Introduction to Objects

Talk to your neighbors:
What did you do for Halloween, either yesterday or this past weekend?

Music: Secrets – The Weeknd
Lecture Outline

• Announcements
• SearchEngine Recap
• OOP Review
• Example
• Abstraction
Announcements

• Programming Assignment 2 (P2) out
  - Due November 9\textsuperscript{th} by 11:59 PM
  - Which means... no assignment releasing on Friday!

• Quiz 0 & 1 grades... someday?

• Resubmission Cycle 3 (R3) form coming soon, due November 7\textsuperscript{th} by 11:59 PM
Lecture Outline

• Announcements

• SearchEngine Recap

• OOP Review

• Example

• Abstraction
searchEngine & Inverted Index

- An **inverted index** is a Mapping from possible query words to the set of documents that contain that word

  - Answers the question: “What documents contain the word ‘corgis’?”
Lecture Outline

• Announcements

• SearchEngine Recap

• OOP Review

• Example

• Abstraction
Object Oriented Programming (OOP)

• **Procedural programming**: Programs that perform their behavior as a series of steps to be carried out
  - Classes that do things

• **Object-oriented programming (OOP)**: Programs that perform their behavior as interactions between objects
  - Classes that represent things
  - We’re going to start writing our own objects!
Classes & Objects

• **Classes** can define the *template* for an object
  - Like the blueprint for a house!
  “What does it mean to be this thing?”

• **Objects** are the actual *instances* of the class
  - Like the actual house built from the blueprint!
  “It is an example of this thing!”

We create a new instance of a class with the **new** keyword e.g., `Scanner console = new Scanner(System.in);`
State & Behavior

• **Objects** can tie related *state* and *behavior* together

• **State** is defined by the object’s *fields* or *instance variables*
  - *Scanner*’s state may include what it’s scanning, where it is in the input, etc.

• **Behavior** is defined by the object’s *instance methods*
  - *Scanner*’s behavior includes “getting the next token and returning it as an *int*”, “returning whether there is a next token or not”, etc.
Syntax

```java
public class MyObject {
   // fields
   type1 fieldName1;
   type2 fieldName2;
   ...

   // instance methods
   public returnType methodName(...) {
      ...
   }
}
```
Instance Variables

- Fields are also referred to as **instance variables**
- Fields are defined in a class
- Each instance of the class has their own copy of the fields
  - Hence *instance* variable! It’s a variable tied to a specific instance of the class!
Instance Methods

• **Instance methods** are defined in a class
• Calling an instance method on a particular *instance* of the class will have effects on that instance
Lecture Outline

• Announcements

• SearchEngine Recap

• OOP Review

• Example

• Abstraction
Representing a Coordinate Point

How would we do this given what we knew last week?

Maybe `int x, int y`?

Maybe `int[]`?
Representing a point

\texttt{int x, int y}

\begin{itemize}
  \item Easy to mix up x, y
  \item Just two random ints floating around – easy to make mistakes!
\end{itemize}

\texttt{int[]}

\begin{itemize}
  \item Not really what an array is for
  \item Again, just two ints – just have to “trust” that we’ll remember to treat it like a point
\end{itemize}

\textbf{Let’s make a class instead!}
Instance Methods

• **Instance methods** are defined in a class

• Calling an instance method on a particular *instance* of the class will have effects on that instance
Instance Methods

• **Instance methods** are defined in a class

• Calling an instance method on a particular *instance* of the class will have effects on *that* instance
Lecture Outline

• Announcements
• SearchEngine Recap
• OOP Review
• Example
  • Abstraction
Abstraction

The separation of ideas from details, meaning that we can use something without knowing exactly how it works.

You were able use the Scanner class without understanding how it works internally!
Client v. Implementor

We have been the clients of many objects this quarter!

Now we will become the implementors of our own objects!