CSCI 161
Introduction to Computer Science

Est. 1888
UNIVERSITY of PUGET SOUND

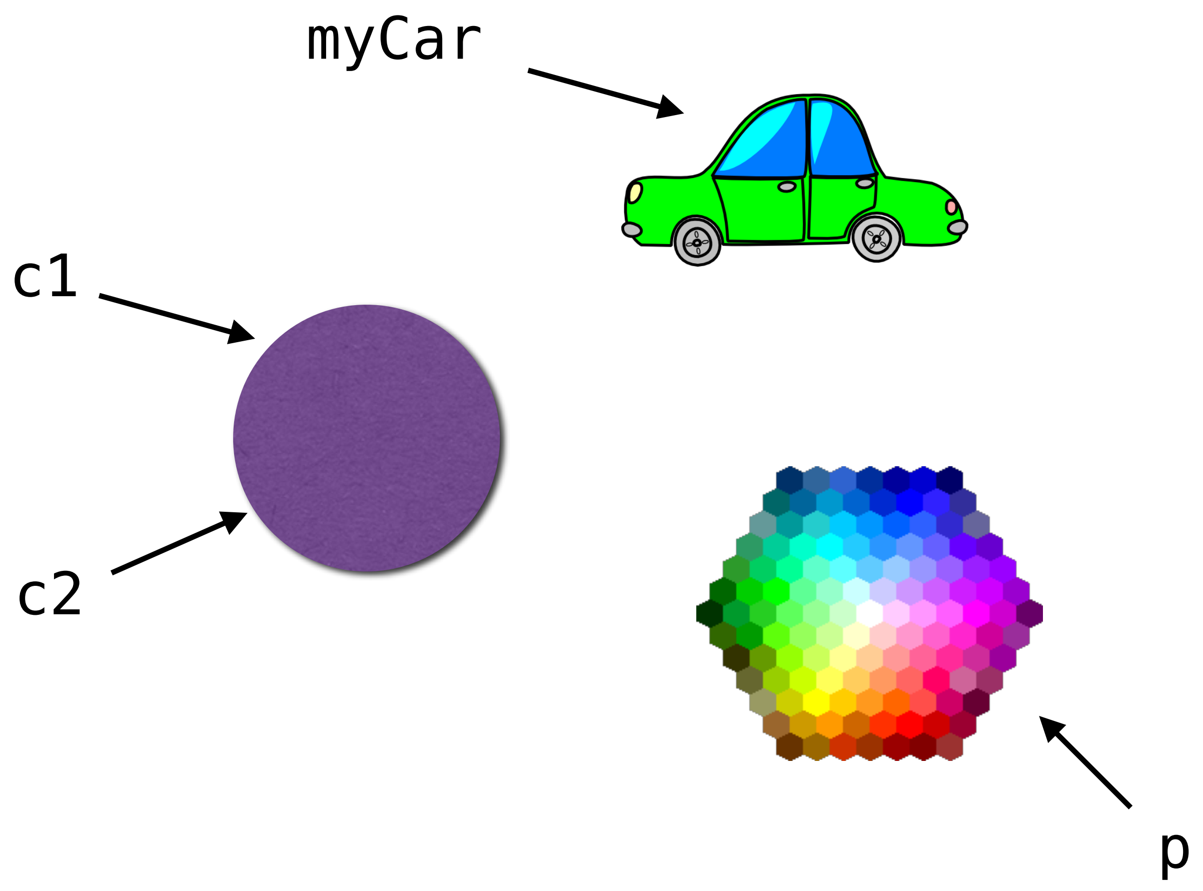
Department of Mathematics
and Computer Science

Lecture 2a
Classes and Objects

Outline

Objects in Software

- In software an *object* models some real-world element. Can be ANYTHING, like:
  - Circle
  - Squares
  - Car
  - Thermostat
  - Color palette
  - PacMan
  - *and so on...*

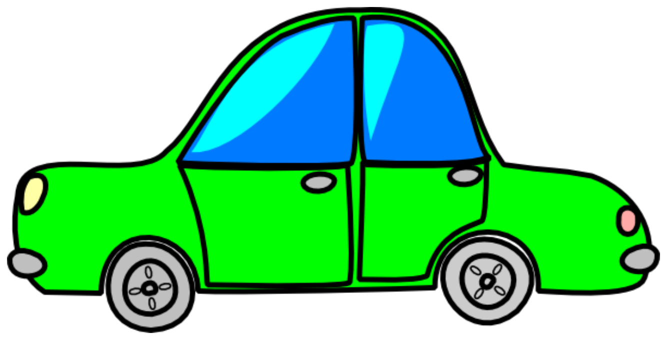


- Objects usually have at least one name that reference it
  - (We get to name objects)

Object State

- **Important:** Objects have both *state* and *behavior*

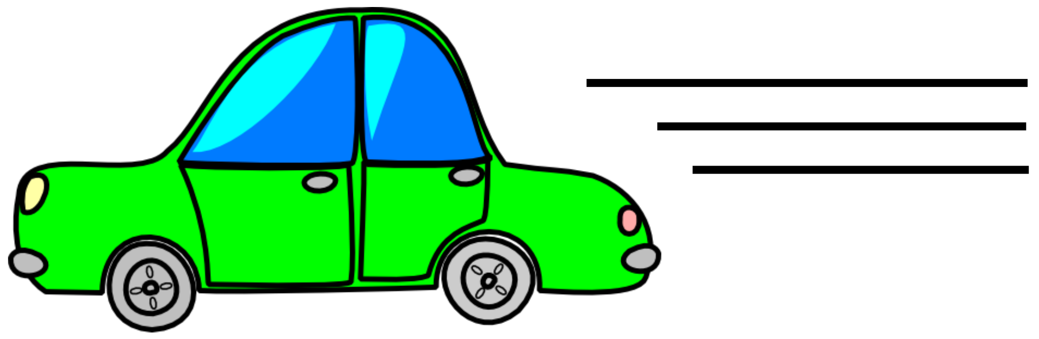
- *State*: A set of things (nouns) that an object remembers about itself
  - Example state for a Car object:
    - Current speed
    - Current amount of gas
    - Mileage
    - Color
    - Number of doors
    - *(May be more...)*



Object Behavior

- **Important:** Objects have both *state* and *behavior*

- *Behavior:* A set of actions (verbs) an object knows how to do
  - Example behavior of Car objects
    - Accelerate
    - Decelerate
    - Turn left
    - Turn right
    - Shift gear
    - *(May be more...)*



"Classes" vs. "Objects"

- Classes group together objects that share a common set of states and behaviors.

Circle
"Class"

```
diameter
color
xPosition
yPosition
isVisible
```

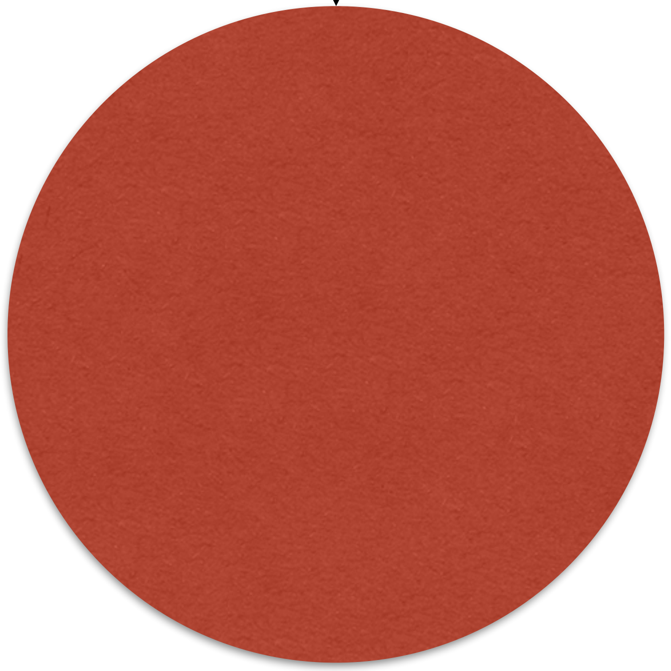
State

```
moveUp()
moveDown()
changeColor()
changeSize()
```

Behavior

"instantiate"

Circle
"Object"

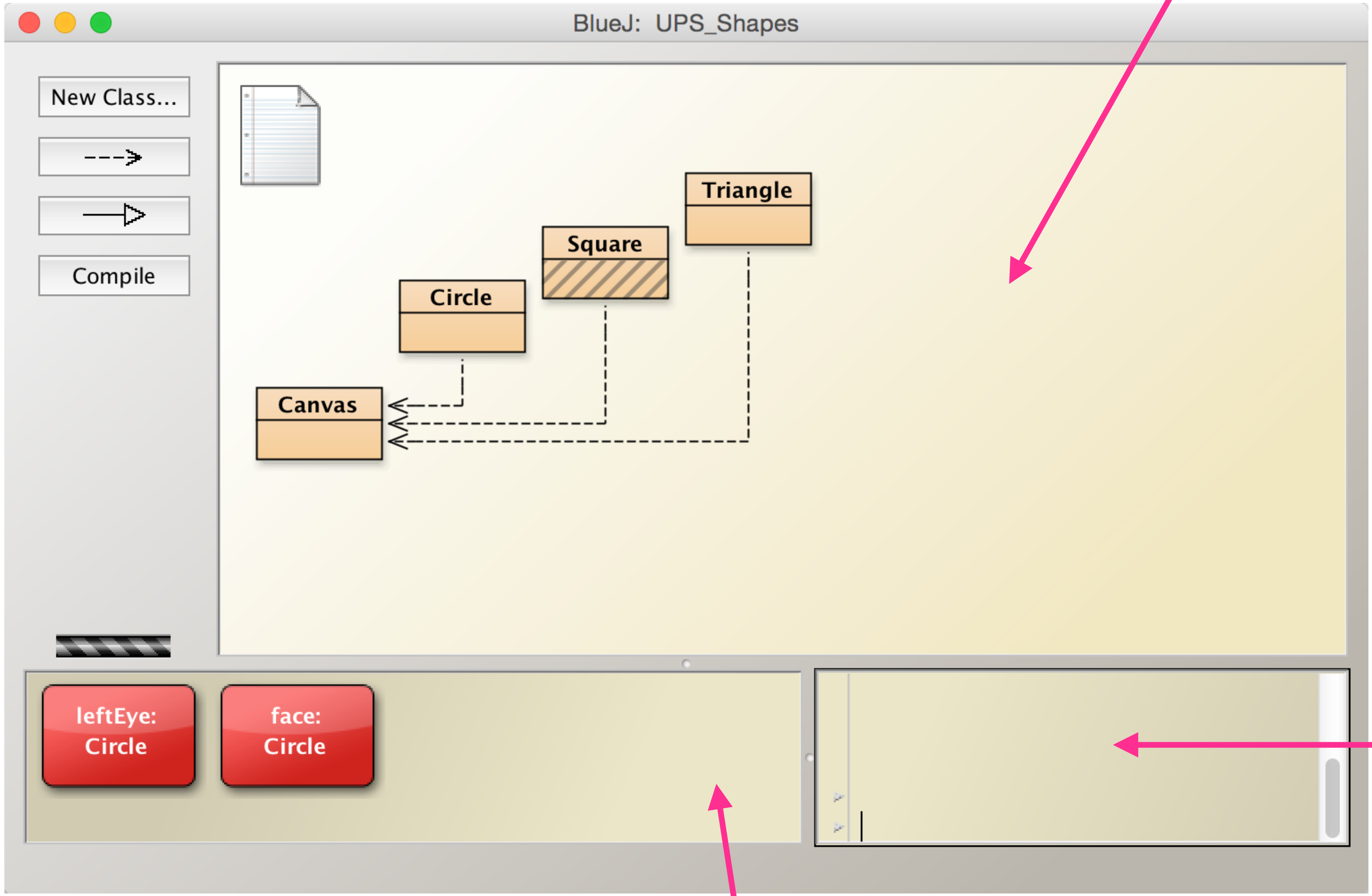


```
diameter: 50
color: red
xPosition: 30
yPosition: 20
isVisible: true
```

```
moveUp()
moveDown()
changeColor()
changeSize()
```

BlueJ Project Window

Project Explorer
Contains documentation and related classes



Code Pad: Lets us write snippets of Java code

Workbench: shows what objects have been created

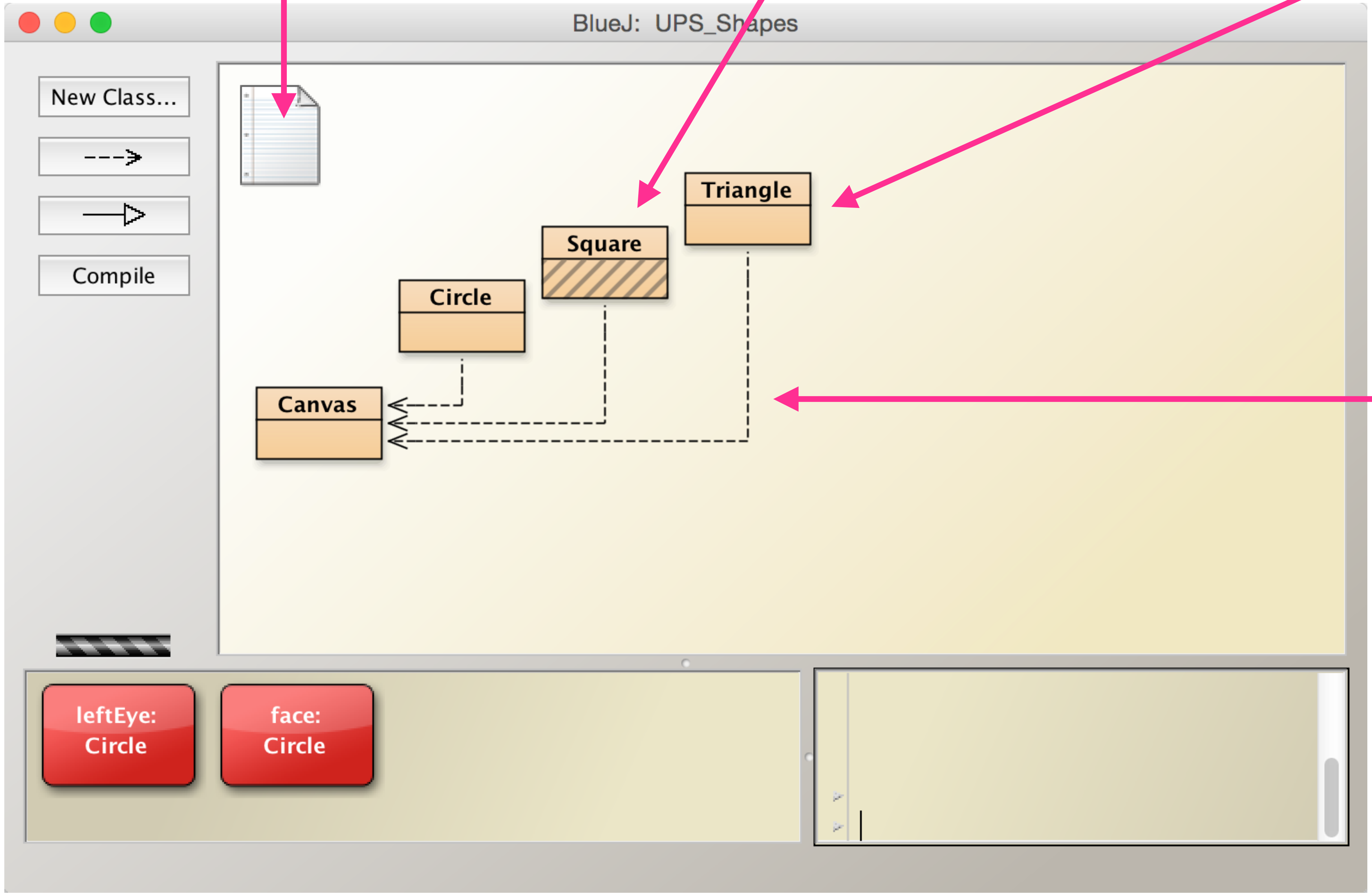
BlueJ Project Window

README File
Description of this project

Greyed-Out Box
An uncompiled class

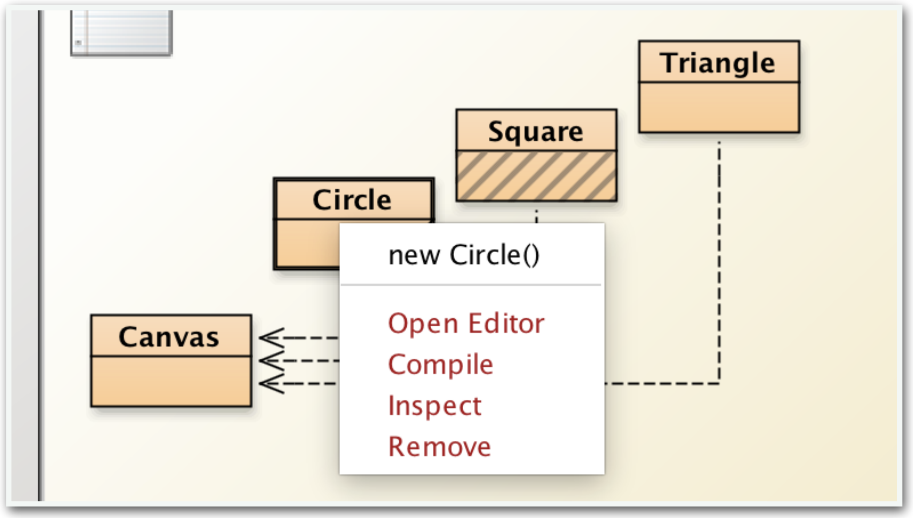
Solid Box
A compiled class

Dotted Arrows
"Uses" relationship

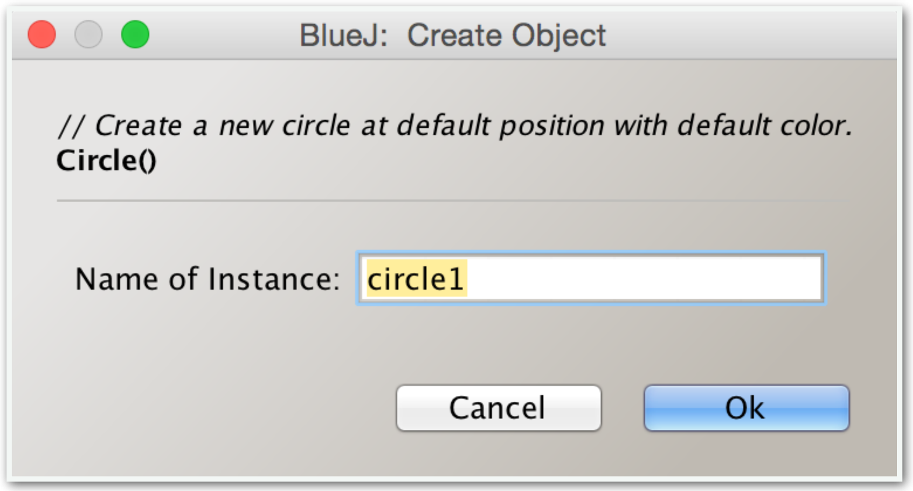


Instantiation (Creation)

- To create a new Circle object (instance), right click on the class and select ***`new Circle()`***



- Now BlueJ will ask you to provide a name for this instance:

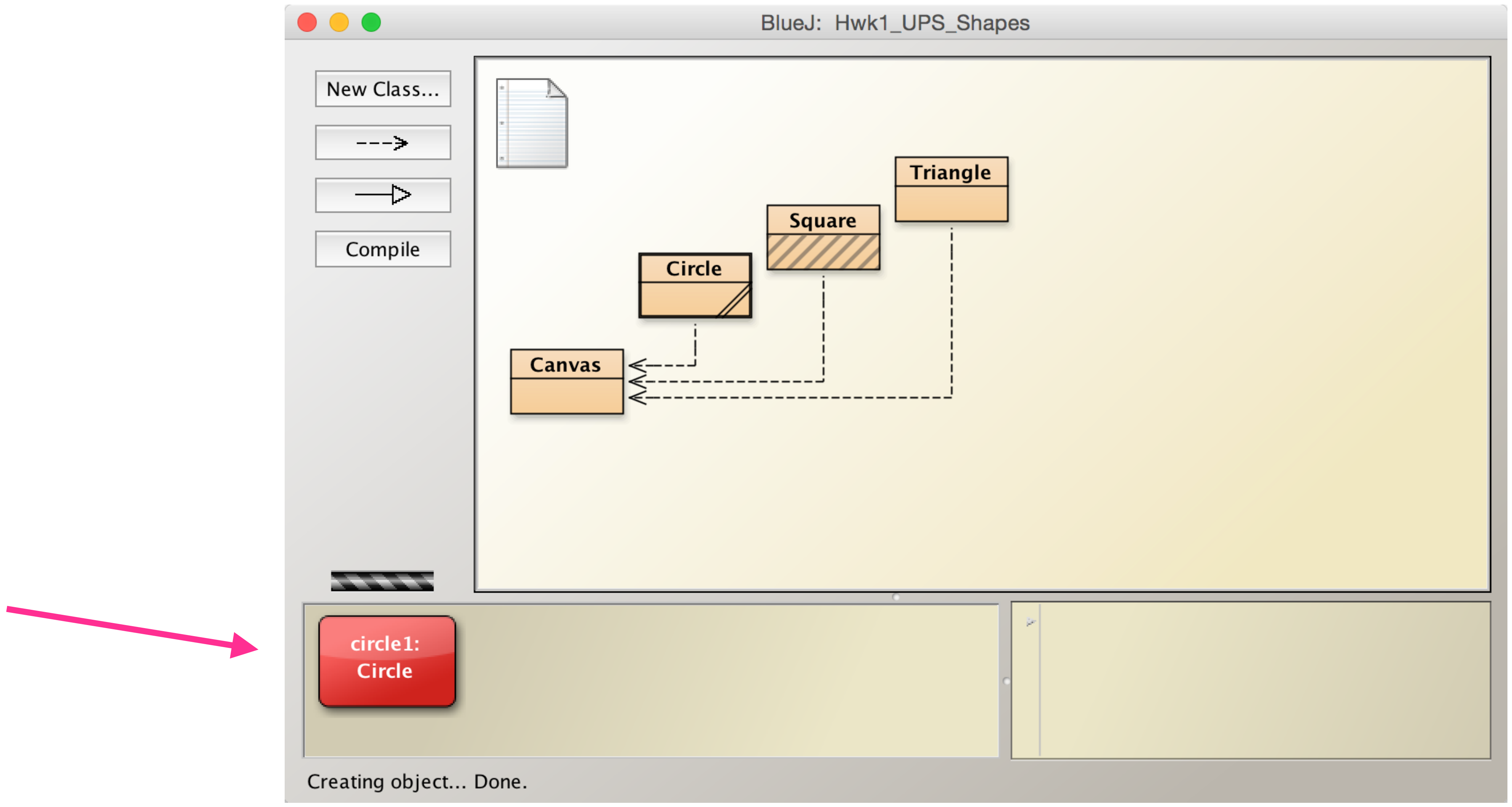


// Create a new circle at default position with default color.
Circle()

Name of Instance: circle1

Inspecting the Object's State

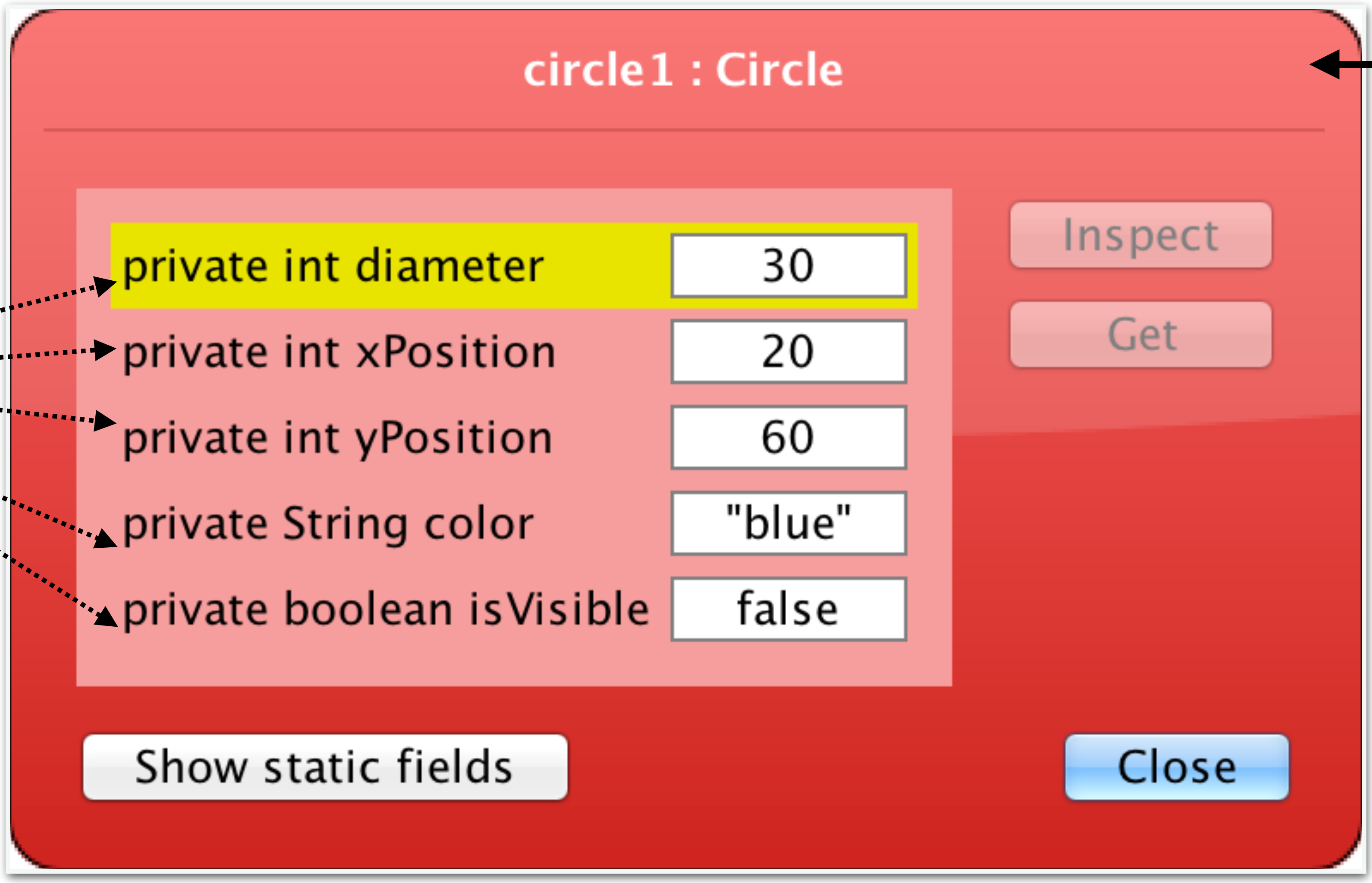
- The new object appears in the Object Workbench.
  - Note: You can instantiate as many objects as you want!



Creating object... Done.

Inspecting the Object's State

- Let's take a look at this particular Circle's state.
  - Double-click on the object's box icon from the Workbench
  - No upper or lower limit to the number of fields an object can have



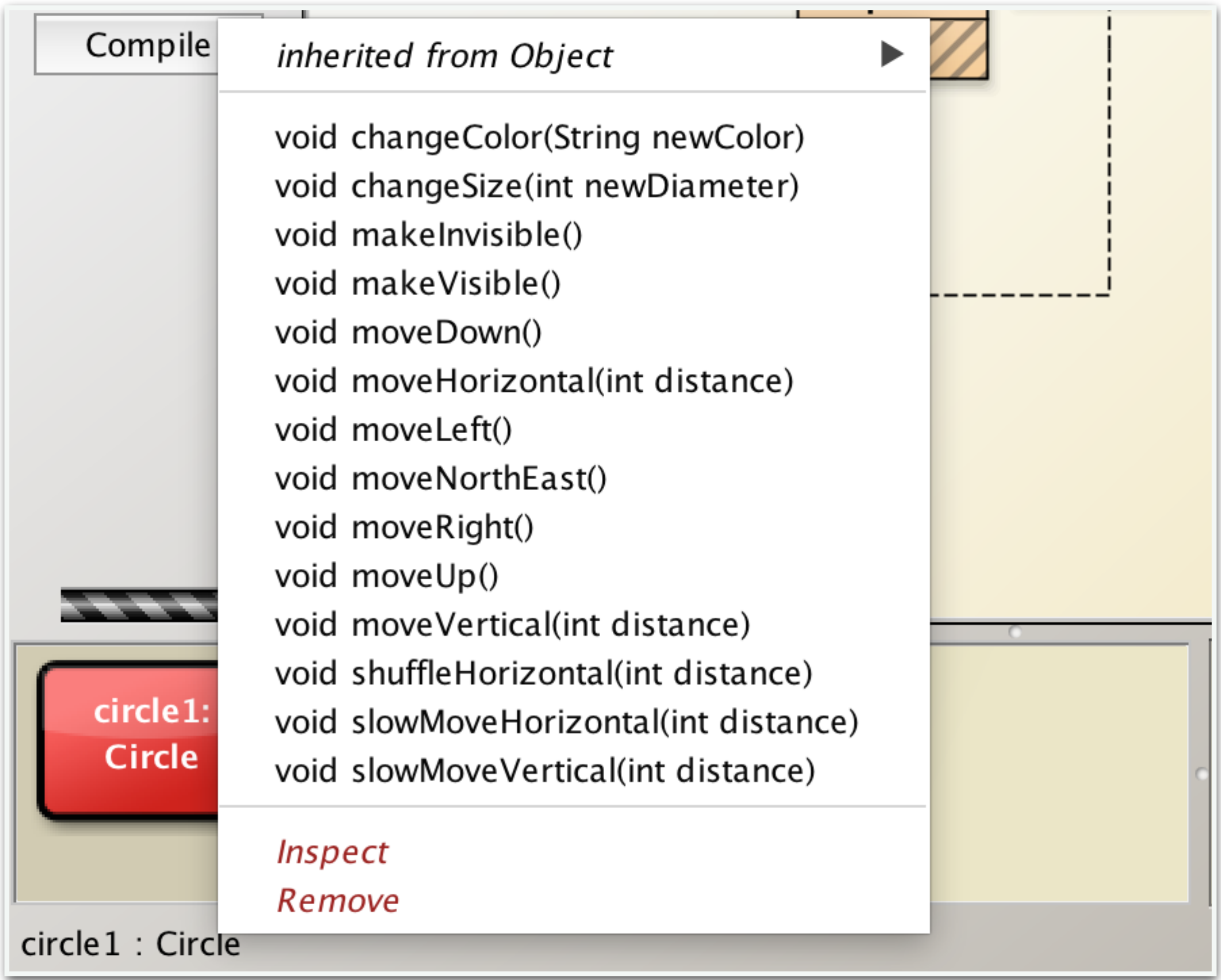
private int diameter	30
private int xPosition	20
private int yPosition	60
private String color	"blue"
private boolean isVisible	false

Instance variables

State of circle1

Accessing an Object's Methods

- Now right-click on this object, and you get a menu showing its *instance methods*

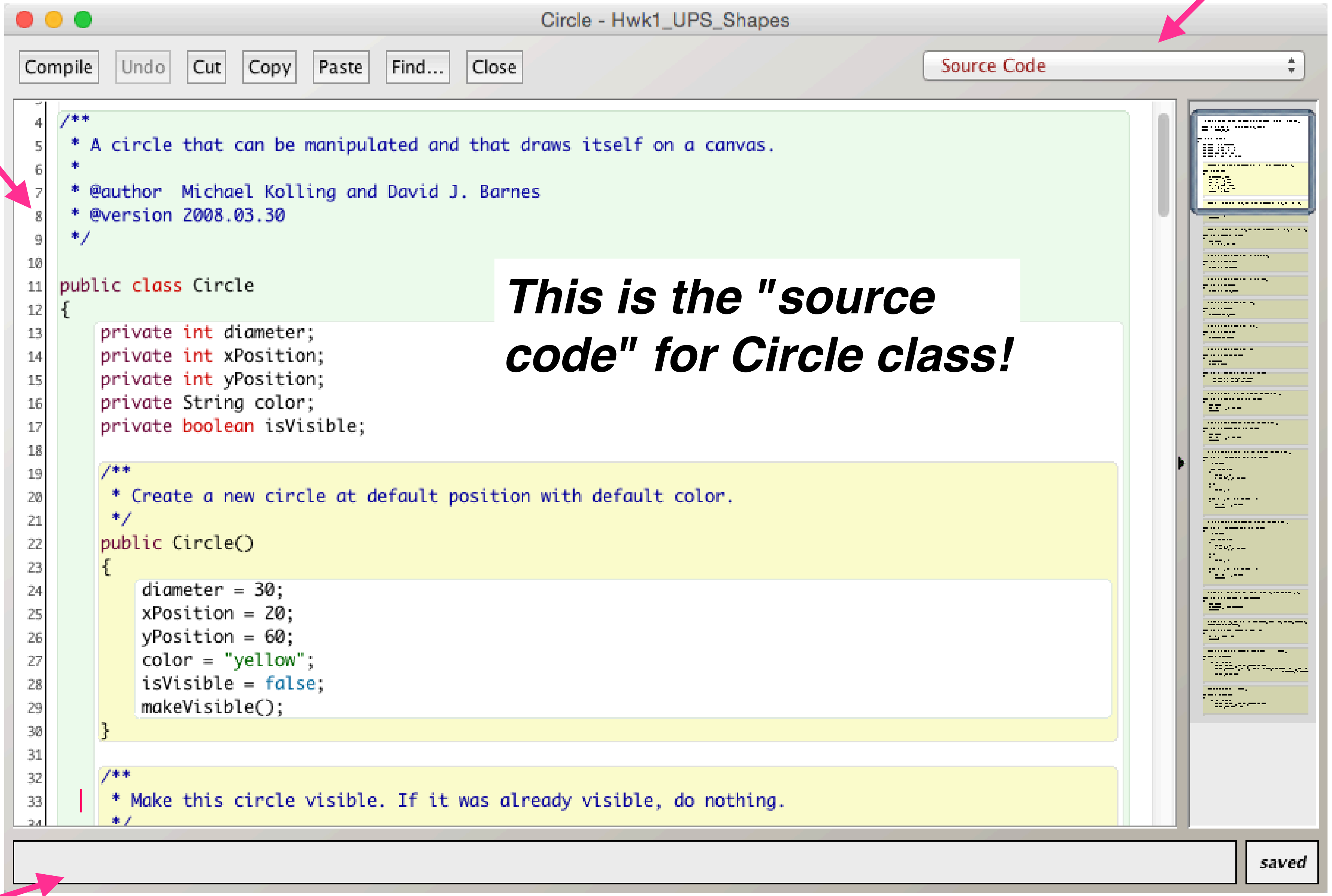


void changeColor(String newColor)
void changeSize(int newDiameter)
void makeInvisible()
void makeVisible()
void moveDown()
void moveHorizontal(int distance)
void moveLeft()
void moveNorthEast()
void moveRight()
void moveUp()
void moveVertical(int distance)
void shuffleHorizontal(int distance)
void slowMoveHorizontal(int distance)
void slowMoveVertical(int distance)

Source Code of a Class

Line Numbers (disabled by default)
To Enable: Options menu > Preferences > Display Line Numbers

Source Code or Documentation



```
/**
 * A circle that can be manipulated and that draws itself on a canvas.
 *
 * @author  Michael Kolling and David J. Barnes
 * @version 2008.03.30
 */

public class Circle
{
    private int diameter;
    private int xPosition;
    private int yPosition;
    private String color;
    private boolean isVisible;

    /**
     * Create a new circle at default position with default color.
     */
    public Circle()
    {
        diameter = 30;
        xPosition = 20;
        yPosition = 60;
        color = "yellow";
        isVisible = false;
        makeVisible();
    }

    /**
     * Make this circle visible. If it was already visible, do nothing.
     */
```

This is the "source code" for Circle class!

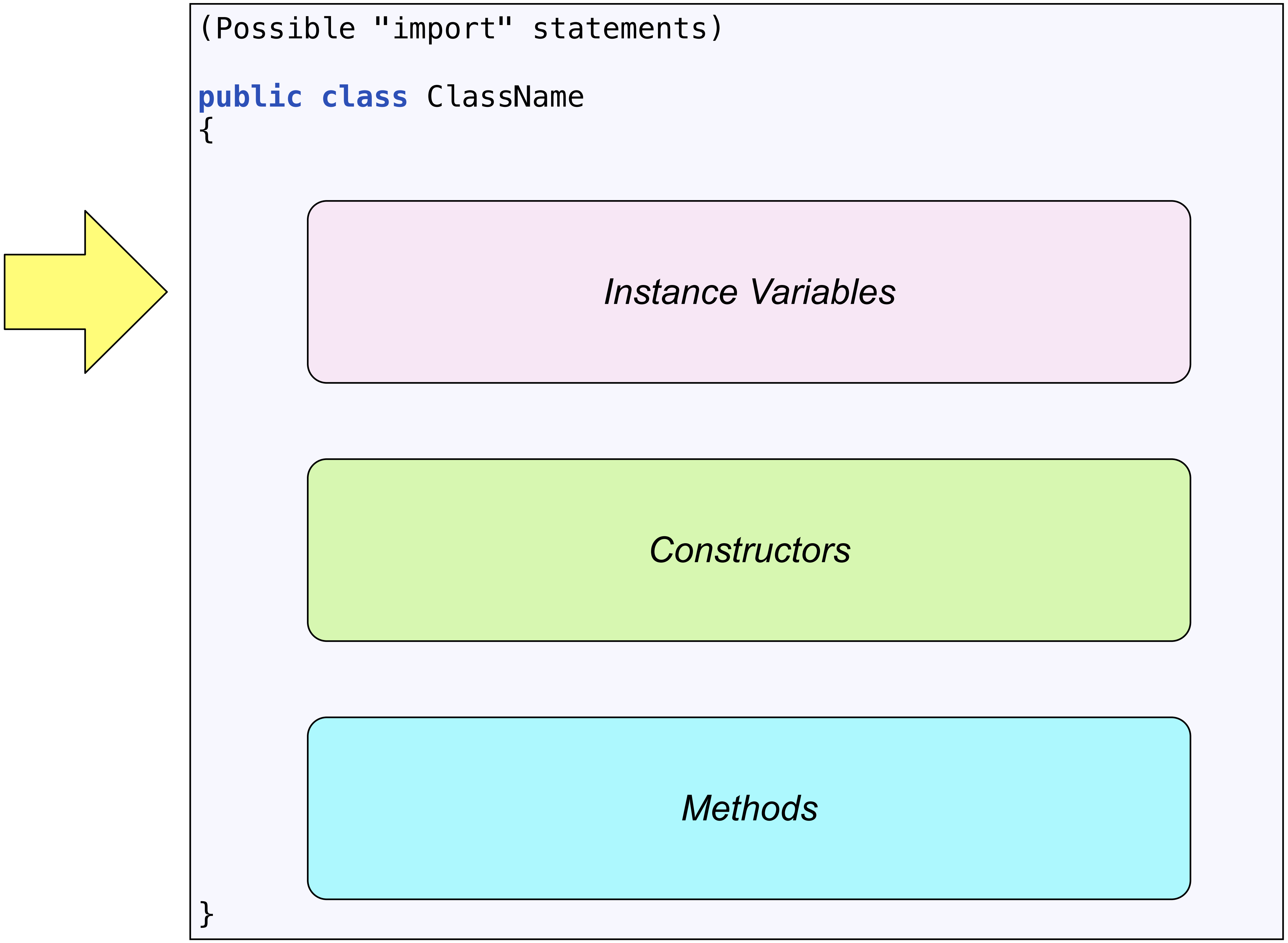
Compile Status
Any syntax errors found when compiling the source code??

File Status
Saved or Changed

Outline

- What Are Objects?
  - State and Behavior
- A Peek at the Source Code
  - Instance Variables and Data Types
  - Constructors
  - Methods
  - Comments
- Conclusion

Basic Class Structure



```
(Possible "import" statements)

public class ClassName
{

}
```

Instance Variables (or "Fields")

- An *Instance Variable* is a property that all objects of that class must remember about itself.
  - Instance variables of the Circle class:

```
private int diameter;
private int xPosition;
private int yPosition;
private String color;
private boolean isVisible;
```

  - Each instance variable is "declared" using this format:

```
private data-type variable-name;
```

What are these? *(next slide)*

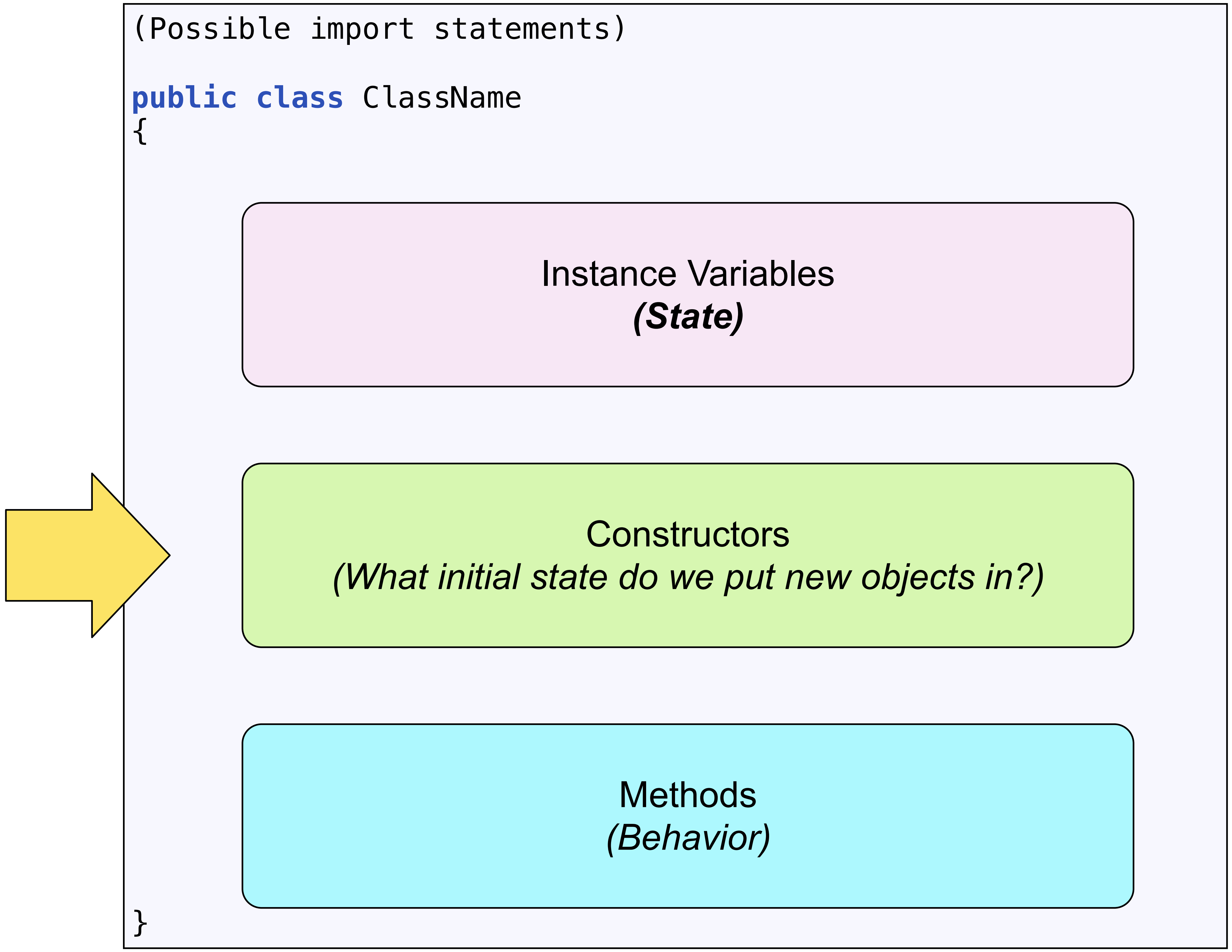
You should to give the instance variable a descriptive name.

Common Data Types

- Each instance variable is required to have a Data Type.
  - Data types declare the nature of data that a variable can hold.

- Common data types:
  - An `int` can only store an integer (i.e., whole number).
  - A `double` can store a number with a decimal point.
  - A `boolean` is a true or false (yes or no) value.
  - A `String` is a sequence of letters, symbols, and numbers
    - Values are *always* enclosed in `"double quotes"`
    - Use Strings to store a word, a phrase, a sentence, a paragraph, ...

Basic Class Structure



```
(Possible import statements)

public class ClassName
{
```

Instance Variables
(State)

Constructors
(What initial state do we put new objects in?)

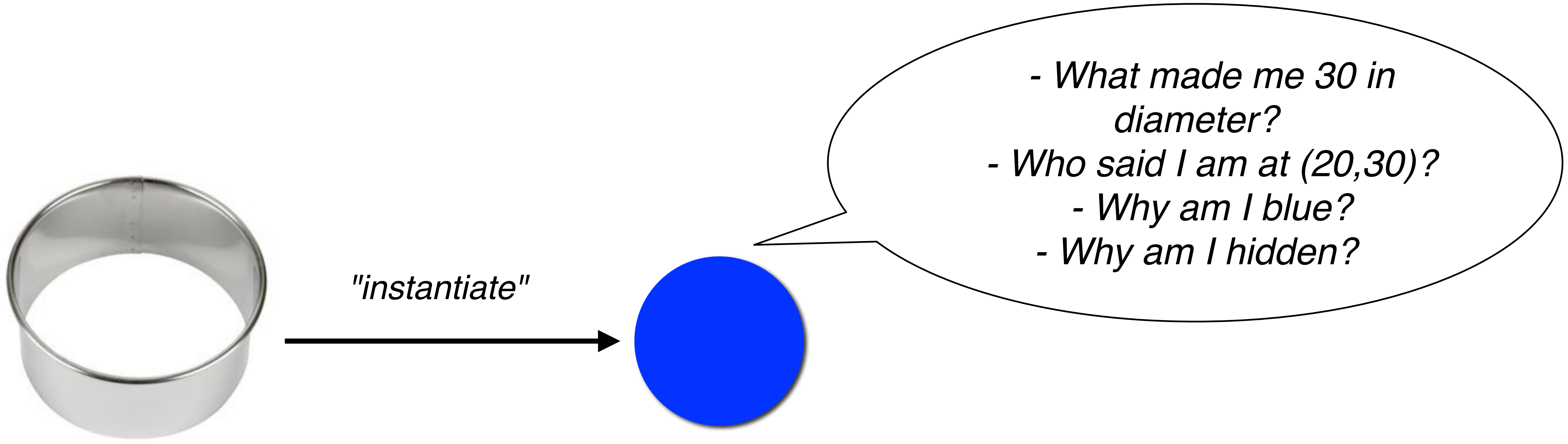
Methods
(Behavior)

```
}
```

Def'n: Constructors and Instantiation

- **Definition:** *"Instantiation"* - The act of creating an object of a class.

- **Definition:** *"Constructor"*
  - The constructor's code is run immediately upon "instantiation."
  - The code must "assign" values to the instance variables of the new object.



Class ("cookie cutter") *Object or Instance*

Constructor in the `Circle` Class

- The *"constructor"* is executed each time a new *object* is instantiated.

```java
public class Circle
{
    //fields
    private int diameter;
    private int xPosition;
    private int yPosition;
    private String color;
    private boolean isVisible;

    public Circle() {
        diameter = 30;
        xPosition = 20;
        yPosition = 60;
        color = "blue";
        isVisible = false;
    }

   //(instance methods omitted)
}
```

Variable Assignment Syntax:

```
variable-name = expression;
```

Constructor (Cont.)

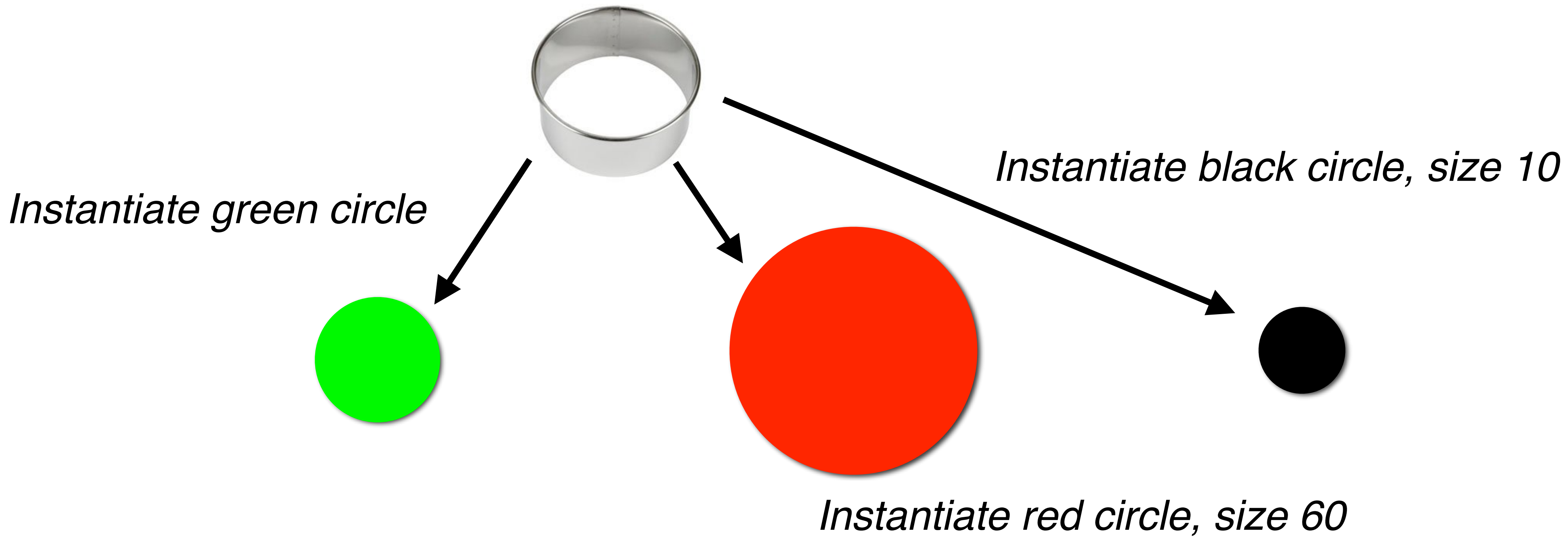
- **Syntax:** This is how you write a Constructor in Java

```
public ClassName(input-parameters) {
    // code to initialize all the
    // instance variables of the new object
}
```

- **Important:** remember these rules!
  - Constructors are ***always*** ...
    - Named after the class
  - May have a list of input parameters to accept data
    - Declare each input parameter separated by comma
    - If the parameter list is **empty**, it's called the *"default constructor"*

Overloading Constructors

- We can write (or "overload") multiple constructors
  - Suppose we want to give users more options to instantiate Circles
  - [Together] Let's write another constructor that lets users input the initial size.
  - [You] Write a 3rd constructor that lets users input the initial size and color.


Instantiate green circle
Instantiate black circle, size 10
Instantiate red circle, size 60

3 Constructors = 3 Ways to Create Objects

```
public class Circle {
    private int diameter;
    private int xPosition;
    private int yPosition;
    private String color;
    private boolean isVisible;

    public Circle() {
        diameter = 30;
        xPosition = 20;
        yPosition = 60;
        color = "blue";
        isVisible = false;
    }
```

(Instantiates a default circle)

```
    public Circle(int initialSize) {
        diameter = initialSize;
        xPosition = 20;
        yPosition = 60;
        color = "blue";
        isVisible = false;
    }
```

(Instantiates a circle but inputs a diameter from user)

```
    public Circle(int initialSize, String initialColor) {
        diameter = initialSize;
        xPosition = 20;
        yPosition = 60;
        color = initialColor;
        isVisible = false;
    }
```

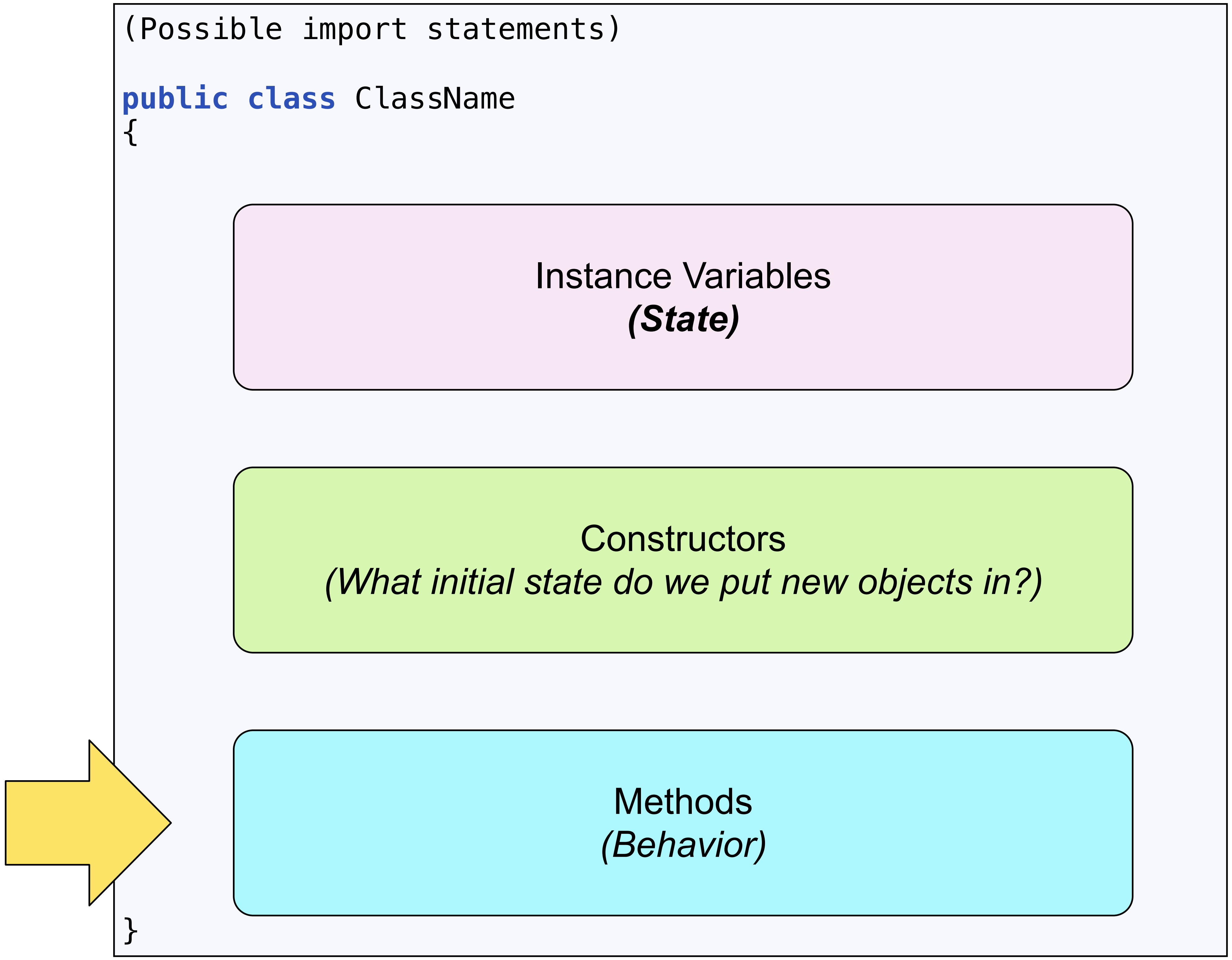
(Inputs a diameter and color from user)

Self-Check: Adding Instance Variables

- We can have as many "instance variables" as we need.

- Suppose we also want **Circles** to remember their areas.
  - Things to consider:
    - First, we need to declare a new instance variable **`area`** in the Circle code.
      - *What data-type should it be declared as?*
    - How does **`area`** get assigned a value?
      - *(Deal with this next)*
    - Maintenance: What if the Circle changes size later?
      - *(Deal with this later)*

Basic Class Structure



```
(Possible import statements)

public class ClassName
{

}
```

Method Writing Syntax

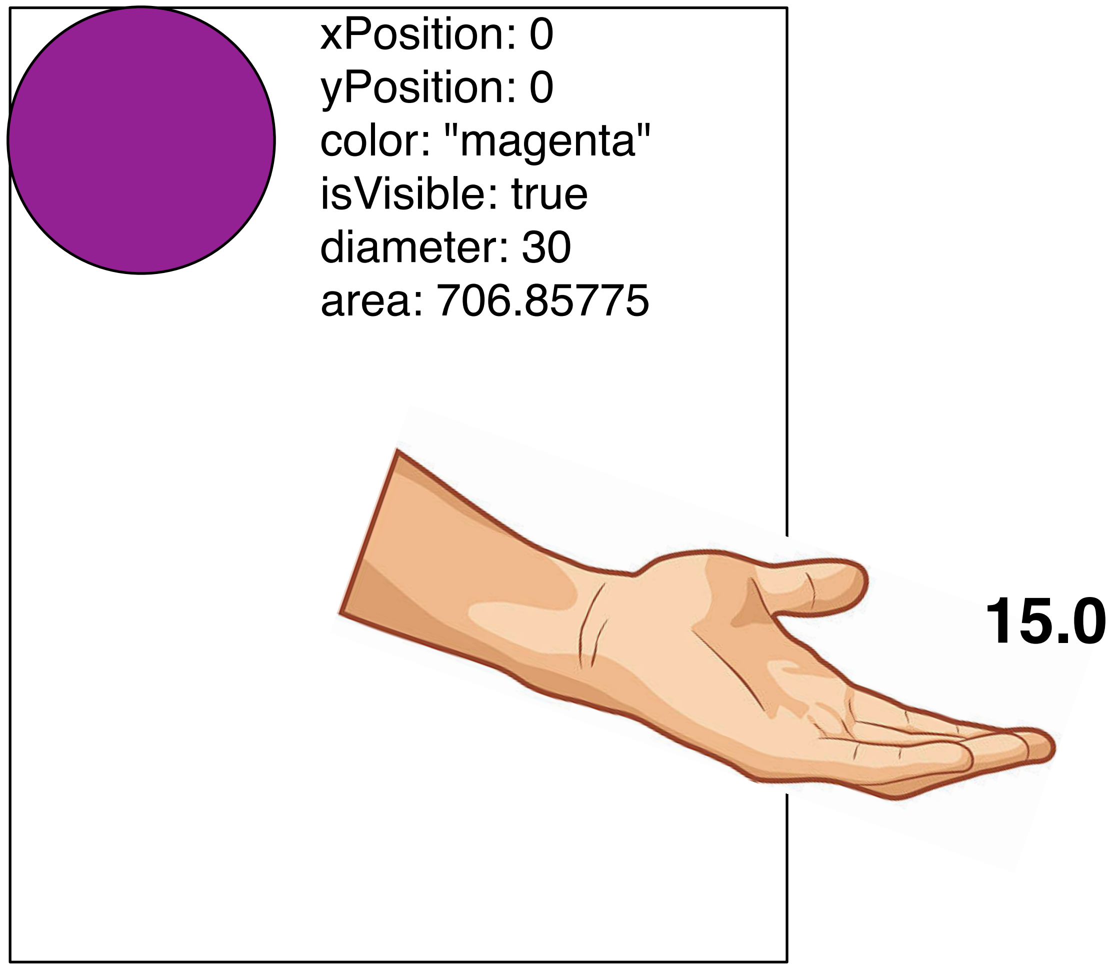
- When writing a *method*, consider:
  - What should we name it? Usually an action verb.
  - What input(s) it requires, if any.
  - What's the body, i.e., its algorithm?
  - If the method **returns** (gives back) a value, what data type is it?
    - Use **void** if method does not return a value before it terminates

- **Syntax for writing a new method:**

```
public return-type method-name(input-parameters) {

    // instructions for the method routine

}
```

Let's Write a New Method: `getRadius()`

- Write a new method, called `radius()` that...
  - Accepts no inputs
  - **returns** the *radius* of the Circle object to the caller



getRadius() Solution

- Write a new method, called `radius()` that...
  - Accepts no inputs
  - **returns** the *radius* of a Circle object to the method caller

- Solution

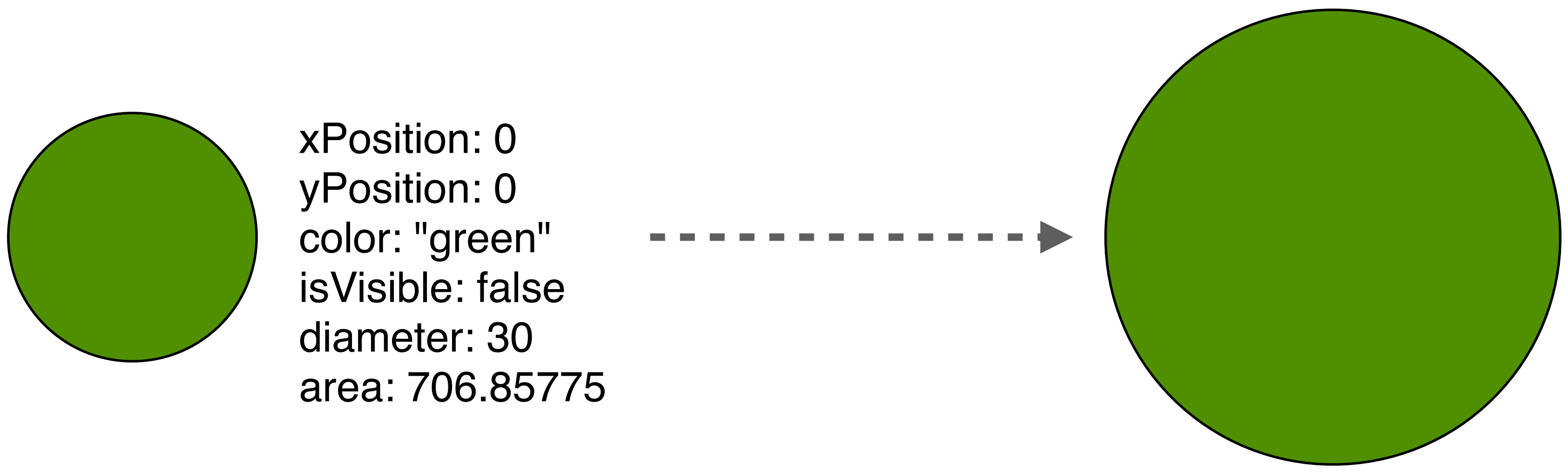
```java
public double radius() {
   double rad = size * 0.5;   // declare a local variable to store the radius
   return rad;
}
```

- An even simpler alternative:

```java
public double radius() {
    return size * 0.5;   // we don't really need to store the radius first
}
```

Method Writing Example: `enlarge()`

- **Example:** Write a new method called enlarge that enlarges the Circle to *twice its current size* and shows it to the canvas (even if currently hidden).
  - Accepts no inputs
  - Returns nothing to the method caller.



returns nothing (void)

Calling Existing Methods

- **Important:** Often perform something *new*, you can *call* existing methods that had been previously written.

- Before writing the new method,
  1. Think about whether there are any existing methods we *could* employ.
  2. Look through the source code and find their *signatures*.
     ```
     Examples: public void makeVisible()
               public void changeSize(int newDiameter)
     ```
  3. Make the call! Write the method's name, and give values to input parameters!
     ```
     Example to make a method call:
               changeSize(30);
     ```

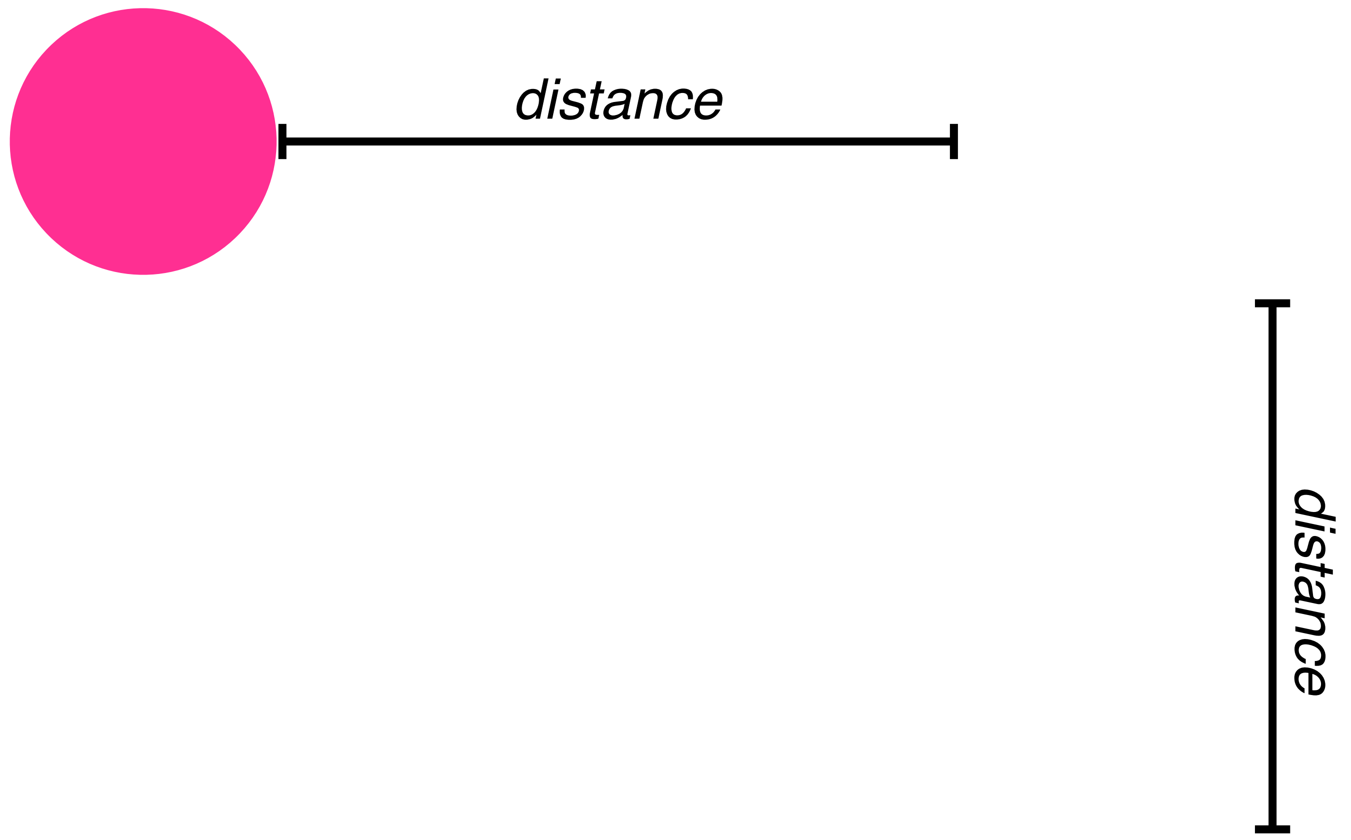
Method Writing Example (Soln)

- **Example:** Write a new method called enlarge() that enlarges the Circle to twice its current size and displays it on canvas.

```java
/**
 * Enlarges a circle to twice its current size
 * and makes it visible.
 */
public void enlarge() {

   // current size is stored in instance variable 'size'
   int newDiameter = size * 2;

   // call changeSize() to enlarge the circle!
   changeSize(newDiameter);

   // call makeVisible() to draw it on the canvas for me!
   makeVisible();
}
```

Group Work

- Write a method called `takeALap(int distance)` that causes the Circle to move East, South, West, North, ending at the original spot.
  - Each "slow move" is of the input distance.
  - **Returns** the total distance traveled to the caller.



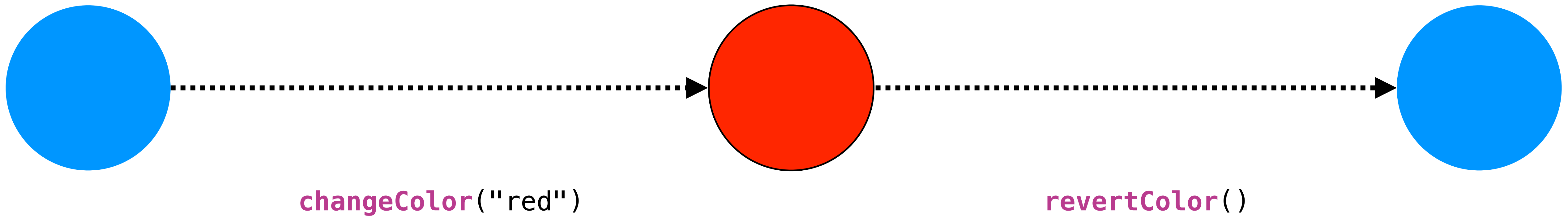
Group Work (Soln)

- Write a method called `takeALap(int distance)` that causes the Circle to move East, South, West, North, ending at the original spot.
  - Each directional move is of the given distance.
  - **Returns** the total distance traveled to the caller.
- Solution:

```java
/**
 * Causes Circle to take a lap of the given distance per edge.
 *
 * @param distance The length to travel in each direction
 * @return total distance traveled
 */
public int takeALap(int distance) {
   slowMoveHorizontal(distance);   // go east
   slowMoveVertical(distance);     // go south
   slowMoveHorizontal(-distance);  // go west
   slowMoveVertical(-distance);    // go north
   return distance * 4;
}
```

On your own

- Write a method, `revertColor()`, that reverts the color back to its *original* color (i.e., when the circle was first created).



Basic Class Structure

```
(Possible import statements)

public class ClassName
{
```

Fields (instance variables)
(State)

Constructors
(What initial state do we put new objects in?)

Methods
(Behavior)

```
}
```

Import Statements

- You may or may not see *import statements* at the top of your class

- They are used to include *pre-written classes* so that you don't have to copy and paste that code into the class you're writing!
  - For example:

```java
import java.util.Scanner;
import java.util.ArrayList;
```

  - The above would allow us to use the Scanner and ArrayList classes in the code we write.

Comments

- *Comments* are a programmer's notes to describe the code
  - They are *ignored* by the compiler
  - Super important to comment your code
- Two types of comments
  - Block (multi-line) comment:
    - Example:

```
/*
   Everything here is ignored by the compiler
   Can span any number of lines
 */
```

  - Line comment:
    - Example:

```
// Everything on this line is ignored by the compiler
```

Outline

- What Are Objects?
  - State and Behavior
- A Peek at the Source Code
  - Instance Variables and Data Types
  - Constructors
  - Methods
  - Comments
- Conclusion

In Conclusion

- Object-Oriented Programming (OOP)
  - A program is composed by a bunch of building-blocks (objects)

- Objects have:
  - Object State (Fields or Instance Variables)
    - Fields are memories that let an object remember things about itself
  - Object Behavior (Instance Methods)
    - Methods perform some action on the object
      - The action is defined by a sequence of statements (an algorithm)
    - Constructor is a special method that sets the default state

In Conclusion (cont.)

- We get to invent lots of names:
  - Class names
  - Variable names
  - Method names
  - Parameter names

- Rules
  - Class names start with upper case
  - Others start with lower case
  - Shift case at word boundaries
  - Use meaningful names!

In Conclusion (cont.)

- We saw some Java syntax:
  - Variable declaration and assignment
    - A few data types: int, boolean, String
  - Curly braces `{...}`
    - Group related statements together
  - Method signatures
    - Contain a name, parameter list

- Getting a sense of what OOP is all about: bossing objects around

- *Next time: More... Accessors/Mutators, Local Vars, Printing*