Objects

So far this semester

- Computer Science is about algorithms, information, and efficient automation of process

- Programming languages provide a way to precisely describe both data and process

- All languages have a **syntax** and associated **semantics**
  - **PRIMITIVES** — built in things
  - **MEANS OF COMBINATION** — ways to combine things

  Ex. Java has primitive data types (**int**, **double**, **String**, **boolean**) and primitive operators (**+**, **-**, **<**, **==**, **&**, **...**)

- One goal of programming language design is to enable software construction in a way that feels **natural**
One way to simplify the design of a complex system is abstraction — hiding detail.

We've seen two kinds of abstraction so far:

naming abstraction — declaring a name for something (such as a variable) and then referring to it by name.

Ex. `int width;
    width = x + y;`

procedural abstraction — defining a method and then calling the method.

Ex. `double square(double x) {
    return x * x;
}

3 = double area = square(width);`

Important:
Abstraction involves definition (details known) and use (details hidden).
• Variables provide a place to store information
• Methods provide a way to describe process, ways to use information

Today:
Introduce data abstraction — extending the set of types available in a language by defining new types that make sense for a particular problem or application.

A data type consists of
• internal representation (how information is stored)
• operations (what you can do with that data type)
In an object-oriented language (like Java):

- Data abstraction is supported by defining new types of objects.

- **Object** = \frac{\text{instance variables} + \text{methods}}{\text{the internal representation, how data is stored in the object}}

- A definition of a type of object is called a **class**.

- Each object of a particular type is an **instance** of its class.
Example: The Color class

```
int getRed() { return red; }

Color brighter() { return new Color(red + 50, green, blue); }
```

Creating objects: call a constructor
Color c = new Color(200, 12, 50);
Color d = c.brighter(); ← calling a method on the object c refers to
<table>
<thead>
<tr>
<th>Method Name</th>
<th>Params</th>
<th>Mutates</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>constructor</td>
<td>String owner</td>
<td>save owner's name &amp; set balance to 0</td>
<td>reference to the object</td>
</tr>
<tr>
<td>getBalance</td>
<td></td>
<td></td>
<td>current balance</td>
</tr>
<tr>
<td>deposit</td>
<td>double amount</td>
<td>add amount to the balance</td>
<td>new balance</td>
</tr>
<tr>
<td>computeInterest</td>
<td>date?</td>
<td>update balance w/ interest</td>
<td></td>
</tr>
<tr>
<td>withdraw</td>
<td>amount</td>
<td>if sufficient funds</td>
<td>true if successful</td>
</tr>
<tr>
<td>transfer</td>
<td>amount, other account</td>
<td>subtract amount do the transfer</td>
<td>true if OK</td>
</tr>
</tbody>
</table>
boolean withdraw(double amount) {
    ...
    getBalance()
    return wasSuccessful;