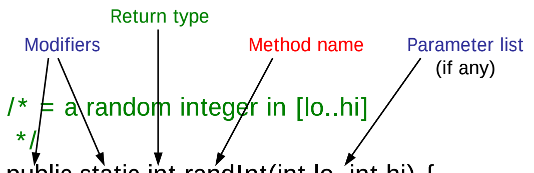
- Previous Lecture:
  - Selection statement
  - Iteration with `while` loop
  - `static` methods
- Today's Lecture:
  - Review methods (functions)
  - Iteration with `for` loop
  - Intro to objects and classes
- Reading: pp 136-145 of Sec 3.3, pp 36-42, pp 112-115



```
/* = a random integer in [lo..hi]
 */
public static int randInt(int lo, int hi) {
  return (int) Math.floor(
                  Math.random()*(hi-lo+1) + lo);
}
```

The `for` loop

Syntax:

for (*initialization***;** *condition***;** *update***)**

    statement-to-repeat;

- *Initialization*, *condition*, and *update* are not required, but the semi-colons (;) are required

The `for` loop

Syntax:

for (*initialization***;** *condition***;** *update***)**

    statement-to-repeat;

- *Initialization* is done once, before loop begins
- *condition* is evaluated
- Loop body executes only if *condition* evaluates to **`true`**
- *update* is executed

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!

What might class **`JFrame`** look like?

```
... class JFrame ... {

  // Variables to store the states (attributes) of the frame
     int width, height;  // Dimensions of the frame
     String title;       // Title of the frame
     ...                 // Other variables

  /* = width of this JFrame */
  public int getWidth() { ... }

  /* Set this JFrame's title */
  public void setTitle(String name) { ... }

  ... // Other methods
}
```

Pre-defined class **`JFrame`**

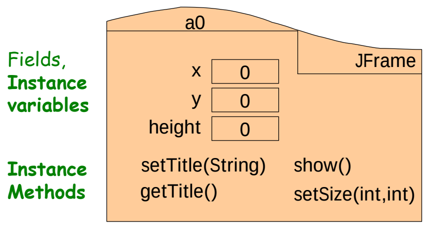
- Deals with windows (frames) on the monitor
- All the predefined classes are collectively called the Java API
- Classes are grouped into packages. E.g., java.io, java.net, javax.swing
- Use the `import` statement:
  `import javax.swing.*;`
- To find out what the classes do, read the API specifications: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

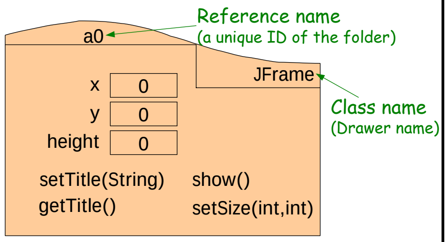
What is in an object?
(What is in a folder?)

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



What is in an object?
(What is in a folder?)

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



Creating an object

The expression

`new JFrame()`

- Creates a `JFrame` object (folder) and gives it a reference name
- Calls method `JFrame()` to set initial values for the object
- Yields the reference of the object

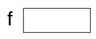
Reference variable

- Use a reference variable to hold on to an object:

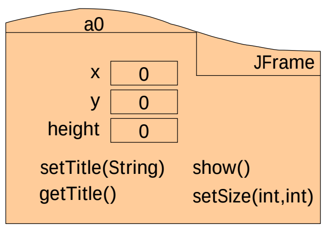
```
JFrame f= new JFrame();
```

Use the class name
as a type

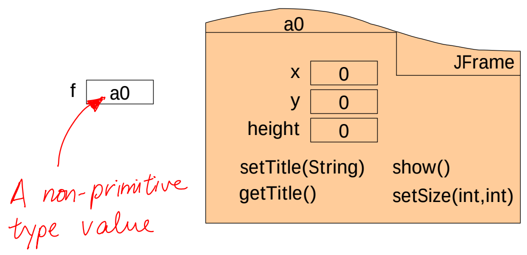
```
JFrame f;
```



```
new JFrame()
```



```
JFrame f = new JFrame();
```



Object & Class

- *Object*: contains variables (fields, instance variables) and methods
  - ***Variables***: "state" or "characteristics" e.g., name, age
  - ***Methods***: "behavior" or "action" e.g., yell, bounce
- *Class*: blueprint (definition) of an object
  - *No memory space* is reserved for object data
- An object is an instance of a class

Calling instance methods

```
JFrame f= new JFrame();

f.show();
f.setSize(600,200);
int w = f.getWidth();
```

Syntax :
referenceVariableName **.** *methodName* (*arguments*)

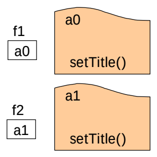
Accessing a field

Syntax:

referenceVariableName **.** *fieldName*

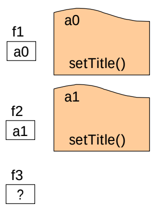
Instance methods are accessed through the instance

```
JFrame f1= new JFrame();

JFrame f2= new JFrame();
```



Reference ≠ Object

```
JFrame f1= new JFrame();

JFrame f2= new JFrame();

JFrame f3; //local variable
           //no default value
```



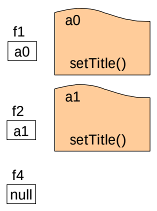
null

```
JFrame f1= new JFrame();

JFrame f2= new JFrame();

JFrame f4= null;
```

~~f4.setTitle("x");~~

A non-primitive type value

null means the reference variable does not refer to an object.


f1
a0
a0
setTitle()
f2
a1
a1
setTitle()
f4
null