Recap: *What* an object can do

- Defined by its *interface*
  - Consists of *public* methods and *public* data
- Accomplished by its *implementation*
  - Includes *private* members and *internal details* of methods
- A `class` provides both the interface and implementation for objects of a particular type
  - Defines the public interface
  - Defines the data that objects store
  - Implements the methods (both public and private)

`Label` objects for example

- Public interface – what clients need to know
  - Includes accessors: `public String getText()`
  - And mutators: `public void setText(String text)`
  - Even constants: `public static final int CENTER`
    - Also `LEFT` and `RIGHT` – where to display the text
- Implementation is in `class` (java.awt.)`Label`
  - Defines the public methods – so they actually work
  - Has non-public features too: `text`, `alignment`, …
    - Includes methods that clients don't have to know about
    - Reason: these parts can change without ruining client's work

A custom example: `BankAccount`

- A software designer identified the need for objects that represent bank accounts
  - Part of a banking system, or personal portfolio, or …
- Q: Why *objects*, not just numbers?
  - A: bank accounts are more complex than numbers
    - Include data (balance, account holder information, …)
    - And methods (controlled ways to deposit and withdraw, …)
- Idea is that other software objects will:
  - Create `new BankAccount` objects
  - Use the objects' public features to solve problems

Notes about choosing classes

- A class represents a concept from the problem domain
- Name for a class – a noun that describes the concept
  - e.g., geometric concepts: Point, Rectangle, Ellipse, …
  - Or real life concepts: BankAccount, CashRegister, …
- Lots of general types of concepts/classes:
  - e.g., Actors (end in -er, -or) – do some kinds of work for you
    - Scanner is a good example
    - Random is not (better name would be RandomNumberGenerator)
  - e.g., Utilities (like Math) – often just static methods/constants
  - e.g., Program starters – only have a main method
- Advice: don't turn actions into classes
  - e.g., Paycheck is better name than ComputePaycheck

Accessor and mutator methods

- Accessors – to allow access to private data
  - Usually call same as variable, or getVariable
    - e.g.,
```
private int var; …
public int getVar() { return var; }
```
  - Note: only if other classes *need* such access
- Mutators – to allow changes to private data
  - e.g., deposit and withdraw methods of BankAccount
  - Basic mutators are usually called "set" methods
```
public void setVar(int x) { var = x; }
```
  - Note: only if other classes *should* change the data, and only in ways that keep the object in a valid state

Notes about `this`

- `this` is an object reference – a constant an object uses to refer to itself ("me" better reflects the concept)
- e.g., print me: `System.out.print(this);`
- Often just an *implicit* reference: `calculate();`
  - Same as explicitly saying `this.calculate();`
  - Also the case for instant variables: `x` ↔ `this.x`
    - See ThisTest.java (Fig. 8.4, pp. 323-324)
- Has a special purpose for overloaded constructors
  - See Time2.java (Fig. 8.5, pp. 325-327)
- Has no meaning (so illegal to use) in a static context

Predicate methods

- Methods that return a `boolean` value
  - e.g., BankAccount enhancement:

```
public boolean isOverdrawn() {
    return balance < 0;
}
```

- Can simplify and clarify programs that use them
  `if (myAccount.isOverdrawn()) …`
- Lots of API examples
  - e.g., Scanner: `input.hasNextInt()`
  - e.g., Stack: `stack.isEmpty()`
  - e.g., Character: `Character.isDigit(aChar)`

Avoid "side effects" of methods

- Any *externally* observable data modification
- e.g., modifying an explicit parameter

```
void transfer(double amount, Account other){
   balance = balance - amount;
   other.balance = other.balance + amount;
}
```

- Unexpected output is another example
  - i.e., don't print unless that is the method's purpose
  - In fact, any printing at all might cause problems

```
public void printBalance() { // Not recommended
    System.out.println("Balance is $" + balance);
}
```

    - Now only works in English locale
    - Also relies on System.out – might not be available in GUI

Packages

- Uppermost level of Java modules
  - Used to bundle related classes – a good design idea
- Declare in each class – `package my.stuff;`
- Store all in same directory – `./my/stuff/`
- Must qualify class names to use them
  - Either explicitly each time name is used – `my.stuff.Thing`
  - Else `import my.stuff.Thing;`
  - Or `import my.stuff.*;` // to get all classes in package
- See text section 8.16 (and related Fig. 8.19 and Fig. 8.20)
- Package access – a.k.a. "friendly" – no access modifier

Applets – an alternate approach

- A way to run a program – but *not an application*
  - No `main` method necessary
- Need a subclass of Applet (or JApplet)
  - So: `class __ extends Applet` (or `extends JApplet`)
- Most web browsers know how to create a new applet, and how to use certain Applet methods
  - So, applets must be embedded in an html page
  - And, to be useful, they must include at least one of the methods the browser invokes (e.g., `paint`)

"Running" an Applet

- The applet is started by the web browser as soon as the web page (html file) is visited
- The html file (stands for hypertext markup language) — must have an applet tag in it:

```
<html> …
<applet code=AppletClassName.class
            width=### height=###>
</applet> <!-- needs a closing tag too -->
… </html>
```

FYI: a little more html

- All based on tags – which come in pairs
  - e.g., italics – "`a <i>stressed</i> word`" – would show on web page as "a *stressed* word"
  - Also underline - `<u>…</u>`, bold - `<b>…</b>`, subscript - `<sub>…</sub>`, and so on
  - Can nest like "`<b><u>ok</u>ay</b> then!`" shows up as "**<u>ok</u>ay** then!"
    - But wrong if not *nested*, like "`<b><u>…</b></u>`"
- Best kind of tags are hyperlinks
  - e.g., "`<a href=http://www.ucsb.edu>my school</a>`" shows up like "my school"
- See any of many web resources

Implementing a "simple" applet

- `import javax.swing.JApplet;` // mandatory
  - Also usually Graphics and Graphics2D and others
- Declare a class that extends JApplet:
  `public class RectangleApplet extends JApplet`
- Implement `paint` method (at least)
  - Same procedures as `paintComponent` for components
- Create an html file to load the applet in a web browser or the `appletviewer` (provided with JDK)
- e.g., RectangleApplet.java (see link on Slides page)

Notes on *rendering* text

- Actually necessary to "draw" the text at a specified location on the Graphics object
  - `g.drawString(aString, x, y)`
  - Uses current rendering context (e.g., color), and current text attributes (e.g., font)
- Font: a *face* name, a *style*, and a point *size*
  ```
  Font f = new Font("Serif", Font.BOLD, 24);
  g.setFont(f); // now drawString uses this font
  ```
- Note: often can just use a `JLabel` to show in adjacent component
  - Other text display components too – even Text objects

Various applet examples

- FontApplet –fonts, and text centering
- TicTacToe – converting units to pixels
  - Note: vertical axis increases downward – so must flip y coordinates if drawing typical graph
- ImageApplet – displaying/scaling images
- EggApplet – handling mouse events
- ColorSlider – slider (state-change) events

Note: all of these programs could have been applications instead. Don't need applets to have graphical features in programs – just to include the programs on a web page.

5JA done! Where to go from here?

- Much deeper computer science to study
  - 1st take CS 10 – if you still like it, take more
- Many other programming languages out there
  - Beginning C is part of Engineering 3 curriculum
  - C++, VisualBasic, C#, … at UC Extension, SBCC, and tech schools like SB Business College
  - But you *can* learn them by yourself now too!
    - For specifics: just get a book, and/or look for online tutorial
- Lots more Java techniques to learn about
  - Suggest starting with Java Tutorial – books, and online at http://java.sun.com/docs/books/tutorial/