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
In software an **object** models real-world entities

- Circle
- Car
- Thermostat
- Color palette
- PacMan
- *and so on...*

Object usually have at least one **reference (name)** identifying it
Object State

- **Important:** Objects have both *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 both *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

- 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 (with Code)

- Conclusion
What does a *class* (or classification) mean?

- A group of things that share similar (e.g., state and behavior)

When we program in Java, we're only writing classes!

**Instances** of Birds (Objects)
Another Class Example: Circles

- Classes can group together anything we find commonalities in
- Another example:

```plaintext
Circle Class

Instances of circles (Objects)

State:
- diameter
- color
- xPosition
- yPosition
- isVisible

Behavior:
- moveUp()
- moveDown()
- changeColor()
- changeSize()
```
Outline

- What Are Objects?
  - State and Behavior

- A Peek at the Source Code
  - Comments
  - Fields and Data Types
  - Constructors
  - Methods

- Bossing around Objects without Menus (with Code)

- Conclusion
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)

}
```
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** 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:

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

- (Single) Line comment
  - Example:

```java
// Everything on this line is ignored by the compiler
```
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)}

}
```
Field declaration (A property that objects of this class remembers)

- Java syntax:

```java
private dataType fieldName;
```

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?
 Constructors

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

 Constructor Syntax:

```java
public ClassName(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 yPosition;
    private String color;
    private boolean isVisible;

    //default constructor
    public Circle() {
        this.diameter = 30;
        this.xPosition = 20;
        this.yPosition = 60;
        this.color = "blue";
        this.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.

```
Instantiate green circle
Instantiate black circle, size 10
Instantiate red circle, size 60
```
Overloading Constructors (Cont.)

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

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

    public Circle(int newSize) {
        this.diameter = newSize;
        this.xPosition = 20;
        this.yPosition = 60;
        this.color = "blue";
        this.isVisible = false;
    }

    public Circle(int newSize, String newColor) {
        this.diameter = newSize;
        this.xPosition = 20;
        this.yPosition = 60;
        this.color = newColor;
        this.isVisible = false;
    }
}
```

(Makes a default circle)

(Inputs a diameter from user)

(Inputs a diameter and color from user)
Self-Check

Now we should have Circles 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!)*
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() {
        this.isVisible = true;
        this.draw();
    }

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

    //... more omitted

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

- The *Method* syntax:
  - Name of the method
  - What input(s) it requires, if any
  - The body contains the algorithm itself
  - If the body *returns* a value, what data type? (*void* if no return)

Visibility | Return Data Type | Method Name | Parameter List (Empty in this case; doesn't require input)
---|---|---|---

```
public void makeVisible() {
    this.isVisible = true;
    this.draw();
}
```
Let's Write a New Method: getArea

- Write a new method, called `getArea` that...
  - Accepts no inputs
  - Doesn't change the state of the Circle at all
  - Returns the value of the `area` field to the caller

```ruby
xPosition: 0
yPosition: 0
color: "black"
isVisible: true
diameter: 30
area: 706.85775
```
Let's Write a New Method

- Write a new method, called **getArea** that...
  - Accepts no inputs
  - Simply **returns** the value of the **area** field to the caller

After calling: `getArea()`

```
xPosition: 0
yPosition: 0
color: "black"
isVisible: true
diameter: 30
area: 706.85775
```
Write a method, `makeMonochrome()`, that turns the color of circle to black.

- Accepts no inputs
- Returns nothing

```plaintext
xPosition: 0
yPosition: 0
color: "yellow"
isVisible: true
diameter: 30
area: 706.85775
```
Write a method, `makeMonochrome()`, that turns the color of circle to black.

- Accepts no inputs
- Returns nothing

After calling: `makeMonochrome()`

```
xPosition: 0
yPosition: 0
**color: "black"**
isVisible: true
diameter: 30
area: 706.85775
```
Group Work

- 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

![Diagram of a circle with arrows indicating movement in different directions and distances.](image)
On your own

- Write a method, `revertColor()`, that reverts the color back to its original color (i.e., when the circle was first created).
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 \{\ldots\}\}
    - 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
Administrivia 1/29

- CS tutoring starts today!
  - 4-6pm everyday except Friday and Saturday
  - (Remember to sign in)

- Last time...
  - Classes vs. Objects
  - Started peeking at Circle class (and annotating it)
    - Import statements
    - Comments
    - Curly braces { ... }
    - State (as "fields")
Hwk 1 due on Monday!

Lab 2 Post-Mortem

- 3 types of variables: fields, local variables, input parameters
  - Fields have "Object Scope" (accessed using this.var)
  - Local vars and input parameters have "Local scope"
  - Scope? Draw "bubbles"

Last time...

- Adding fields, overloading constructors
  - We added an "area" field
  - We added a new constructor that inputs a diameter
Hwk 1 due tonight!!
  • Hwk 2 and Lecture 2a have been posted!

Last time... got further on the Circle class:
  • We have 3 Circle constructors now
  • We wrote takeLap() method

Today:
  • Return types for methods, finish Circle class
  • Start on TicketMachine