
The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer (left):** Shows the project structure under "cs10 [cs10 master]". The "day1" folder contains files: Blob.java, Blob0.java, Blob01.java (selected), Blob02.java, BlobDrive, BlobDrive, HelloWorld, and WebcamT.
- Code Editor (center):** Displays the content of `Blob01.java`. The code defines a class `Blob01` with instance variables `x`, `y`, and `r`. It includes a Javadoc comment for the `setX` method and a `main` method that creates a `Blob01` object, prints its initial position, calls `setX(500)`, and prints its new position.
- Annotations (right):** A red arrow points from the `this.x = x;` line to a bulleted list explaining the meaning of `this.x`.
- Bottom Status Bar:** Shows tabs for Problems, Javadoc, Declaration, Console, Debug, Expressions, Error Log, and Call Hierarchy. The status bar also indicates the application is terminated and provides the Java version and date.

**Annotations (red text and arrow):**

- “**this.x**” means the “instance variable **x**” for this object
- **x** refers to the parameter
- “**this**” is like “**self**” in Python

# Blob01: Add a “setter” method to allow the object to update its own instance variables



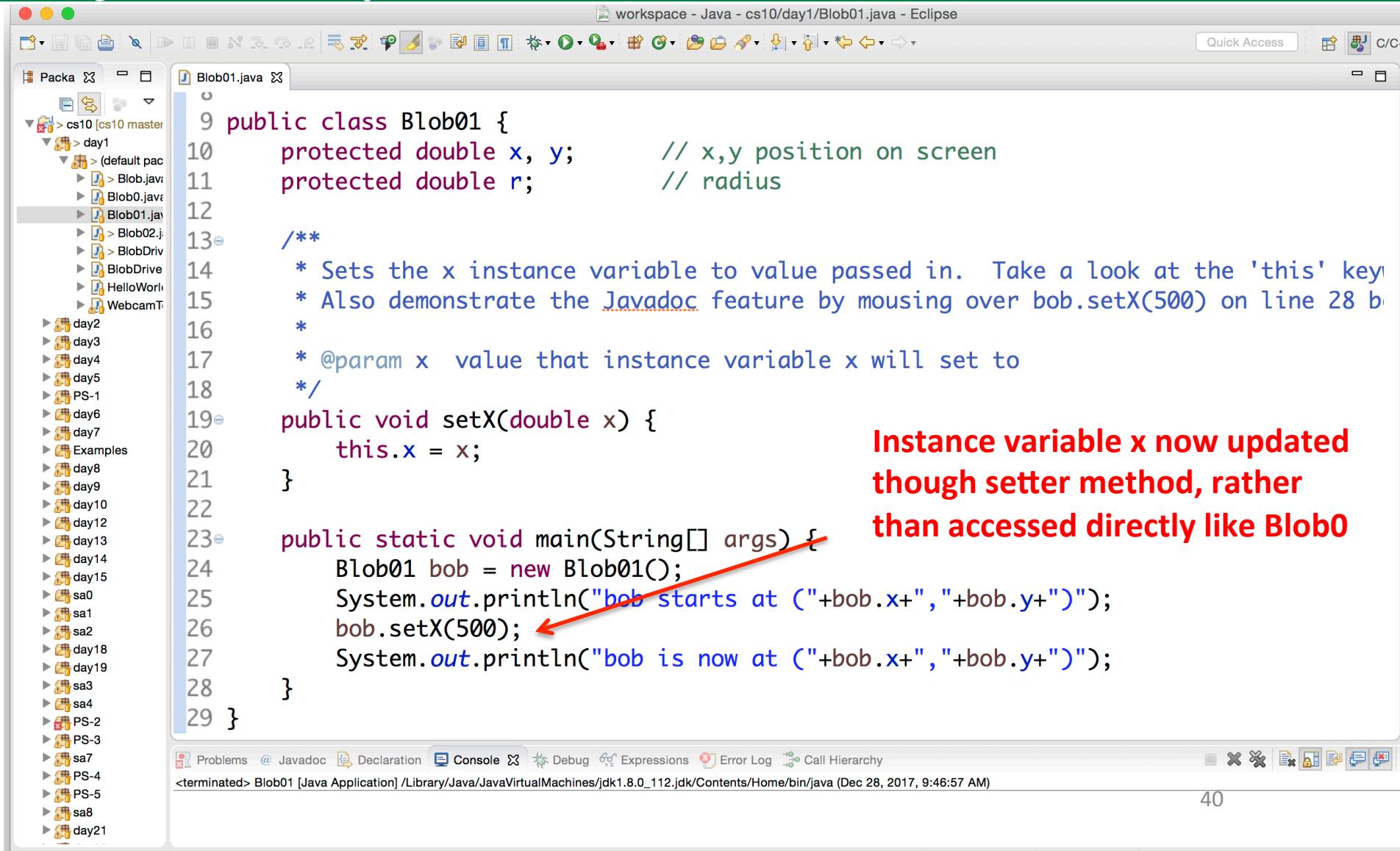
The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer (left):** Shows the project structure under "cs10 [cs10 master]". The "day1" folder contains files: Blob.java, Blob0.java, Blob01.java, Blob02.java, BlobDrive, BlobDrive, HelloWorld, and WebcamT.
- Code Editor (center):** Displays the content of `Blob01.java`. The code defines a class `Blob01` with protected instance variables `x`, `y`, and `r`. It includes a Javadoc comment for the `setX` method and a sample `main` method demonstrating its use.
- Annotations (right):** A red arrow points from the text "this.x = x;" in the `setX` method to a bulleted list explaining the meaning of `this` in Java.
- Bottom Status Bar:** Shows tabs for Problems, Javadoc, Declaration, Console, Debug, Expressions, Error Log, and Call Hierarchy. The status bar also indicates the application is terminated and provides the path to the Java VM.

**Annotations and Explanations:**

- Red Arrow:** Points to the line `this.x = x;` in the `setX` method.
- Listed Items:**
  - “`this.x` means the “instance variable `x` for this object”
  - `x` refers to the parameter
  - “`this` is like “self” in Python”
- Text at Bottom:** “Getter” methods return the value of instance variables (see `Blob.java`)

# Blob01: Add a “setter” method to allow the object to update its own instance variables



The screenshot shows the Eclipse IDE interface with the following details:

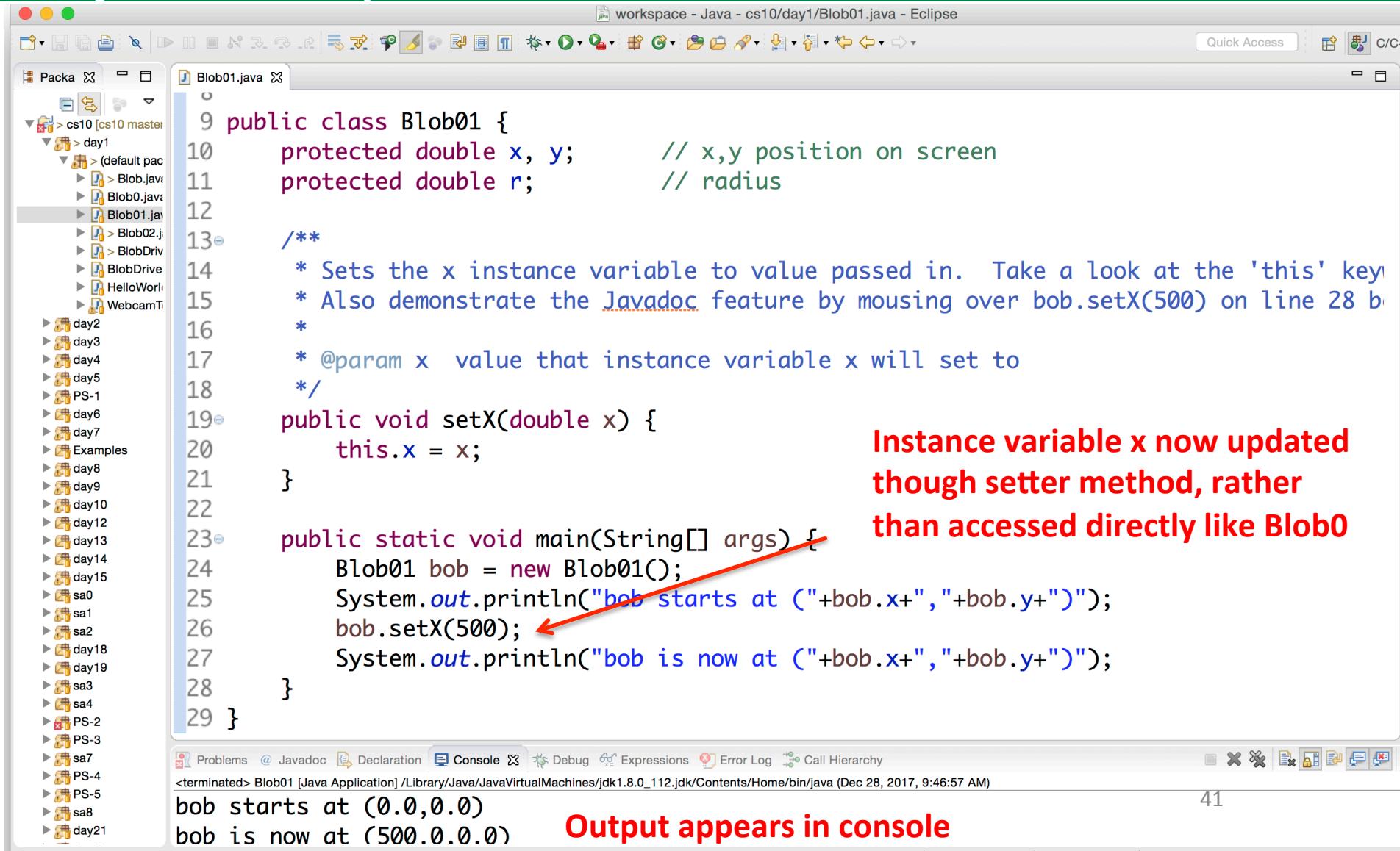
- Title Bar:** workspace - Java - cs10/day1/Blob01.java - Eclipse
- Toolbar:** Standard Eclipse toolbar with various icons.
- Left Sidebar:** Project Explorer showing the file structure under 'cs10 [cs10 master]'. The 'day1' folder contains files: Blob.java, Blob0.java, Blob01.java (selected), Blob02.java, BlobDrive, BlobDrive, HelloWorld, and WebcamT.
- Central Area:** Editor tab for 'Blob01.java' containing the following code:

```
public class Blob01 {  
    protected double x, y;          // x,y position on screen  
    protected double r;            // radius  
  
    /**  
     * Sets the x instance variable to value passed in. Take a look at the 'this' keyword.  
     * Also demonstrate the Javadoc feature by mousing over bob.setX(500) on line 28 below.  
     *  
     * @param x  value that instance variable x will set to  
     */  
    public void setX(double x) {  
        this.x = x;  
    }  
  
    public static void main(String[] args) {  
        Blob01 bob = new Blob01();  
        System.out.println("bob starts at ("+bob.x+","+bob.y+")");  
        bob.setX(500); ←  
        System.out.println("bob is now at ("+bob.x+","+bob.y+")");  
    }  
}
```
- Bottom Status Bar:** Shows tabs for Problems, Javadoc, Declaration, Console, Debug, Expressions, Error Log, and Call Hierarchy. The 'Console' tab is active.
- Bottom Footer:** Includes links for Writable, Smart Insert, and a page number indicator (1 : 1).

**Red Annotation:** A red arrow points to the line of code `bob.setX(500);` in the main() method, highlighting the use of the setter method to update the instance variable.

**Text Overlay:** Red text on the right side of the code area states: "Instance variable x now updated though setter method, rather than accessed directly like Blob0".

# Blob01: Add a “setter” method to allow the object to update its own instance variables



The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** workspace - Java - cs10/day1/Blob01.java - Eclipse
- Toolbar:** Standard Eclipse toolbar.
- Left Sidebar (Project Explorer):** Shows the project structure under "cs10 [cs10 master]". The "day1" folder contains files: Blob.java, Blob0.java, Blob01.java (selected), Blob02.java, BlobDrive, BlobDrive, HelloWorld, and WebcamT.
- Central Area (Code Editor):** Displays the content of Blob01.java. The code defines a class Blob01 with protected instance variables x, y, and r. It includes a Javadoc comment for the setX method and a main method that creates a Blob01 object, prints its initial position, calls setX(500), and prints its new position.
- Bottom Status Bar:** Shows tabs for Problems, Javadoc, Declaration, Console, Debug, Expressions, Error Log, and Call Hierarchy. The status bar also indicates the application is terminated and provides the date and time (Dec 28, 2017, 9:46:57 AM).
- Bottom Console:** Displays the output of the program: "bob starts at (0.0,0.0)" and "bob is now at (500.0,0.0)".
- Annotations:** A red arrow points from the text "Instance variable x now updated though setter method, rather than accessed directly like Blob0" to the line "bob.setX(500);".
- Text Overlay:** The text "Instance variable x now updated though setter method, rather than accessed directly like Blob0" is displayed in red font on the right side of the code editor.
- Page Number:** 41
- Page Footer:** Output appears in console

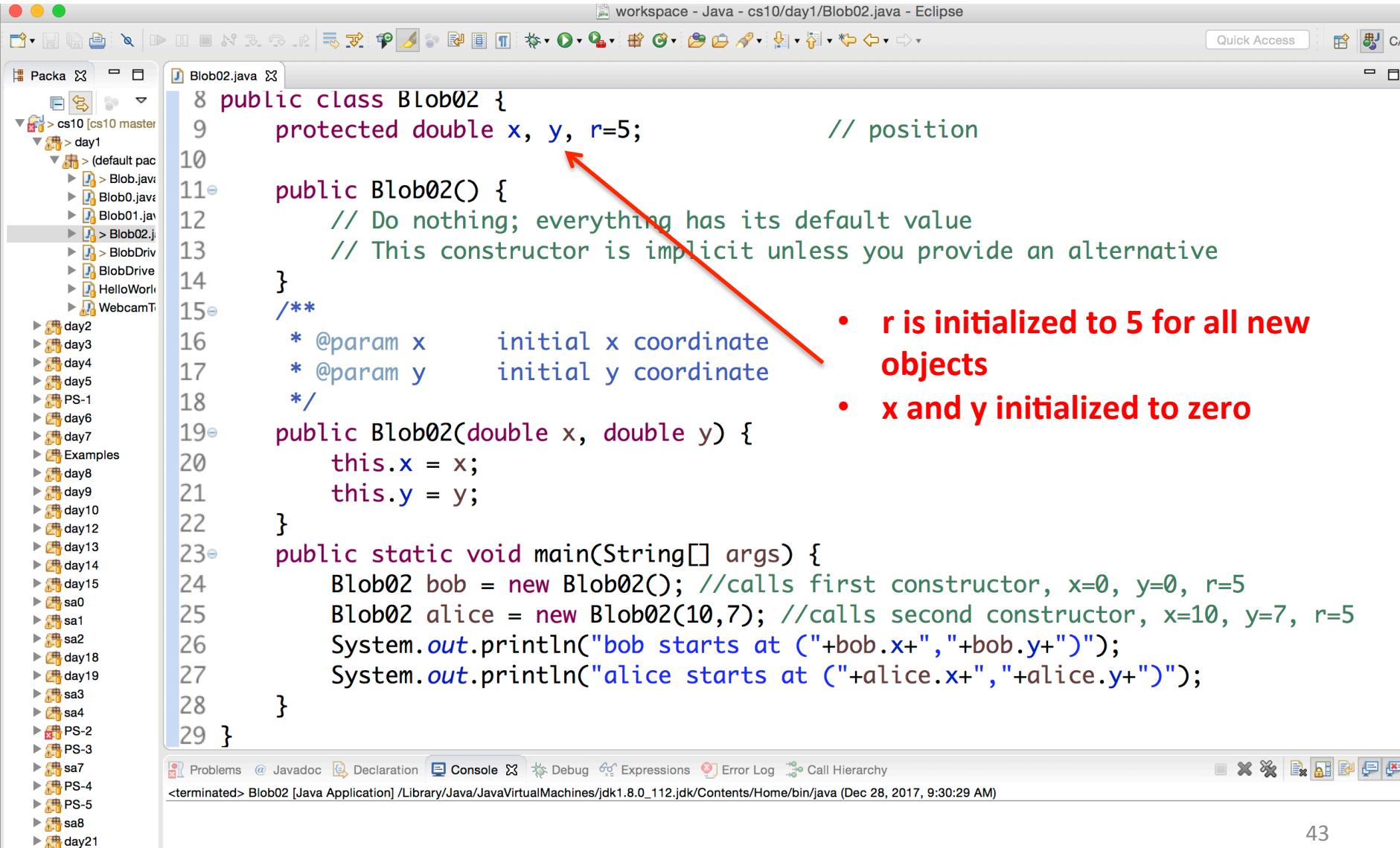
# Blob02.java

The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** workspace - Java - cs10/day1/Blob02.java - Eclipse
- Toolbar:** Standard Eclipse toolbar with various icons for file operations, search, and navigation.
- Left Sidebar (Project Explorer):** Shows the project structure under "cs10 [cs10 master]". The "day1" folder contains several Java files: Blob.java, Blob01.java, Blob02.java (selected), BlobDriver.java, HelloWorld.java, and WebcamTest.java. It also lists "day2" through "day21" and "Examples".
- Central Area (Code Editor):** Displays the content of Blob02.java. The code defines a class Blob02 with protected attributes x and y, and a constructor that initializes r to 5. It includes two constructors: one taking no arguments and one taking x and y coordinates. The main method creates instances of Blob02 for "bob" and "alice" and prints their initial positions.
- Bottom Bar:** Standard Eclipse bottom bar with tabs for Problems, Javadoc, Declaration, Console (selected), Debug, Expressions, Error Log, and Call Hierarchy. The console tab shows the output: "terminated> Blob02 [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0\_112.jdk/Contents/Home/bin/java (Dec 28, 2017, 9:30:29 AM)".
- Right Sidebar:** Standard Eclipse right sidebar with icons for quick access, C/C++, and other tools.

```
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 }
```

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



The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer (left):** Shows the project structure under "cs10 [cs10 master]".
- Code Editor (center):** Displays the `Blob02.java` file content.
- Console (bottom):** Shows the output of the application's execution.

**Code Content:**

```
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 }
```

A red arrow points from the text "r is initialized to 5 for all new objects" to the line `r=5` in the code editor.

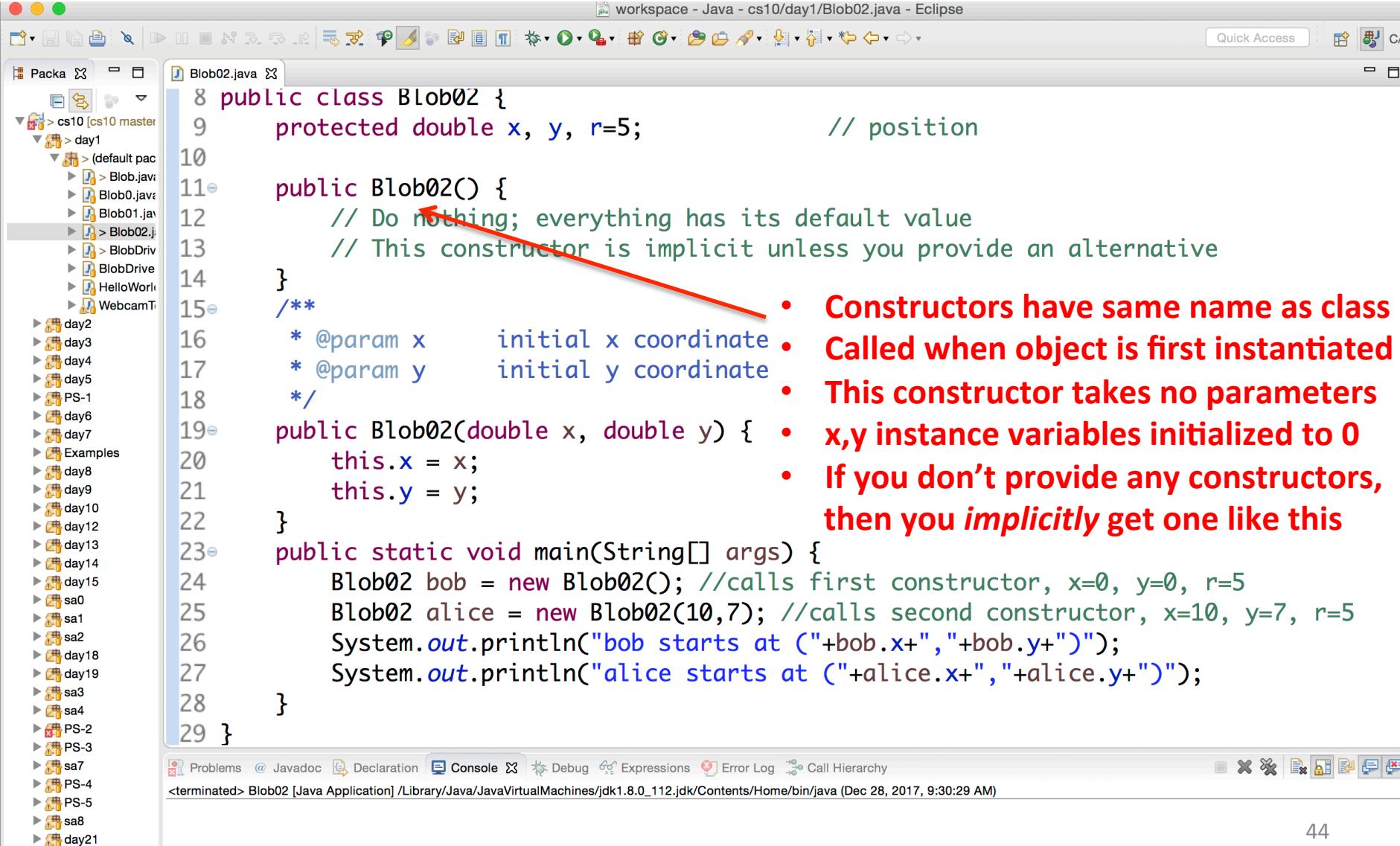
**Annotations (red text):**

- **r is initialized to 5 for all new objects**
- **x and y initialized to zero**

**Console Output:**

```
<terminated> Blob02 [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Dec 28, 2017, 9:30:29 AM)
```

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

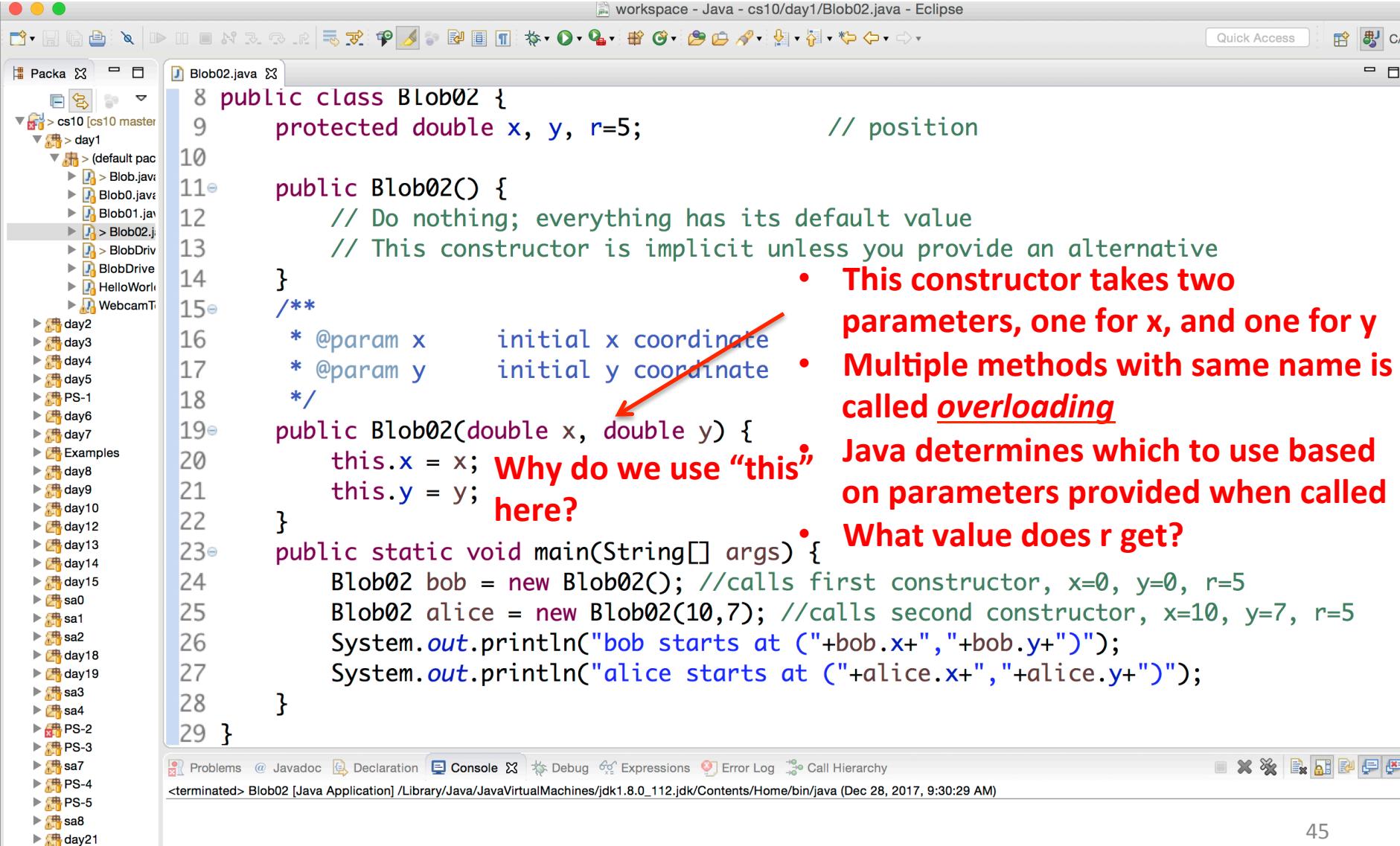


The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer (left):** Shows the workspace structure with projects like cs10, day1, and various sub-folders and files.
- Code Editor (center):** Displays the Java code for `Blob02.java`.

```
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 }
```
- Console (bottom):** Shows the output of the program execution.
- Annotations (right side):** A red arrow points from the explanatory text below to the implicit constructor definition in the code editor.
- List of bullet points (right side):**
  - 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

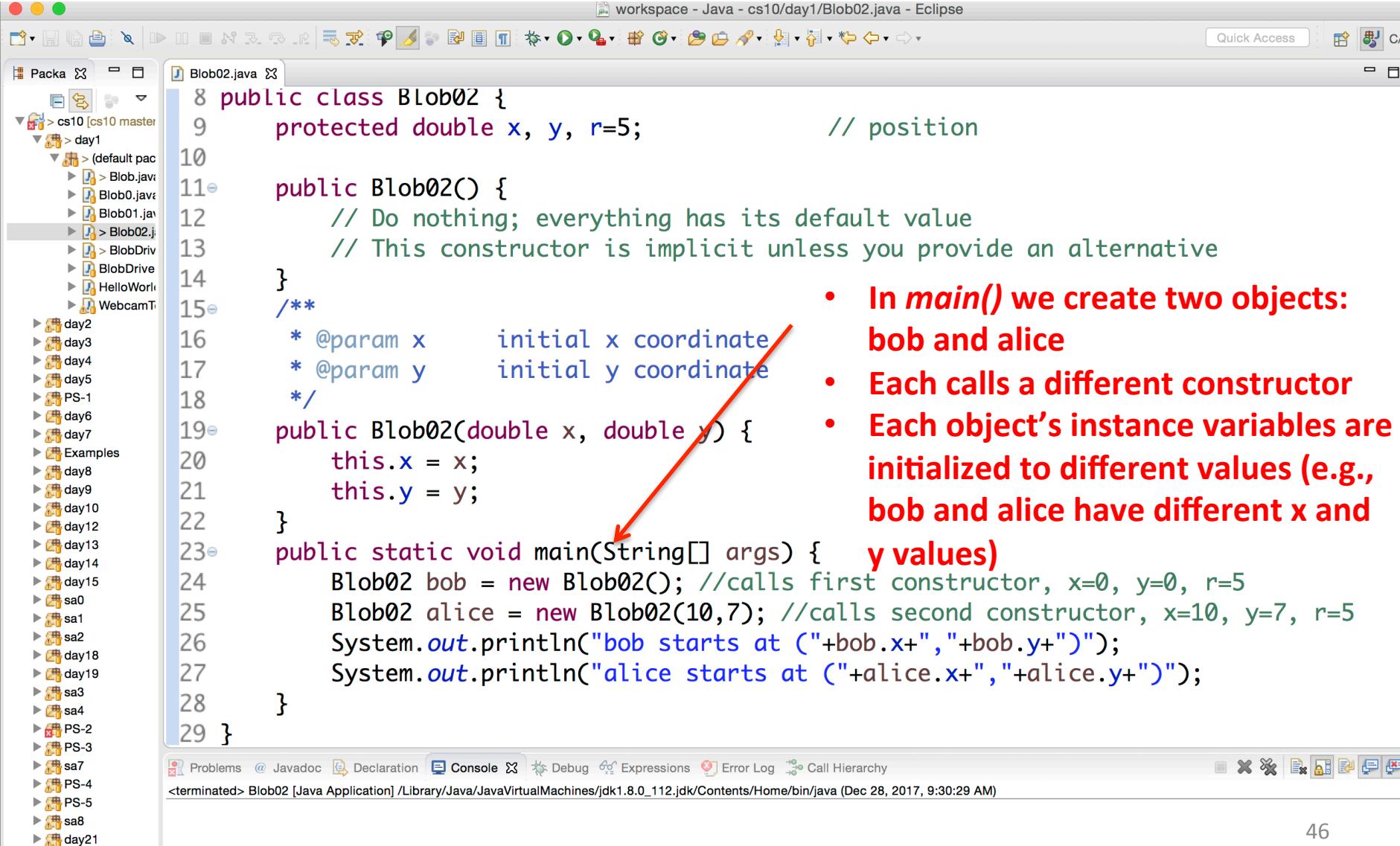
# Blob02: Constructors called when an object is first instantiated (run before other code)



The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer (left):** Shows the project structure under "cs10 [cs10 master]". The "day1" folder contains files: Blob.java, Blob01.java, Blob02.java, BlobDrive, HelloWorld, and WebcamT.
- Code Editor (center):** Displays the content of `Blob02.java`. The code defines a class `Blob02` with protected instance variables `x`, `y`, and `r=5`. It includes two constructors: an implicit constructor (labeled as implicit) and a parameterized constructor that takes `x` and `y`. The parameterized constructor uses `this.x = x;` and `this.y = y;`. A red arrow points from the question "Why do we use "this" here?" to the `this` keyword in the constructor. The code also includes a `main` method that creates instances of `Blob02` and prints their initial coordinates.
- Console (bottom):** Shows the output of the `main` method execution.
- Annotations (right side of the code):** Red annotations explain the code:
  - This constructor takes two parameters, one for x, and one for y**
  - Multiple methods with same name is called overloading**
  - Java determines which to use based on parameters provided when called**
  - What value does r get?**

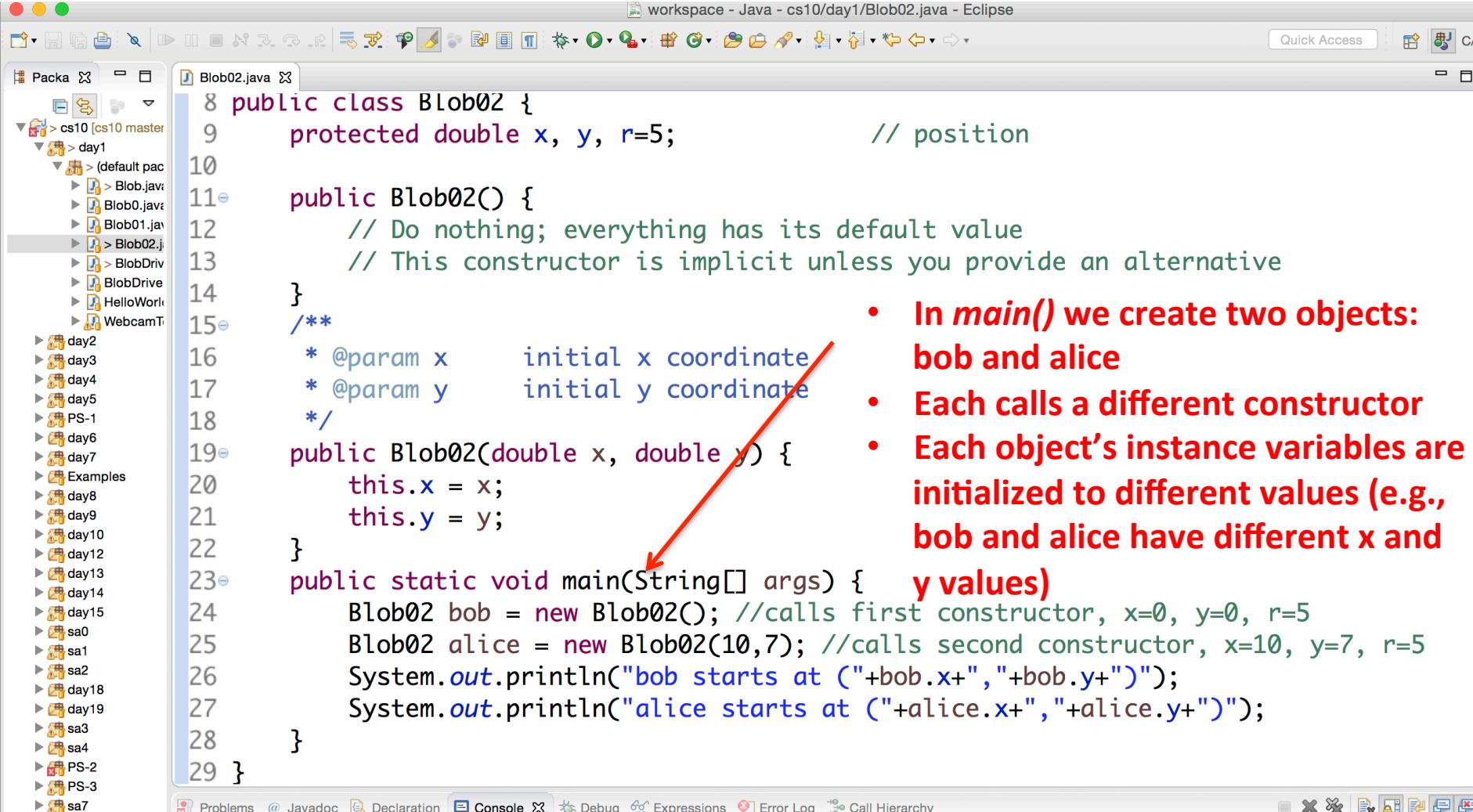
# Blob02: Constructors called when an object is first instantiated (run before other code)



The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer (left):** Shows the project structure under "cs10 [cs10 master]". The "day1" folder contains files: Blob.java, Blob01.java, Blob02.java, BlobDrive, HelloWorld, and WebcamT.
- Code Editor (center):** Displays the content of `Blob02.java`. The code defines a class `Blob02` with instance variables `x`, `y`, and `r`. It includes two constructors: an implicit constructor (empty parentheses) and a parameterized constructor taking `x` and `y`. The `main` method creates two objects, `bob` and `alice`, using these constructors and prints their initial coordinates.
- Console (bottom):** Shows the output of the `main` method execution.
- Right side (red annotations):** A red arrow points from the text "In main() we create two objects: bob and alice" to the `main` method in the code editor.
- List of bullet points (right side):**
  - 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)



The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows a project named "cs10" with a "master" branch. Inside "day1", there are files: Blob.java, Blob01.java, Blob02.java (selected), BlobDrive.java, HelloWorld.java, and WebcamT.java.
- Code Editor:** Displays the content of `Blob02.java`. The code defines a class `Blob02` with instance variables `x`, `y`, and `r`. It contains two constructors: an implicit constructor (empty parentheses) and a parameterized constructor taking `x` and `y`. The main method creates two objects: `bob` using the implicit constructor and `alice` using the parameterized constructor with parameters `10` and `7`.
- Console:** At the bottom, the console output is shown:  
`bob starts at (0.0,0.0)`  
`alice starts at (10.0,7.0)`
- Annotations:** A red arrow points from the text "In `main()` we create two objects: bob and alice" to the line `public static void main(String[] args) {` in the code editor.
- Text on the right:** A list of bullet points explaining the behavior:
  - 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)
- Page Number:** 47

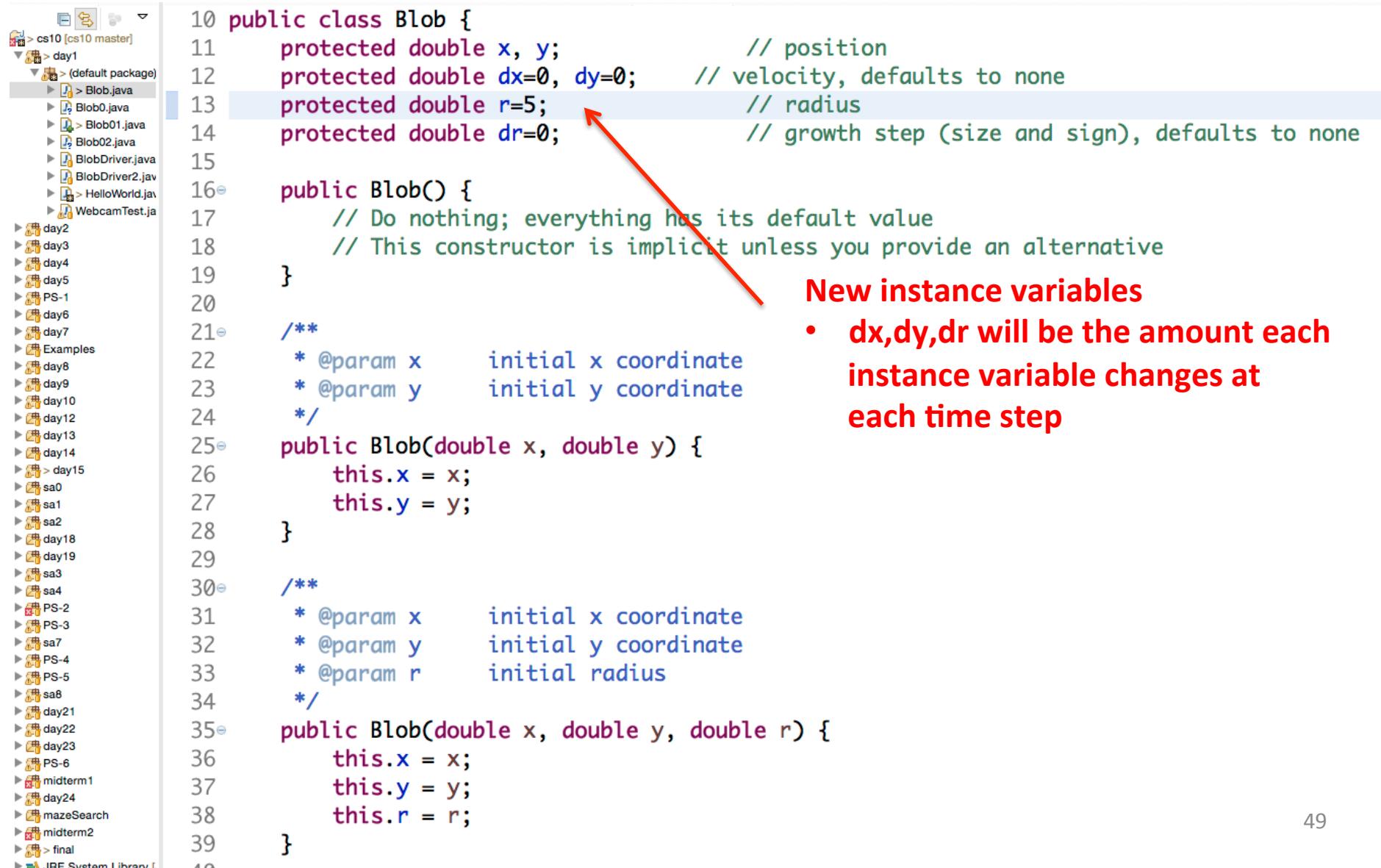
Output appears in console

# 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     }  
40 }
```

The code editor interface shows a sidebar with project navigation and a list of files. The current file, Blob.java, is highlighted. The code itself defines a class Blob with several constructors and protected member variables.

# 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     }  
40 }
```

## New instance variables

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

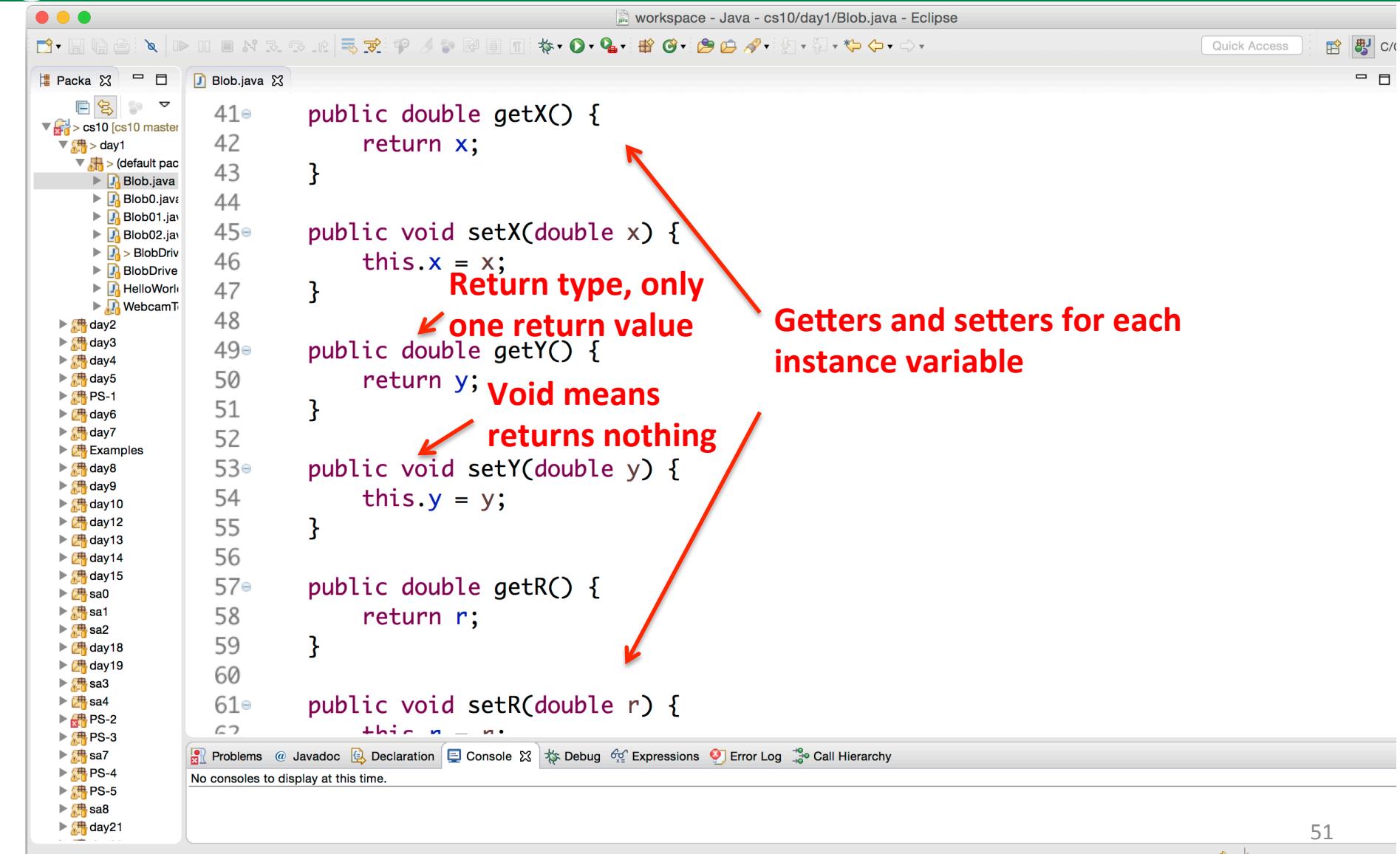
# Blob: Small changes to previous versions give a more full featured Blob

```
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     }  
40 }
```

## 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



The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer (left):** Shows the workspace structure with a project named "cs10" containing a package "day1" which contains a file "Blob.java". Other files like "Blob01.java", "Blob02.java", "BlobDrive.java", etc., are also listed.
- Code Editor (center):** Displays the content of the "Blob.java" file. The code defines a class with four instance variables: x, y, and r (all of type double), and a constructor. It includes four pairs of getters and setters: getX/setX, getY/setY, getR/setR.
- Annotations (red text and arrows):**
  - An annotation above the first set of getters and setters states: "Return type, only one return value".
  - An annotation below the second set of getters and setters states: "Void means returns nothing".
  - An annotation to the right of the code states: "Getters and setters for each instance variable".
- Bottom Bar:** Shows tabs for Problems, Javadoc, Declaration, Console, Debug, Expressions, Error Log, and Call Hierarchy. The "Console" tab is selected, displaying the message "No consoles to display at this time."

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

The screenshot shows the Eclipse IDE interface with the following details:

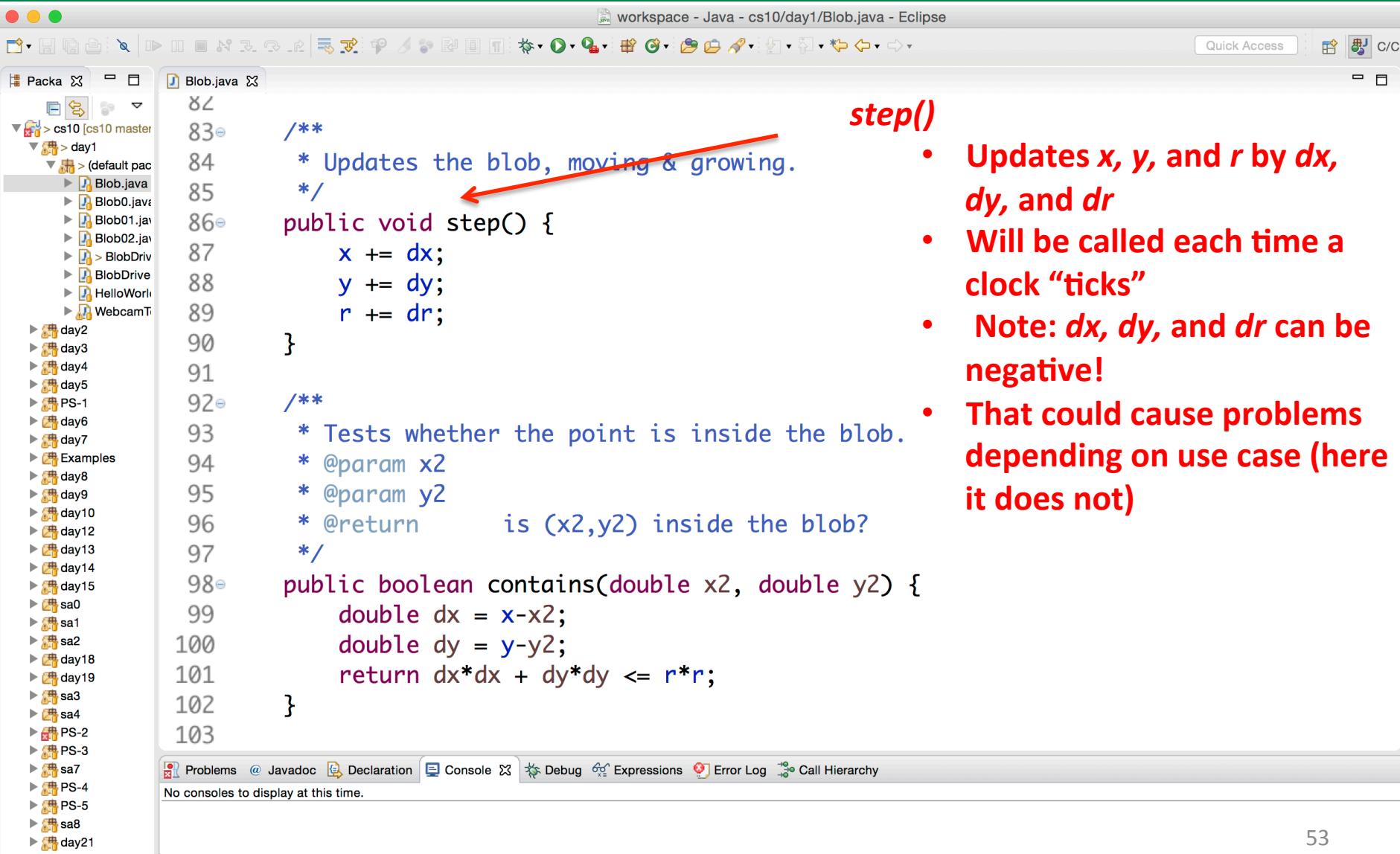
- Toolbar:** Standard Eclipse toolbar with icons for file operations, search, and navigation.
- ActionBar:** Shows "workspace - Java - cs10/day1/Blob.java - Eclipse".
- Package Explorer:** Left sidebar showing the project structure. The "day1" folder contains several Java files: Blob.java, Blob01.java, Blob02.java, BlobDriver.java, BlobDriver2.java, HelloWorld1.java, and WebcamTest.java. Other folders like "day2" through "day16" and "sa0" through "sa4" are also listed.
- Code Editor:** Right pane displaying the content of `Blob.java`. The code defines a class with methods for setting position, radius, velocity, and growth direction.

```
49     }
50
51     public void setY(double y) {
52         this.y = y;
53     }
54
55     public double getR() {
56         return r;
57     }
58
59     public void setR(double r) {
60         this.r = r;
61     }
62
63     /**
64      * Sets the velocity.
65      * @param dxnew dx
66      * @param dynew dy
67      */
68     public void setVelocity(double dx, double dy) {
69         this.dx = dx;
70         this.dy = dy;
71     }
72
73     /**
74      * Sets the direction of growth.
75      * @param drnew dr
76      */
77     public void setGrowth(double dr) {
78         this.dr = dr;
79     }
80 }
```

Annotations and arrows highlight specific parts of the code:

- A red arrow points to the `setVelocity` method signature with the text ***setVelocity()* sets dx and dy**.
- A red arrow points to the `setGrowth` method signature with the text ***setGrowth()* sets dr**.

# Blob: New methods to control how the blob behaves



The screenshot shows the Eclipse IDE interface with the following details:

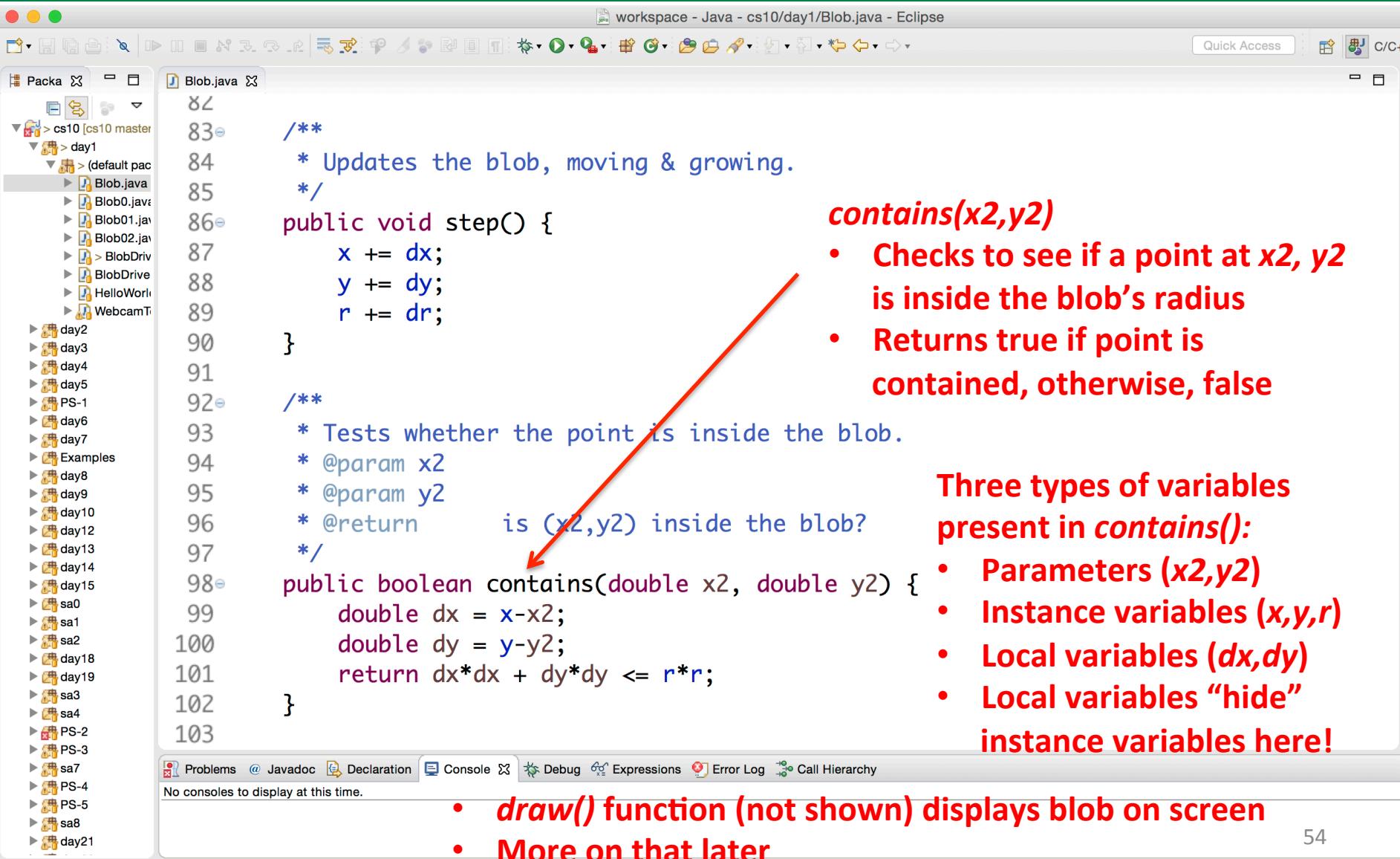
- Title Bar:** workspace - Java - cs10/day1/Blob.java - Eclipse
- Toolbar:** Standard Eclipse toolbar with various icons.
- Left Sidebar:** Project Explorer showing the file structure under "cs10 [cs10 master]".
- Central Area:** Blob.java code editor with the following content:

```
82
83  /**
84   * Updates the blob, moving & growing.
85   */
86  public void step() {
87      x += dx;
88      y += dy;
89      r += dr;
90  }
91
92  /**
93   * Tests whether the point is inside the blob.
94   * @param x2
95   * @param y2
96   * @return      is (x2,y2) inside the blob?
97   */
98  public boolean contains(double x2, double y2) {
99      double dx = x-x2;
100     double dy = y-y2;
101     return dx*dx + dy*dy <= r*r;
102 }
```
- Annotations:** A red arrow points from the word "step()" in the list below to the "step()" method definition in the code editor.
- Right Side:** A red box highlights the word "step()", and a list of bullet points describes its behavior.
- Bottom:** Eclipse status bar showing tabs for Problems, Javadoc, Declaration, Console, Debug, Expressions, Error Log, and Call Hierarchy. It also says "No consoles to display at this time."

## 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



The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer (left):** Shows the project structure under "cs10 [cs10 master]". The "day1" folder contains "Blob.java", "Blob0.java", "Blob01.java", "Blob02.java", "BlobDrive", "HelloWorld", and "WebcamT". Other folders like "day2" through "day21" and "PS-1" through "PS-5" are also listed.
- Code Editor (right):** Displays the content of `Blob.java`. The code includes a `step()` method and a `contains(double x2, double y2)` method. A red arrow points from the explanatory text below to the parameter `x2` in the `contains` method signature.
- Bottom Bar:** Shows tabs for Problems, Javadoc, Declaration, Console, Debug, Expressions, Error Log, and Call Hierarchy. The Console tab indicates "No consoles to display at this time."

**contains( $x2, y2$ )**

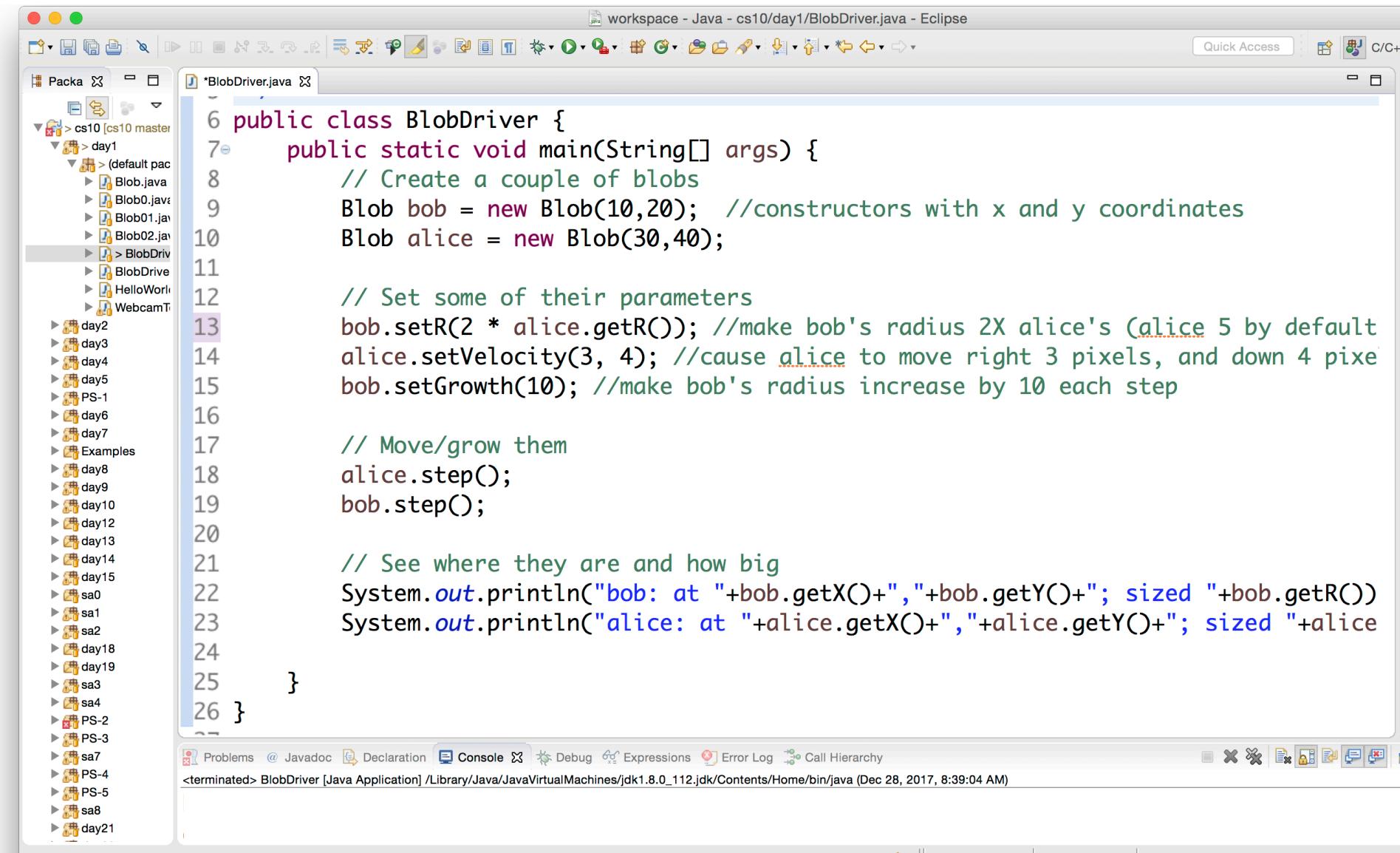
- Checks to see if a point at  $x2, y2$  is inside the blob's radius
- Returns true if point is contained, otherwise, false

**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



The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** workspace - Java - cs10/day1/BlobDriver.java - Eclipse
- Toolbar:** Standard Eclipse toolbar with various icons for file operations, search, and navigation.
- Left Sidebar (Package Explorer):** Shows the project structure under "cs10 [cs10 master]". The "day1" package is expanded, showing files: Blob.java, Blob0.java, Blob01.java, Blob2.java, BlobDriver.java (selected), BlobDrive.java, HelloWorld.java, and WebcamT.java. Below "day1" are other packages like day2 through day21, Examples, PS-1, PS-2, PS-3, PS-4, PS-5, PS-6, PS-7, PS-8, PS-9, PS-10, PS-11, PS-12, PS-13, PS-14, PS-15, sa0, sa1, sa2, sa3, sa4, and PS-21.
- Central Editor Area:** Displays the code for `BlobDriver.java`. The code creates two `Blob` objects, `bob` and `alice`, with initial coordinates (10, 20) and (30, 40). It then sets their parameters: `bob.setR(2 * alice.getR())`, `alice.setVelocity(3, 4)`, and `bob.setGrowth(10)`. Finally, it moves them by calling `step()` on both blobs and prints their current positions and sizes using `System.out.println`.
- Bottom Status Bar:** Shows the status "terminated> BlobDriver [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0\_112.jdk/Contents/Home/bin/java (Dec 28, 2017, 8:39:04 AM)".

```
public class BlobDriver {
    public static void main(String[] args) {
        // Create a couple of blobs
        Blob bob = new Blob(10,20); //constructors with x and y coordinates
        Blob alice = new Blob(30,40);

        // Set some of their parameters
        bob.setR(2 * alice.getR()); //make bob's radius 2X alice's (alice 5 by default
        alice.setVelocity(3, 4); //cause alice to move right 3 pixels, and down 4 pixels
        bob.setGrowth(10); //make bob's radius increase by 10 each step

        // Move/grow them
        alice.step();
        bob.step();

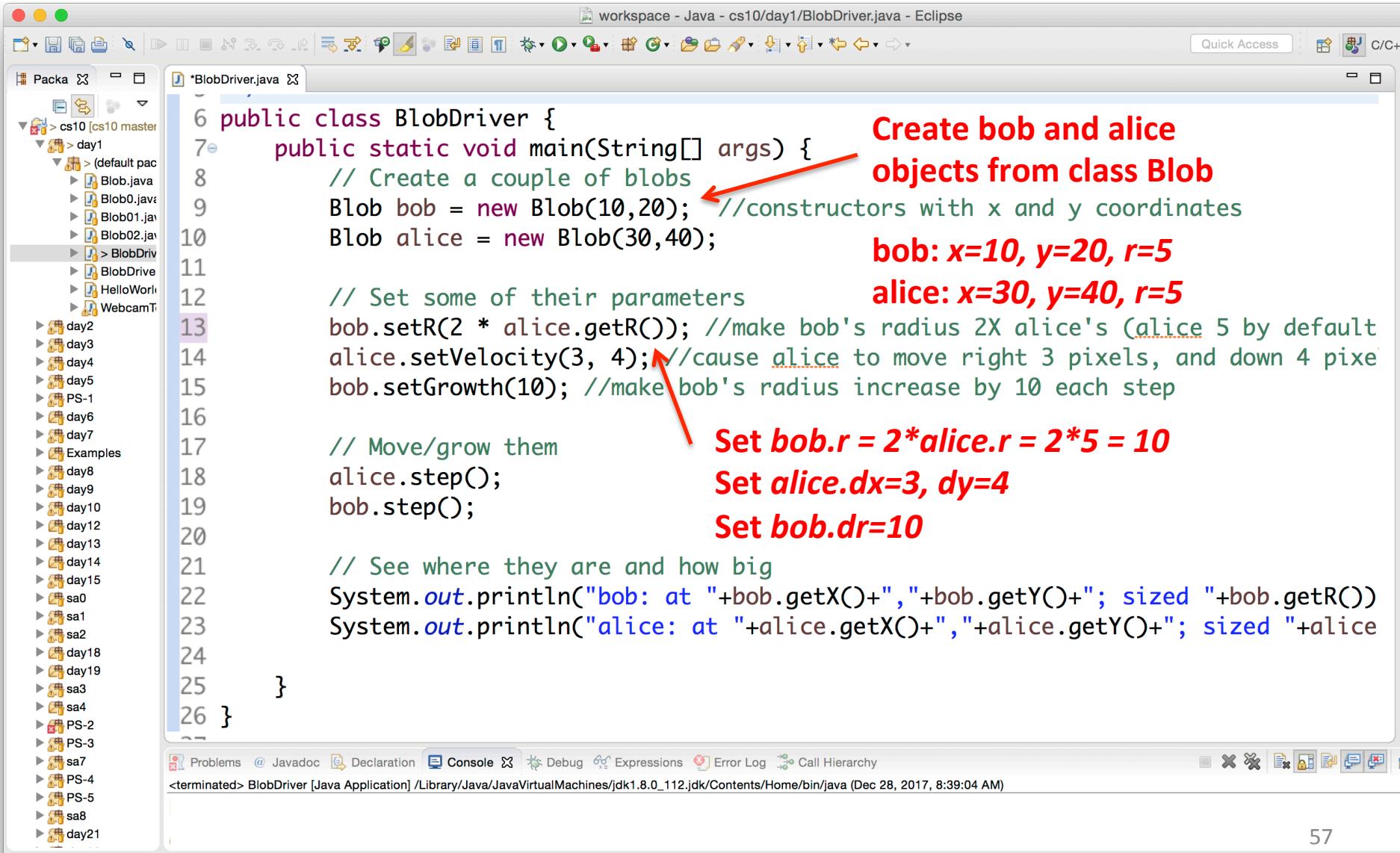
        // See where they are and how big
        System.out.println("bob: at "+bob.getX()+","+bob.getY()+" sized "+bob.getR())
        System.out.println("alice: at "+alice.getX()+","+alice.getY()+" sized "+alice.getR())
    }
}
```

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

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer (left):** Shows a project named "cs10" with a "master" branch. Inside "day1", there are files: Blob.java, Blob0.java, Blob01.java, Blob02.java, BlobDriver.java, BlobDrive.java, HelloWorld.java, and WebcamTest.java.
- Code Editor (center):** Displays the content of `BlobDriver.java`. The code defines a `BlobDriver` class with a `main` method. It creates two `Blob` objects, `bob` and `alice`, with specific coordinates. It then sets parameters for these blobs and moves them by calling `step()` twice. Finally, it prints their current positions and sizes.
- Annotations (red text):**
  - An annotation points to the `BlobDriver` class definition with the text: **BlobDriver class in file BlobDriver.java**.
  - An annotation points to the `alice` variable when it is first defined with the text: **Uses class Blob defined in file Blob.java**.
  - An annotation points to the closing brace of the `main` method with the text: **Works if classes defined in same project (cs10 here)**.
  - An annotation points to the `Blob` class reference in the `main` method with the text: **Blob class is not defined in the “driver” or “application” class BlobDriver**.
- Bottom Status Bar:** Shows the terminal output: `<terminated> BlobDriver [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Dec 28, 2017, 8:39:04 AM)`.

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



The screenshot shows the Eclipse IDE interface with the file `BlobDriver.java` open in the editor. The code implements a `BlobDriver` class with a `main` method. It creates two `Blob` objects, `bob` and `alice`, with initial coordinates (10, 20) and (30, 40) respectively. It then sets parameters like radius, velocity, and growth rate. Finally, it moves the blobs and prints their current positions and sizes.

```
public class BlobDriver {
    public static void main(String[] args) {
        // Create a couple of blobs
        Blob bob = new Blob(10,20); //constructors with x and y coordinates
        Blob alice = new Blob(30,40);

        // Set some of their parameters
        bob.setR(2 * alice.getR()); //make bob's radius 2X alice's (alice 5 by default
        alice.setVelocity(3, 4); //cause alice to move right 3 pixels, and down 4 pixels
        bob.setGrowth(10); //make bob's radius increase by 10 each step

        // Move/grow them
        alice.step();
        bob.step();

        // See where they are and how big
        System.out.println("bob: at "+bob.getX()+","+bob.getY()+" sized "+bob.getR());
        System.out.println("alice: at "+alice.getX()+","+alice.getY()+" sized "+alice.getR());
    }
}
```

**Create bob and alice objects from class Blob**

**bob:  $x=10, y=20, r=5$**

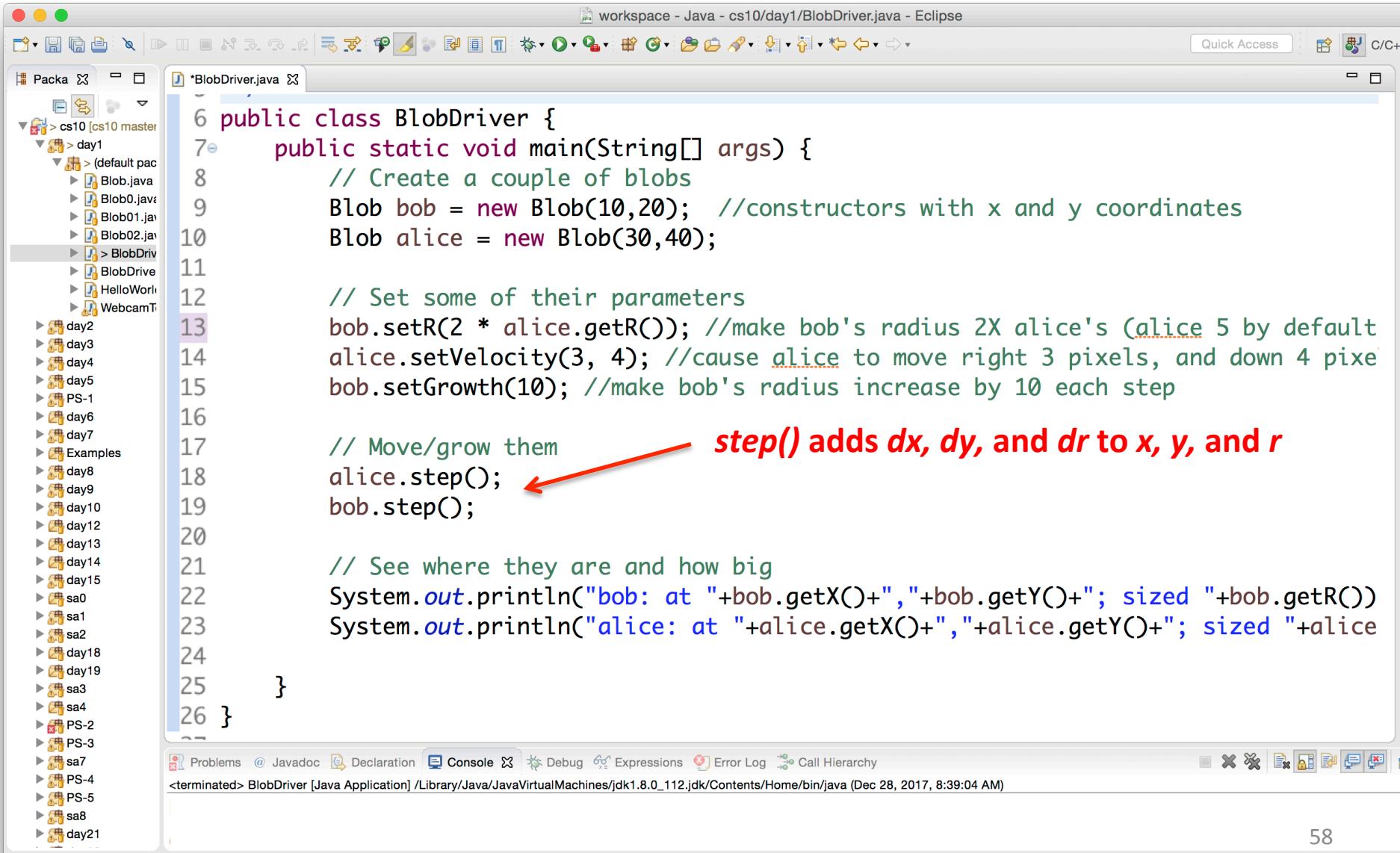
**alice:  $x=30, y=40, r=5$**

**Set  $bob.r = 2 * alice.r = 2 * 5 = 10$**

**Set  $alice.dx=3, dy=4$**

**Set  $bob.dr=10$**

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



The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** workspace - Java - cs10/day1/BlobDriver.java - Eclipse
- Toolbar:** Standard Eclipse toolbar with various icons for file operations, search, and navigation.
- Left Sidebar (Project Explorer):** Shows the project structure under "cs10 [cs10 master]". The "day1" folder contains several Java files: Blob.java, Blob0.java, Blob01.java, Blob2.java, BlobDriver.java (selected), BlobDrive.java, HelloWorld.java, and WebcamT.java. Below "day1" are other folders like "day2" through "day21" and "PS-1" through "PS-5".
- Central Editor Area:** Displays the content of `BlobDriver.java`. The code creates two `Blob` objects, `bob` and `alice`, with initial coordinates (10, 20) and (30, 40). It then sets some parameters: `bob.setR(2 * alice.getR())`, `alice.setVelocity(3, 4)`, and `bob.setGrowth(10)`. The code then moves the blobs with `alice.step()` and `bob.step()`. Finally, it prints their current positions and sizes to the console.
- Annotations:** A red arrow points from the text ***step() adds dx, dy, and dr to x, y, and r*** to the `alice.step()` line in the code.
- Bottom Status Bar:** Shows the status `<terminated> BlobDriver [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Dec 28, 2017, 8:39:04 AM)`.

```
public class BlobDriver {  
    public static void main(String[] args) {  
        // Create a couple of blobs  
        Blob bob = new Blob(10,20); //constructors with x and y coordinates  
        Blob alice = new Blob(30,40);  
  
        // Set some of their parameters  
        bob.setR(2 * alice.getR()); //make bob's radius 2X alice's (alice 5 by default  
        alice.setVelocity(3, 4); //cause alice to move right 3 pixels, and down 4 pixels  
        bob.setGrowth(10); //make bob's radius increase by 10 each step  
  
        // Move/grow them  
        alice.step(); // step() adds dx, dy, and dr to x, y, and r  
        bob.step();  
  
        // See where they are and how big  
        System.out.println("bob: at "+bob.getX()+","+bob.getY()+" sized "+bob.getR())  
        System.out.println("alice: at "+alice.getX()+","+alice.getY()+" sized "+alice.getR())  
    }  
}
```

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

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer (left):** Shows a project named "cs10" with a "day1" folder containing several Java files: Blob.java, Blob0.java, Blob01.java, Blob02.java, BlobDriver.java, and BlobDrive.java.
- Code Editor (center):** Displays the content of `BlobDriver.java`. The code creates two `Blob` objects, `bob` and `alice`, with initial coordinates (10, 20) and (30, 40) respectively. It then sets their parameters: `bob`'s radius is doubled, `alice`'s velocity is set to (3, 4), and `bob`'s growth rate is set to 10. Finally, it calls `step()` on both blobs to move them. Afterward, it prints their current locations and sizes to the console.
- Annotations (red text and arrows):**
  - An annotation **step() adds dx, dy, and dr to x, y, and r** points to the first call to `step()` at line 18.
  - An annotation **Print locations to console** points to the `System.out.println` statements at lines 22 and 23.
  - An annotation **Output** points to the terminal output at the bottom.
- Terminal Output (bottom):** Shows the console output:

```
bob: at 10.0,20.0; sized 20.0
alice: at 33.0,44.0; sized 5.0
```
- Page Number (bottom right):** 59

# BlobDriver2.java

The screenshot shows a Java development environment with the following details:

- Packages:** The left sidebar lists several packages and files, including cs10 [cs10 master], day1, (default package), 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, day22, day23, PS-6, midterm1, day24, mazeSearch, midterm2, and final.
- Editors:** The main editor window displays the code for `BlobDriver2.java`. The code defines a class `BlobDriver2` with a constructor that takes two `Blob` objects and initializes them. It contains a test method `test1()` that creates two blobs, sets their initial positions and velocities, and then prints their current positions and sizes. It also includes a `main` method to demonstrate the driver's functionality.
- Toolbars and Status Bar:** The bottom of the interface features standard Java IDE toolbars and a status bar indicating the application is terminated.

```
6 public class BlobDriver2 {  
7     private Blob blob1, blob2;  
8  
9     public BlobDriver2(Blob blob1, Blob blob2) {  
10         this.blob1 = blob1;  
11         this.blob2 = blob2;  
12     }  
13  
14     /**  
15      * One test case  
16     */  
17     public void test1() {  
18         blob1.setR(2 * blob2.getR());  
19         blob2.setVelocity(3, 4);  
20         blob2.step();  
21         blob1.setGrowth(10);  
22         blob1.step();  
23         System.out.println("blob1: at "+blob1.getX()+" , "+blob1.getY()+" ; sized "+blob1.getR());  
24         System.out.println("blob2: at "+blob2.getX()+" , "+blob2.getY()+" ; sized "+blob2.getR());  
25     }  
26  
27     public static void main(String[] args) {  
28         // Create a couple of blobs  
29         Blob bob = new Blob(10,20);  
30         Blob alice = new Blob(30,40);  
31         // Create the test driver with them  
32         BlobDriver2 driver = new BlobDriver2(bob, alice);  
33         // Test  
34         driver.test1();  
35         // Could insert more test cases....  
36     }  
}
```

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

```
public class BlobDriver2 {
    private Blob blob1, blob2;

    public BlobDriver2(Blob blob1, Blob blob2) {
        this.blob1 = blob1;
        this.blob2 = blob2;
    }

    /**
     * One test case
     */
    public void test1() {
        blob1.setR(2 * blob2.getR());
        blob2.setVelocity(3, 4);
        blob2.step();
        blob1.setGrowth(10);
        blob1.step();
        System.out.println("blob1: at "+blob1.getX()+" , "+blob1.getY()+" ; sized "+blob1.getR());
        System.out.println("blob2: at "+blob2.getX()+" , "+blob2.getY()+" ; sized "+blob2.getR());
    }

    public static void main(String[] args) {
        // Create a couple of blobs
        Blob bob = new Blob(10,20);
        Blob alice = new Blob(30,40);
        // Create the test driver with them
        BlobDriver2 driver = new BlobDriver2(bob, alice);
        // Test
        driver.test1();
        // Could insert more test cases....
    }
}
```

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

```
public class BlobDriver2 {
    private Blob blob1, blob2;

    public BlobDriver2(Blob blob1, Blob blob2) {
        this.blob1 = blob1;
        this.blob2 = blob2;
    }

    /**
     * One test case
     */
    public void test1() {
        blob1.setR(2 * blob2.getR());
        blob2.setVelocity(3, 4);
        blob2.step();
        blob1.setGrowth(10);
        blob1.step();
        System.out.println("blob1: at " + blob1.getX() + ", " + blob1.getY() + "; sized " + blob1.getR());
        System.out.println("blob2: at " + blob2.getX() + ", " + blob2.getY() + "; sized " + blob2.getR());
    }

    public static void main(String[] args) {
        // Create a couple of blobs
        Blob bob = new Blob(10, 20);
        Blob alice = new Blob(30, 40);
        // Create the test driver with them
        BlobDriver2 driver = new BlobDriver2(bob, alice);
        // Test
        driver.test1();
        // Could insert more test cases....
    }
}
```

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

Output

# 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!)

