- Previous Lecture:
  - `static` methods
  - Experiment with `JFrame` objects
- Today's Lecture:
  - Intro to objects and classes
  - Creating objects and calling their methods
  - OO thinking
- Reading: pp 36-42, 112-115, start reading Sec 4.1

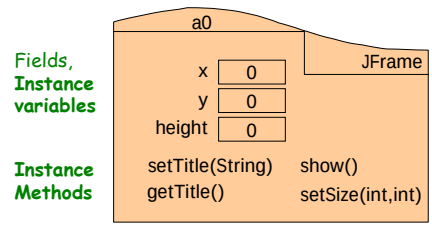

```
import javax.swing.*;

public class MakeFrame {
  public static void main(String[] args)
  {
     JFrame f= new JFrame();
     f.show();
     f.setSize(600,200);
     int w= f.getWidth();
     System.out.println("Width is " + w);
     f.setTitle("My window");
  }
}
```

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

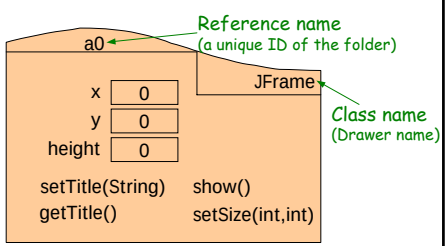
What is in an object?
(What is in a folder?)

- 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();
```

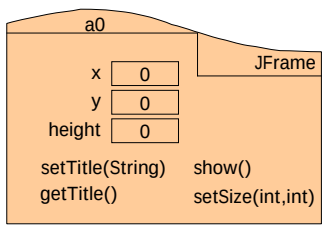
Use the class name as a type:

a non-primitive type

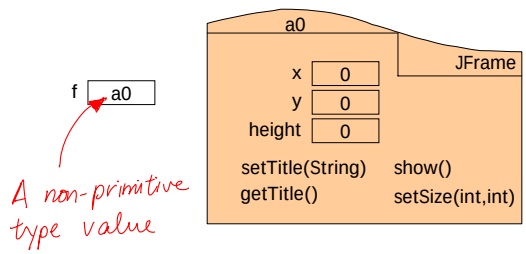

```
JFrame f;
```

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:

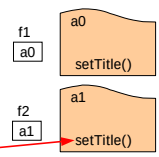
referenceVariableName . *fieldName*

```
import javax.swing.*;

public class MakeFrame {
  public static void main(String[] args)
  {
     JFrame f= new JFrame();
     f.show();
     f.setSize(600,200);
     int w= f.getWidth();
     System.out.println("Width is " + w);
  }
}
```

Local variables: "Live and die" inside a block (in a method)

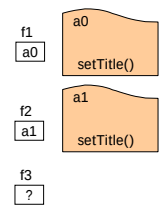
Instance methods are accessed through the instance

```
JFrame f1= new JFrame();

JFrame f2= new JFrame();

f2.setTitle("x");
```



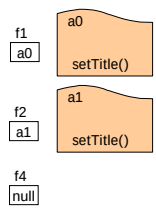
Reference ≠ Object

```
JFrame f1= new JFrame();

JFrame f2= new JFrame();

JFrame f3;

f3.setTitle("x");
```



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.

Primitive vs non-primitive values

```
int x= 2;
int y= 2;
JFrame f1= new JFrame();
JFrame f2= new JFrame();
JFrame f3= f1;
```

alias

f3==f1 gives

Class *definition* vs. object *instantiation*

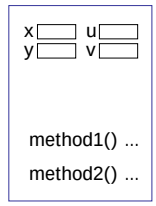
If you want make a whole lot of cookies, you may want to

- Make a cookie cutter—*define the class*
- Stamp out the cookie—*instantiate an object*

Making a cookie cutter
≠
Getting a cookie

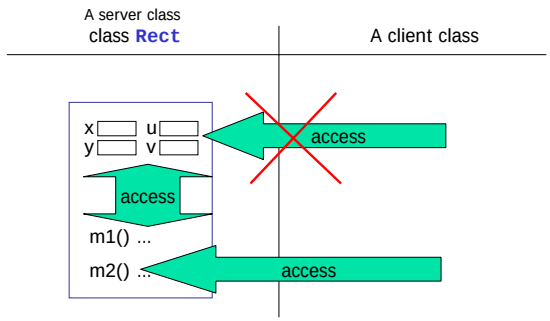

```
class Rect {

  // attributes
    private double left;
    private double right;
    ...

  // drawRect method
     ...
  // area method
     ...
  // perimeter method
     ...
}
```

Object from class **Rect**



OOP ideas

- Aggregate variables/methods into an abstraction (a class) that makes their relationship to one another explicit
- Objects (instances of a class) are self-governing (protect and manage themselves)
- Hide details from client, and restrict client's use of the services
- Allow clients to create/get as many objects as they want



Data within objects should be protected: **`private`**
Provide only a set of methods for **`public`** access.

```
class Rect {

  // attributes
    private double left;
    private double right;
    ...

  // drawRect method
     ...
  // area method
     ...
  // perimeter method
     ...
}
```

Server class

```
public class UseRect {

  public static void main
   (String[] args) {

    // create a rect
      Rect r1 = new Rect(...);
    // calculation on r1
      r1.area()

    // create another rect
      Rect r2 = new Rect(...);
      r2.drawRect()
  }
}
```

Client class

- We have used different classes already:
  - **`System`**, **`Math`**
  - **`Keyboard`**, **`JFrame`**
- Above classes provide various *services* (related services are grouped in same class)
- Implementation details of the class are hidden from the *client* (user)