CSCI 161
Introduction to Computer Science
Outline

- **What Are Objects?**
  - State and Behavior

- **A Tour of BlueJ**
  - Point-and-Click Elements
  - A Peek at the Source Code
  - Bossing around Objects *without* Menus (with Code)

- Conclusion
Objects in Software

- In software an object models, or simulates, things in real-life
  - Circle
  - Car
  - Scroll bar
  - Thermostat
  - Color palette
  - (Literally anything...)

- Object usually have at least one reference (name) identifying it
Important: Objects have *state* and *behavior*

**Object 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 *state* and *behavior*

- **Object 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...)*
Outline

- What Are Objects?
  - State and Behavior
- A Tour of BlueJ
  - Point-and-Click Elements
  - A Peek at the Source Code
  - Bossing around Objects without Menus (with Code)
- Conclusion
A Tour of BlueJ (In class)

- Project View

**Project Explorer**
Contains documentation and related classes

**Code Pad:** Lets us write snippets of Java code

**Workbench:** shows what objects have been created
A Tour of BlueJ (In class)

- **README File**: Description of this project
- **Greyed-Out Box**: An uncompiled class
- **Solid Box**: A compiled class
- **Dotted Arrows**: "Uses" relationship
A Tour of BlueJ: Source Code View

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

Source Code or Documentation

Code Navigator

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

File Status
Saved or Changed
Outline

- What Are Objects?
  - State and Behavior

- A Tour of BlueJ
  - Point-and-Click Elements
  - A Peek at the Source Code
  - Bossing around Objects without Menus (with Code)

- Conclusion
Basic Class Structure

(Possible import statements)

```java
public class ClassName {

  Fields (instance variables)  
  \textit{(State)}

  Constructors  
  \textit{(What initial state do we put new objects in?)}

  Methods  
  \textit{(Behavior)}

}
```
Anatomy of the Circle Class: Comments

- **Comments** are a programmer's notes to describe the code
  - They are *ignored* by the compiler
  - Important to comment your code in this class

- Two types of comments
  - Block (multi-line) comment:
    - Example:
      ```
      /*
       * Everything here is ignored by the compiler
       * Can span any number of lines
      */
      ```
  - (Single) Line comment
    - Example:
      ```
      // Everything on this line is ignored by the compiler
      ```
Field declaration (A property that objects of this class remembers)

- Java syntax:

```java
private dataType variableName;
```

Fields are always private (for now)

What are these? (next slide)

Name of the storage unit (Are there any restrictions? Yes!)

- Fields in Circle class:

```java
public class Circle {
    // the state of Circle object is characterized by the following set of fields
    private int diameter; // the Circle's diameter
    private int xPosition; // the Circle's x coordinate
    private int yPosition; // the Circle's y coordinate
    private String color; // the Circle's color
    private boolean isVisible; // whether the Circle appears on the screen
}
```
Fields: Common Data Types

- An **int** is a whole number (*i.e.*, without a fraction)
  - Legal int values: 123, -45900, 0
  - Bad (won't compile): 14.4  13,000,000  $500

- A **double** is a real number (*i.e.*, may contain fraction)
  - Legal double values: 1.0, 3.14159, -90, 0.125, 0
  - Bad: 13,000,000  $500

- A **char** is a single character (enclosed in single quotes)
  - Legal char values: 'a', '#', 'Z', '+', '3'
Fields: Common Data Types (Cont.)

- **A boolean** is a true or false value
  - The only values can be: `true`, `false`
  - Bad: `True`, `False`, `t`, `f`, `1`, `0`

- **A String** is a sequence of chars (like a word or phrase)
  - Legal String values: "Hello world", "red", "4.0", ""
  - Bad: `Hello world`, `'red'`, `4.0`

- **Down the road**: Every class we invent can also be a data type
  - Circle, Square, Triangle, ...
We can add as much to the object's state as want...

Let's have Circles also remember their areas

Things to consider:

• Need to add area to a circle's state. *(Deal with this now)*
• When does the value get assigned initially? *(Deal with this next)*
• Maintenance
  - Can't the Circle change size? *(Deal with this later)*
Basic Class Structure

(Possible import statements)

```java
public class ClassName {

  Fields (instance variables)  
  (`State`) 

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

  Methods 
  (`Behavior`) 

}
```
Objects have state, but what state do they *start out* with?

Classes can have a special initialization method called the *constructor*

- Lets the programmer prepare an object for use
- Has the same name as the class
- Recall Circle's instance variables...

```java
//recall these are my fields
private int diameter;
private int xPosition;
private int yPosition;
private String color;
private boolean isVisible;
```

**When someone creates a Circle...**
- *What should the default diameter be?*
- *Where should it be located?*
- *What should its default color be?*
- *Should it be visible?*
Constructor (Cont.)

- Constructors
  - Purpose: Initialize the default state of an object when it's created

- Constructor Syntax:
  ```
  public Class_name(parameter_list)
  {
    //code to initialize the state of a newly-created object
  }
  ```

- Know these:
  - Constructors are *always* named after the class
  - There must be *at least* one constructor defined per class
  - Parameter list may be empty (when that's the case, it's called a *default constructor*)
The *constructor* is called when a new object is instantiated

- Purpose is to give the new object with a startup state

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

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

    //(instance methods omitted)
}
```
Overloading Constructors

- Currently, Circle creation is very limited:
  - Why always blue? Why always 30 diameter? Why always at same location?

- Better: let the object's "creator" have a say!
  - Let's write a second constructor that lets users determine the size.
  - A third constructor that lets users determine size and color.
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;
    }
    (Makes a default circle)

    public Circle(int newSize) {
        diameter = newSize;
        xPosition = 20;
        yPosition = 60;
        color = "blue";
        isVisible = false;
    }
    (Inputs a diameter from user)

    public Circle(int newSize, String newColor) {
        diameter = newSize;
        xPosition = 20;
        yPosition = 60;
        color = newColor;
        isVisible = false;
    }
    (Inputs a diameter and color from user)
Self-Check

- Have Circles also remember their areas
  - Okay to use 3.14159 to approximate $\pi$

- Things to consider:
  - What data type is the area?
  - When does the value get assigned initially? *(Deal with this now)*
  - Maintenance
    - Can't the Circle change size?
    - Area should change when size changes!!
    - *(Deal with this too!)*

Area of Circle = $\pi r^2$
Basic Class Structure

(Possible import statements)

```java
public class ClassName {

    Fields (instance variables)  
    (State)

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

    Methods  
    (Behavior)

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

    //instance methods
    public void makeVisible() {
        isVisible = true;
        draw();
    }

    public void makeInvisible() {
        erase();
        isVisible = false;
    }

    //... more omitted

(1) Let's take a closer look at these two methods
A Close Look at the Method Signature

The **Method Signature** tells us:

- Name of the method
- What input(s) it requires, if any
- If it **returns** a result, what type of data it returns (**void** if no return)
- Its visibility to objects of other classes (assume **public** for now)

```java
public void makeVisible()
{
    isVisible = true;
    draw();
}
```

<table>
<thead>
<tr>
<th>Visibility</th>
<th>Return Data Type</th>
<th>Method Name</th>
<th>Parameter List (Empty in this case; doesn't require input)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>public</strong></td>
<td></td>
<td><strong>void makeVisible()</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Signature**

**Body** (contains the algorithm)
public void makeVisible()
{
    isVisible = true;
    draw();
}

Method body: within each method are a sequence of Java statements
What Happens When A Method Is Called?

(1) *Which method* is being called? Java identifies it by its *signature*!

```java
public void makeVisible()
{
    isVisible = true;
    draw();  //this calls/uses the draw() method! (Look down!)
}

private void draw()
{
    if (isVisible) {
        Canvas canvas = Canvas.getCanvas();
        canvas.draw(this, color, new Ellipse2D.Double(xPosition, yPosition, diameter, diameter));
        canvas.wait(10);
    }
}
```

*Denotes "control" (i.e., which instruction is Java executing now?)*
Another Self Check

- Write a method, `makeMonochrome()`, that turns the color of circle to black.

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

- Write a method, `takeALap(int distance)`, that causes the Circle to move through the progression of East, South, West, North, ending at the original spot. Each directional move is of the given distance.
  - *Return* the total distance traveled (What do we mean by "return?")
Outline

- What Are Objects?
  - State and Behavior
- What Are Classes?
  - How Are Objects Made?
- A Tour of BlueJ
  - Point-and-Click Elements
  - A Peek at the Source Code
    - Bossing around Objects without Menus (Using Code)
- Conclusion
Making Method Calls Using Dot Notation

- Telling objects what to do using the menu system is getting old...
  - We want to remove the human element in algorithm execution
  - Get Java to do it!

- Let's make pac-man (from Lab 1) again, but without using menus.
Make PacMan again Using CodePad

1. Create a new Circle named body
2. Create a new Circle named eye
3. Create a new Triangle named mouth
4. Make body visible
5. Make eye visible
6. Make mouth visible
7. Change body color to "yellow"
8. Change body size to 25
9. Change body size to 75
10. Move body down
11. Move body down
12. Move body up
13. Move body up
14. Move body up
15. Move body up
16. Change eye color to "black"
17. Change eye size to 5
18. Move eye horizontally by 58
19. Change mouth size to 75 (height) and 75 (width)
20. Move mouth horizontally by 10
21. Move mouth vertically by 40
22. Change mouth color to "white"
Important! Object Creation (Instantiation)

- Before you can manipulate objects, they must first be created
- Create the object by calling the class *constructor*
  - Code syntax:

  ```java
  ClassName nameOfObject = new ClassName(parameterValuesIfAny);
  ```

- Example:

  ```java
  1 Create a new Circle named body
  2 Create a new Circle named eye
  3 Create a new Triangle named mouth
  
  Circle body = new Circle();
  Circle eye = new Circle();
  Triangle mouth = new Triangle();
  ```
Make PacMan again Using CodePad

1 Create a new Circle named body
2 Create a new Circle named eye
3 Create a new Triangle named mouth
4 Make body visible
5 Make eye visible
6 Make mouth visible
7 Change body color to "yellow"
8 Change body size to 25
9 Change body size to 75
10 Move body down
11 Move body down
12 Move body up
13 Move body up
14 Move body up
15 Move body up
16 Change eye color to "black"
17 Change eye size to 5
18 Move eye horizontally by 58
19 Change mouth size to 75 (height) and 75 (width)
20 Move mouth horizontally by 10
21 Move mouth vertically by 40
22 Change mouth color to "white"

Circle body = new Circle();
Circle eye = new Circle();
Triangle mouth = new Triangle();
Important! Calling Methods

- After an object is created, you can call its methods!
  - Code syntax:
    
    ```
    nameOfObject.methodName(parameterValuesIfAny);
    ```
  
  - Example:
    ```
    // create my objects first (from before)
    Circle body = new Circle();
    Circle eye = new Circle();
    Triangle mouth = new Triangle();

    // draw the body on canvas
    body.makeVisible();
    body.changeColor("yellow");
    ```
Make PacMan again Using Code

1. Create a new Circle named body
2. Create a new Circle named eye
3. Create a new Triangle named mouth
4. Make body visible
5. Make eye visible
6. Make mouth visible
7. Change body color to "yellow"
8. Change body size to 25
9. Change body size to 75
10. Move body down
11. Move body down
12. Move body up
13. Move body up
14. Move body up
15. Move body up
16. Change eye color to "black"
17. Change eye size to 5
18. Move eye horizontally by 58
19. Change mouth size to 75 (height) and 75 (width)
20. Move mouth horizontally by 10
21. Move mouth vertically by 40
22. Change mouth color to "white"

```java
Circle body = new Circle();
Circle eye = new Circle();
Triangle mouth = new Triangle();
body.makeVisible();
eye.makeVisible();
mouth.makeVisible();
body.changeColor("yellow");
body.changeSize(25);
body.changeSize(75);
body.moveDown();
body.moveDown();
body.moveUp();
body.moveUp();
body.moveUp();
body.moveUp();
eye.changeColor("black");
eye.changeSize(5);
eye.moveHorizontal(58);
mouth.changeSize(75, 75);
mouth.moveHorizontal(10);
mouth.moveVertical(40);
mouth.changeColor("white");
```
Where to Type the Code?

- If just testing things out, want quick and dirty feedback:
  - Put it in the codepad!

- If you want to be able to make repeated calls to it:
  - Put it in a method, then call it whenever you want
  - Or, this method is special: the `main()` method
    
    - (next slide)
The `main()` method

- There's got to be a more "automatic" way than having to paste code in the Code Pad all the time.
- Create another class, we'll call it `CanvasTester`
The `main()` method (Cont.)

- There will be some starter code inside `CanvasTester`.
  - Erase the starter code and type in the following

- The `main()` method is special: it tells Java "Start running the program here!"

```java
public class CanvasTester {
    
    public static void main(String[] args) {
        Circle body = new Circle();
        Circle eye = new Circle();
        Triangle mouth = new Triangle();
        body.makeVisible();
        eye.makeVisible();
        mouth.makeVisible();
        // the remaining pacman producing code omitted
    }
}
```
Outline

- What Are Objects?
  - State and Behavior

- What Are Classes?
  - How Are Objects Made?

- A Tour of BlueJ
  - Point-and-Click Elements
  - A Peek at the Source Code
  - Bossing around Objects without Menus (Using Code)

- 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
CS tutoring starts today!
- 4-6pm everyday except Friday and Saturday

Reminder: Hwk 1 due Friday

Last time...
- Classes vs. Objects in Java
- Object state and behavior
- Started peeking at Circle class (and annotating it)
Hwk 1 extended to Monday!

Lab 2 Post-Mortem

• 3 types of variables: fields, local variables, input parameters
  - Fields: Object scope
  - Local vars and input parameters: Local scope
• Assignment statements are one-way operations
• What's up with integer divide?

Last time... Looked through Circle class

• Class structure, block/Javadoc & line comments, fields, constructors
• Today: Overloading constructors, methods, method calls
Administrivia 2/4

- Hwk 1 due tonight!!

- Last time... got further on the Circle class:
  - We have 3 Circle constructors now
  - We wrote takeLap() method
  - We talked about method return types
    - And wrote getArea()

- Today:
  - Finish Circle class
  - Start on TicketMachine