Python and Matlab have objects and classes. Strong-typing nature of Java changes how OO is done and how useful it is. Put aside your previous experience with OO (if any).

This lecture:

First: describe objects, demoing their creation and use.

Second: Show you a class definition and how it contains definitions of functions and procedures that appear in each object of the class.

Third: Talk about keyword null.

Fourth (if there is time). Show you a Java application, a class with a “static” procedure with a certain parameter.
Homework

1. Study material of this lecture.
2. Visit course website, click on Resources and then on Code Style Guidelines. Study
   3. Documentation
      3.1 Kinds of comments
      3.2 Don’t over-comment
      3.4 Method specifications
         3.4.1 Precondition and postcondition
3. Spend a few minutes perusing slides for lecture 3; bring them to lecture 3.
References to course text and JavaSummary.pptx

Objects: B.1 slide 10-16
Calling methods: B.2-B.3 slide 18
Class definition: B.5 slide 11

public, private: B.5 slide 11, 12
Indirect reference, aliasing: B.6 slide 17
Method declarations: B.7
Parameter vs argument: B.12-B.14 slide 14

Methods may have parameters
Method calls may have arguments

Text mentions fields of an object. We cover these in next lecture

Text uses value-producing method for function and void method for procedure.
Get used to terminology: function and procedure
Object is associated with a window on your computer monitor

Name of object, giving class name and its memory location (hexadecimal). Java creates name when it creates object

Object contains methods (functions and procedures), which can be called to operate on the object

Function: returns a value; call on it is an expression
Procedure: does not return a value; call is a statement to do something
Evaluation of new-expression creates an object

Evaluation of
\[
\text{new javax.swing.JFrame()}
\]
creates an object and gives as its value the name of the object

If evaluation creates this object, value of expression is

\[
\text{JFrame@25c7f37d}
\]

\[
2 + 3 + 4 = 9
\]
A class variable contains the name of an object

Type JFrame: Names of objects of class JFrame

```
h = new javax.swing.JFrame();
```

If evaluation of new-exp creates the object shown, name of object is stored in h

```
h JFrame@25c7f37d
```

Consequence: a class variable contains not an object but the name of an object. Objects are referenced indirectly.

```
JFrame@25c7f37d
hide() show()
setTitle(String) getTitle()
getX() getY() setLocation(int, int)
getWidth() getHeight() setSize(int,int)
...
```
A class variable contains the name of an object

If variable \( h \) contains the name of an object, you can call methods of the object using dot-notation:

Procedure calls: \( h\.\text{show}() \); \( h\.\text{setTitle}(\text{“this is a title”}) \);

Function calls: \( h\.\text{getX}() \); \( h\.\text{getX}() + h\.\text{getWidth}() \)

\begin{align*}
\text{h} & \quad \text{JFrame@25c7f37d} \\
\text{JFrame@25c7f37d} & \quad \text{JFrame}
\end{align*}

\text{JFrame@25c7f37d}

hide() show()
setTitle(String) getTitle()
getX() getY() setLocation(int, int)
getWidth() getHeight() setSize(int,int)
...
**Class definition**: Describes format of an object (instance) of the class.

```java
/** description of what the class is for */

public class C {
    declarations of methods (in any order)
}
```

This is a comment

Access modifier

`public` means `C` can be used anywhere

Class definition `C` goes in its own file named `C.java`

On your hard drive, have separate directory for each Java program you write; put all class definitions for program in that directory. You’ll see this when we demo Eclipse
First class definition

```java
/** An instance (object of the class) has (almost) no methods */
public class C {

}
```

Then, execution of
```
C k;
k = new C();
```
creates object shown to right and stores its name in k
Class extends (is a subclass of) JFrame

/** An instance is a subclass of JFrame */
public class C extends javax.swing.JFrame {

}

C: subclass of JFrame
JFrame: superclass of C
C inherits all methods that are in a JFrame

Object has 2 partitions:
one for JFrame methods,
one for C methods

C@6667f34e

hide()   show()
setTitle(String) getTitle()
getX()   getY()   setLocation(int, int)
getWidth() getHeight()   …

JFrame

C

Easy re-use of program part!
Class definition with a function definition

/** An instance is a subclass of JFrame with a function area */
public class C extends javax.swing.JFrame {
    /** Return area of window */
    public int area() {
        return getWidth() * getHeight();
    }
}

You know it is a function because it has a return type

Spec, as a comment

Function calls automatically call functions that are in the object

C@6667f34e

... getWidth() getHeight()

area()
Inside-out rule for finding declaration

/** An instance ... */

public class C extends javax.swing.JFrame {
    /** Return area of window */
    public int area() {
        return getWidth() * getHeight();
    }
}

To what declaration does a name refer? **Use inside-out rule:**
Look first in method body, starting from name and moving out; then look at parameters; then look outside method in the object.

The whole method is in the object

C@6667f34e
 getWidth() getHeight() ...
 area() {
     returngetWidth() * getHeight();
 }
Inside-out rule for finding declaration

```java
/** An instance ... */
public class C extends JFrame {
 /** Return area of window */
 public int area() {
   return getWidth() * getHeight();
 }
}
```

Function `area`: in each object, `getWidth()` calls function `getWidth` in the object in which it appears.
/** An instance is a JFrame with more methods */
public class C extends javax.swing.JFrame {
    public int area() {
        return getWidth() * getHeight();
    }
}

/** Set width of window to its height */
public void setWtoH() {
    setSize(getHeight(), getHeight());
}

Call on procedure setSize
It is a procedure because it has void instead of return type
Using an object of class Date

/** An instance is a JFrame with more methods */
public class C extends javax.swing.JFrame {
    ...
    /** Put the date and time in the title */
    public void setTitleToDateTime() {
        setTitle((new java.util.Date()).toString());
    }
}

An object of class java.util.Date contains the date and time at which it was created.
It has a function toString(), which yields the data as a String.
null denotes the absence of a name.

v2.getName() is a mistake! Program stops with a NullPointerException

You can write assignments like:  v1 = null;

and expressions like:  v1 == null
Hello World!

```java
/** A simple program that prints Hello, world! */
public class myClass {

    /** Called to start program. */
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```

`args` is an array of `String` elements

We explain `static` next week. **Briefly**: there is only one copy of procedure `main`, and it is not in any object