This week:

- more on object-oriented programming (OOP)
  - objects vs. primitive types
  - creating new objects with `new`
  - calling methods on an object
  - importing classes
- input/output
objects and primitive types (1)

- everything in Java is either
  - an object
  - a primitive type

- differences?
  - objects take up more memory
  - objects allocated in main computer memory (heap)
  - objects have methods
  - manipulating primitive types is faster
objects and primitive types (2)

- primitive types:
  - boolean, byte, char, int, long, float, double
  - char is actually 16-bits (Unicode)

- object types:
  - String, arrays, user-defined classes
classes

- classes
  - are a template for creating objects
  - define the types of the data (fields)
  - define the code that acts on the data (methods)
  - can define methods that don't act on data
    - static methods
    - like global functions in most languages
    - main() was an example of this
public class Point2d {
    private double xcoord;
    private double ycoord;
    // Constructor:
    public Point2d(double x, double y) {
        this.xcoord = x;
        this.ycoord = y;
    }
    // continued on next slide...
// no-argument constructor:
public Point2d() {
    this.xcoord = 0.0;
    this.ycoord = 0.0;
}

// accessors:
public double getX() { return xcoord; }
public double getY() { return ycoord; }
// not done yet...
example class (3)

```java
public void setX(double value) {
    xcoord = value;
}

public void setY(double value) {
    ycoord = value;
}

} // end of class Point2d
```
public and private

- what do public and private mean, anyway?
- private: can only be used by methods of this class
- public: can be used anywhere
- usually fields are private and methods are public (some exceptions)
- also protected (for a later lecture)
using the class

- Creating new objects of the class:
  
  ```java
  Point2d a = new Point2d(2.0, 3.0);
  Point2d b = new Point2d(2.0, 3.0);
  System.out.println(a == b);  // false!
  ```

- Calling methods:
  ```java
  a.setY(10.0);
  ```

- n.b. methods have direct access to all of an object's fields
input/output (I/O) (1)

- input/output is very powerful and flexible in java
- downside: simple tasks are much more complicated than you'd expect
- first need to import the java i/o classes:

```java
// import all i/o classes
import java.io.*;
// can now say "Foo" instead of "java.io.Foo"
```
input/output (I/O) (2)

- **printing:**
  - `System.out.println()`
    - prints a data value + a newline
  - `System.out.print()`
    - prints a data value without a newline
  - `+` is used for string concatenation (overloading)

// this prints "foo10":
System.out.println("foo" + 10);
input/output (I/O) (3)

- input is more complicated...
- need to know the following classes:
  - **InputStreamReader**: converts bytes to characters
  - **StreamTokenizer**: converts characters to "tokens"
    - a token is a string of characters separated by space
    - `nextToken()` method – gets next token
    - `nval` field (double) – contains the numeric value of the next token (if it has one)
input/output (I/O) (4)

- sample input code:

```java
// System.in is the terminal input
InputStreamReader isr = new InputStreamReader(System.in);
StreamTokenizer tokens = new StreamTokenizer(isr);

// Repeat for every token you need:
tokens.nextToken();

// token value is in tokens.sval (and also in tokens.nval if it's a number)
```
input/output (I/O) (5)

- `nextToken()` can throw an `IOException` if something goes wrong
- this is our first encounter with "exception handling" in java
- we'll deal with this in later lectures
- for now, have to change `main()` to be

```java
public static void main(String[] args) throws IOException {
    // ...
}
```
next week

- exception handling in depth
- more on OOP
- documentation (javadoc)