- Previous Lecture:
  - Selection statement
  - Iteration with **`while`** loop, **`for`** loop
  - **`static`** methods
- Today's Lecture:
  - More on methods
  - Scope of a variable
  - Intro to objects and classes
- Reading: Chapter 5 (Pay attention to Sec 5.2)

Calling a **`static`** method ...

... that is in a different class:

classname.*methodname*(...)

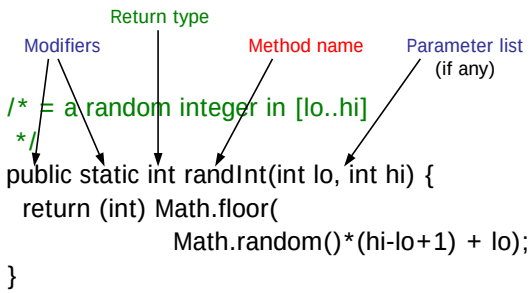
Examples: **`Math.random()`**
`Math.pow(2.5,2)`

... that is in the same class:

methodname(…)

Our class **`MyRandom`** has a **`static`** method **`randInt`**, so an example method call within the class can be

`randInt(3,8)`



Method

A method is a named, parameterized group of statements

modifiers return-type method-name **(** *parameter-list* **) {**
statement-list
}

return-type **`void`** means no value is returned from the method ⇒ no need for a return statement, but you can have one: **`return;`**

There must be a **`return`** statement, unless *return-type* is **`void`**

Method

A method is a named, parameterized group of statements

modifiers return-type method-name **(** *parameter-list* **) {**
statement-list
}

parameter-list : type-name pairs separated by commas

Example: **`int randInt(int lo, int hi)`**

A parameter is a variable that is declared in the method

Scope of a local variable

- A variable declared inside a method is a local variable. We say it is "local to the method."
- The scope of a variable is the "area" of the program in which the variable is recognized
- The scope of a local variable *starts at its declaration and ends with the block* in which the variable is declared
- Example: See method main in MyRandom

`Math` class

- A collection of common mathematical functions and constants
- **`static`** methods and constants
  - Belong to the class
  - An object is *not needed* to access **`static`** members of a class


```
import javax.swing.*;

public class MakeFrame {
  public static void main(String[] args){
    JFrame f= new JFrame();
    f.show();
    f.setSize(500,200);
    int w= f.getWidth();
    System.out.println("Width is " + w);
    f.setTitle("My new window");
    JFrame f2=new JFrame();//another one!
    f2.show();  f2.setSize(100,700);
  }
}
```

Notice these behaviors:

- We can have multiple JFrame objects
- We can access the individual JFrames by declaring a different name for each
- Each JFrame has its own states (e.g., width, height, title, position, etc.)
- To have JFrame f2 perform some action we call f2's method. E.g., f2.show()
- ⇨ Each object has its own variables and methods!

Pre-defined class `JFrame`

- Deals with windows (frames) on the monitor
- All the predefined classes are collectively called the Java API
- Classes are grouped into packages. E.g., java.io, java.net, javax.swing
- Use the `import` statement:

  `import javax.swing.*;`
- To find out what the classes do, read the API specifications:

  http://java.sun.com/j2se/1.5.0/docs/api

Object & Class—an analogy

- *Object*: a folder that stores information (data and instructions)
- *Class*: a drawer in a filing cabinet that holds folders of the same type

What is in an object?

(What is in a folder?)

- Fields to store data
- Instructions for dealing with the object