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? Other?

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


### ASSESSMENT

<table>
<thead>
<tr>
<th>5</th>
<th>Section (Recitation) meetings</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>Homework</td>
</tr>
<tr>
<td></td>
<td>• Short assignments (SA): 10%</td>
</tr>
<tr>
<td></td>
<td>• Problem sets (PS): 30%</td>
</tr>
<tr>
<td>55</td>
<td>Exams</td>
</tr>
<tr>
<td></td>
<td>• Two midterms, each</td>
</tr>
<tr>
<td></td>
<td>worth 15%</td>
</tr>
<tr>
<td></td>
<td>• One cumulative final</td>
</tr>
<tr>
<td></td>
<td>worth 25%</td>
</tr>
</tbody>
</table>

### Textbook:

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

**Canvas**
- Course announcements and homework submissions
- Section assignments

**Slack (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
  - We will use IntelliJ IDEA for this course
- Create your first Java class
- Read and acknowledge course policies and honor code
- Complete survey **before 8:00am tomorrow** (or risk getting assigned to inconvenient section time!)
- **X-hour this week**
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

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 syntactically 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 IntelliJ, 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”
   - IntelliJ “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

Right click on code and choose “Run <class name>.main()” button to run
1. Create “cs10” Project to hold source code (only need to do this one time)

Start IntelliJ, then select “Create new project” or click File->New->Project

1) Choose Java
2) Choose Java version
3) Take defaults
4) Click Next
1. Create “cs10” Project to hold source code (only need to do this one time)

Do not create project from template

1) Leave UNCHECKED

2) Click Next
1. Create “cs10” Project to hold source code (only need to do this one time)

Name project “cs10” and set directory on disk where code will be stored

1) Choose Project name (“cs10”)

2) Choose directory where code will be stored

3) Click Finish
2. Create Source folder to hold your source code for day one of class

Click File->New->Directory to create directory for related code (e.g., “day1” or “PS1”)

1) Click File->New->Directory

2) Give directory a name

3) Right click on new directory then select “Mark Directory as” and “Sources Root”

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

Right click on Source folder and select New->Java Class

1) Right click on Source folder (e.g. “day1”), then select New->Java Class

2) Give class a name (starting with capital letter)

3) IntelliJ creates file on disk (e.g., “HelloWorld.java”) and sets up your new class
IntelliJ creates `HelloWorld.java` “boilerplate” code

- File on disk is `HelloWorld.java`
- Class is named `HelloWorld`
We can flesh out the boilerplate code to print “Hello World!” to the console.

Execution begins at main() method. Type “main” then enter and IntelliJ expands to include the main method declaration.

In Java a print statement is `System.out.println(“text you want to print goes here”);`. Type “sout” then enter to have IntelliJ fill out print statement for you (saves a lot of typing!)
We can flesh out the boilerplate code to print “Hello World!” to the console.

```java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

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

Add tags such as “@author” or “@param”
Running the program prints “Hello World!” to console

Run program by right clicking on program text and selecting “Run <class name>.main()”

Output appears in console below
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.

- 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).

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.
ENOUGH TALK
SHOW US HOW IT WORKS
```java
/**
 * Animated blob, defined by a position and size.
 * Version 0: just core instance variables and a main method.
 *
 * @author Chris Bailey-Kellogg, Dartmouth CS 10, Spring 2016, based on animated agents from previous terms
 */

public class Blob0 {
    double x, y;  // x,y position on screen
    double r;  // radius

    public static void main(String[] args) {
        Blob0 bob = new Blob0();
        System.out.println("bob starts at ("+bob.x+","+bob.y+)");
        bob.x = 100;
        bob.y = 50;
        System.out.println("bob is now at ("+bob.x+","+bob.y+");
    }
}
```
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 blob’s location and size
• Must declare type (double is a numeric value that has a 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

```java
public class Blob0 {
    double x, y; // x,y position on screen
    double r; // radius

    public static void main(String[] args) {
        Blob0 bob = new Blob0();
        System.out.println("bob starts at \(+(bob.x+","bob.y+\)\));
        bob.x = 100;
        bob.y = 50;
        System.out.println("bob is now at \(+(bob.x+","bob.y+\)\));
    }
}
```
Blob0: Our first “real” class uses instance variables to store data about objects

```java
public class Blob0 {
    double x, y; // x,y position on screen
    double r; // radius

    public static void main(String[] args) {
        Blob0 bob = new Blob0();
        System.out.println("bob starts at ("+bob.x+","+bob.y+")");
        bob.x = 100;
        bob.y = 50;
        System.out.println("bob is now at ("+bob.x+","+bob.y+")");
    }
}
```

- Location of “bob the Blob” is printed to console
- x and y are updated (like in Python)
- New location is printed
Blob0: Our first “real” class uses instance variables to store data about objects when run, 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: Declaring instance variables as “protected” prevents outside modification

- “Protected” allows this class (and subclasses) to access instance variables
- Others cannot
- More on this later
Blob01: Add a “setter” method to allow the object to update its own instance variables

```
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. Why is it there?
     * Also demonstrate the Javadoc feature by mousing over bob.setX(500) on line 26 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")");
    }
}*/
```

- “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)
- Note the one line syntax!
Blob01: Add a “setter” method to allow the object to update its own instance variables

```
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. Why is it there?
     * Also demonstrate the Javadoc feature by mousing over bob.setX(500) on line 26 below.
     */
    @param x value that instance variable x will set to
    /**
     * This.x refers to the parameter
     * this is like self in Python
     */
    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+"\n");
        bob.setX(500);
        System.out.println("bob is now at "+bob.x+","+bob.y+"\n");
    }
```
Blob01: Add a “setter” method to allow the object to update its own instance variables

```java
/**
 * Animated blob, defined by a position and size.
 * Version 0: just core instance variables and a main method.
 *
 * @author Chris Bailey-Kellogg, Dartmouth CS 10, Spring 2016, based on animated agents from previous terms
 * @author Tim Pierson, Dartmouth CS 10, Winter 2018, added setX method
 */

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. Why is it there?
     * Also demonstrate the Javadoc feature by mousing over bob.setX(500) on line 26 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+");
    }
}
```

Instance variable x now updated though setter method, rather than accessed directly like Blob0
Blob01: Javadoc allows you to document your methods

- JavaDoc describes what this method does and what parameters it takes
- Get used to writing these!
```java
/**
 * Animated blob, defined by a position and size.
 * Version 0: just core instance variables and a main method.
 *
 * @author Chris Bailey-Kellogg, Dartmouth CS 10, Spring 2016, based on animated agents from previous terms
 */

public class Blob02 {
    protected double x, y, r=5; // position

    // Do nothing; everything has its default value
    // This constructor is implicit unless you provide an alternative

    @param x initial x coordinate
    @param y initial y coordinate

    public Blob02(double x, double y) {
        this.x = x;
        this.y = y;
    }

    public static void main(String[] args) {
        Blob02 bob = new Blob02(); // calls first constructor, x=0, y=0, r=5
        Blob02 alice = new Blob02(x: 10, y: 7); // calls second constructor, x=10, y=7, r=5
        System.out.println("bob starts at (" + bob.x + "," + bob.y + ")");
        System.out.println("alice starts at (" + alice.x + "," + alice.y + ")");
    }
}
```
Blob02: Instance variables can be initialized to values other than zero

- `r` is initialized to 5 for all new objects
- `x` and `y` initialized to 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)

- 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 (signature)
- What value does r get?

```java
public class Blob02 {
    protected double x, y, r=5;  // position

    public Blob02() {
        // Do nothing; everything has its default value
        // This constructor is implicit unless you provide an alternative
    }

    public Blob02(double x, double y) {
        this.x = x;
        this.y = y;
    }
}
```

Why do we use “this” here?
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)

```java
/**
 * Animated blob, defined by a position and size.
 * Version 0: just core instance variables and a main method.
 * *
 * @author Chris Bailey-Kellogg, Dartmouth CS 10, Spring 2016, based on animated agents from previous terms
 */

public class Blob02 {
    protected double x, y, r=5; // position

    public Blob02() {
        // Do nothing; everything has its default value
        // This constructor is implicit unless you provide an alternative
    }

    /**
     * @param x initial x coordinate
     * @param y initial y coordinate
     */
    public Blob02(double x, double y) {
        this.x = x;
        this.y = y;
    }

    public static void main(String[] args) {
        Blob02 bob = new Blob02(); // calls first constructor, x=0, y=0, r=5
        Blob02 alice = new Blob02(x: 10, y: 7); // calls second constructor, x=10, y=7, r=5
        System.out.println("bob starts at ("+bob.x+","+bob.y+");
        System.out.println("alice starts at ("+alice.x+","+alice.y+");
    }
}
```

Output appears in console

bob starts at (0.0,0.0)
alice starts at (10.0,7.0)
public class Blob {
    protected double x, y, r=5; // position
    protected double dx=0, dy=0; // velocity, defaults to none
    protected double dr=0; // growth step (size and sign), defaults to none

    public Blob() {
        // Do nothing; everything has its default value
    }

    /**
     * @param x initial x coordinate
     * @param y initial y coordinate
     */
    public Blob(double x, double y) {
        this.x = x;
        this.y = y;
    }

    /**
     * @param x initial x coordinate
     * @param y initial y coordinate
     * @param r initial radius
     */
    public Blob(double x, double y, double r) {
        this.x = x;
        this.y = y;
        this.r = r;
    }
}
Blob: Small changes to previous versions give a more full featured Blob

```java
public class Blob {
    protected double x, y, r=5;    // position
    protected double dx=0, dy=0;   // velocity, defaults to none
    protected double dr=0;         // growth step (size and sign), defaults to none

    public Blob() {
        // Do nothing; everything has its default value
    }

    /**
     * @param x initial x coordinate
     * @param y initial y coordinate
     */
    public Blob(double x, double y) {
        this.x = x;
        this.y = y;
    }

    /**
     * @param x initial x coordinate
     * @param y initial y coordinate
     * @param r initial radius
     */
    public Blob(double x, double y, double r) {
        this.x = x;
        this.y = y;
        this.r = r;
    }
}
```

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

```java
public class Blob {
    protected double x, y, r=5;
    protected double dx=0, dy=0;
    protected double dr=0;

    public Blob() {
        // Do nothing; everything has its default value
    }

    /*
     * @param x initial x coordinate
     * @param y initial y coordinate
     */
    public Blob(double x, double y) {
        this.x = x;
        this.y = y;
    }

    /*
     * @param x initial x coordinate
     * @param y initial y coordinate
     * @param r initial radius
     */
    public Blob(double x, double y, double r) {
        this.x = x;
        this.y = y;
        this.r = r;
    }

    // position
    // velocity, defaults to none
    // growth step (size and sign), defaults to none
}
```

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

```java
public double getX() {
    return x;
}

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

public double getY() {
    return y;
}

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

public double getR() {
    return r;
}

public void setR(double r) {
    this.r = r;
}
```

- Return type, only one return value
- Void means returns nothing
- Getters and setters for each instance variable
- Nothing special about the names `getY`, `setY` to get/set Y
- We use `get<Variable name>` by convention
- Could use another name to get or set Y, but other programmers will look for getter/setter in this format (but Java doesn’t care what we call them)
Blob: Small changes to previous versions give a more full featured Blob

```java
/**
 * Sets the velocity.
 * @param dx new dx
 * @param dy new dy
 */
public void setVelocity(double dx, double dy) {
    this.dx = dx;
    this.dy = dy;
}

/**
 * Sets the direction of growth.
 * @param dr new dr
 */
public void setGrowth(double dr) {
    this.dr = dr;
}
```

`setVelocity()` sets `dx` and `dy`
Can update multiple variables in one call if you want, but you’d also normally have a `setDX` and `getDX`

`setGrowth()` sets `dr`
This is ok, but doesn’t follow Java’s typical naming convention
Blob: New methods to control how the blob behaves

```java
/**
 * Updates the blob, moving & growing.
 */
public void step() {
    x += dx;
    y += dy;
    r += dr;
}

/**
 * Tests whether the point is inside the blob.
 * @param x2
 * @param y2
 * @return is (x2,y2) inside the blob?
 */
public boolean contains(double x2, double y2) {
    double dx = x-x2;
    double dy = y-y2;
    return dx*dx + dy*dy <= r*r;
}

/**
 * Draws the blob on the graphics. Called by jwt everytime repaint called.
 * @param g
 */
public void draw(Graphics g) {
    g.fillOval((int)(x-r), (int)(y-r), (int)(2*r), (int)(2*r));
}
```

- `step()`: Updates `x`, `y`, and `r` by `dx`, `dy`, and `dr`.
- Will soon 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

```java
/**
 * Updates the blob, moving & growing.
 */
public void step() {
    x += dx;
    y += dy;
    r += dr;
}

/**
 * Tests whether the point is inside the blob.
 * @param x2
 * @param y2
 * @return is (x2,y2) inside the blob?
 */
public boolean contains(double x2, double y2) {
    double dx = x-x2;
    double dy = y-y2;
    return dx*dx + dy*dy <= r*r;
}

/**
 * Draws the blob on the graphics. Called by jwtevrytime repaint called.
 * @param g
 */
public void draw(Graphics g) {
    g.fillOval((int)(x-r), (int)(y-r), (int)(2*r), (int)(2*r));
}
```

`contains(x2,y2)`
- Checks to see if a point at $x_2, y_2$ is inside the blob's radius
- Returns true if point is contained, otherwise false

Three types of variables present in `contains()`:
- Parameters $(x_2,y_2)$
- Instance variables $(x,y,r)$
- Local variables $(dx,dy)$
- Local variables “hide” instance variables here!

- `draw()` function displays blob on screen
- More on that later
BlobDriver.java: Create a “driver” program separate from class definitions

```java
/**
 * Test case for the Blob class
 */

/* @author Chris Bailey-Kellogg, Dartmouth CS 10, Spring 2016 */

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

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

```java
/**
 * Test case for the Blob class
 * @author Chris Bailey-Kellogg, Dartmouth CS 10, Spring 2016
 */

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

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

Key point: Blob class is not defined in the “driver” or “application” class BlobDriver, but BlobDriver can use the Blob code if it is in the same project

BlobDriver class in file BlobDriver.java
Uses class Blob defined in file Blob.java
Reference to Blob allowed if classes defined in same project (cs10 here)
BlobDriver: Uses Blob class to create blob objects and then move/grow them

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

```
/**
 * Test case for the Blob class
 * 
 * @author Chris Bailey-Kellogg, Dartmouth CS 10, Spring 2016
 */

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

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

- `step()` adds $dx$, $dy$, and $dr$ to $x$, $y$, and $r$
- Print bob and alice locations to console
BlobDriver: Uses Blob class to create blob objects and then move/grow them.

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

        // Set some of their parameters
        bob.setR(2 * alice.getR()); // make bob's radius 2x alice's (alice 5 by default)
alice.setVelocity(dx: 3, dy: 4); // cause alice to move right 3 pixels, and down 4 pixels each step
        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());
    }
```
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
  • We will use IntelliJ IDEA for this course
• Create your first Java class
• Read and acknowledge course policies and honor code
• Complete survey before 8:00am tomorrow (or risk getting assigned to inconvenient section time!)
• X-hour this week