- Previous Lecture:
  - Review methods (functions)
  - Scope of a local variable
  - Static variables
  - Intro to objects and classes
- Today's Lecture:
  - Intro to objects and classes
  - Creating objects and calling their methods
  - OO thinking
- Reading: Sec 6.1, 6.6


```
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:

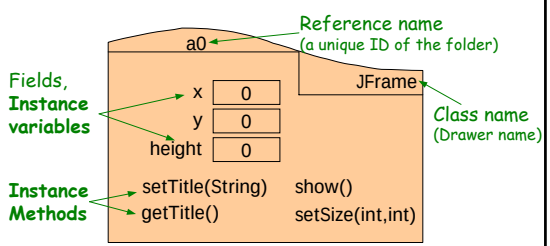
- We can have multiple JFrame objects
- We can access the individual JFrames by declaring a different name for each
- Each JFrame has its own states (e.g., width, height, title, position, etc.)
- To have JFrame f2 perform some action we call f2's method. E.g., f2.show()
- ⇨ Each object has its own variables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

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



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

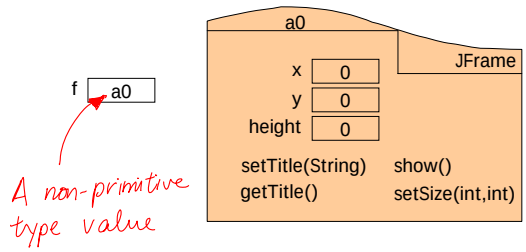
A reference variable "points to" an object

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

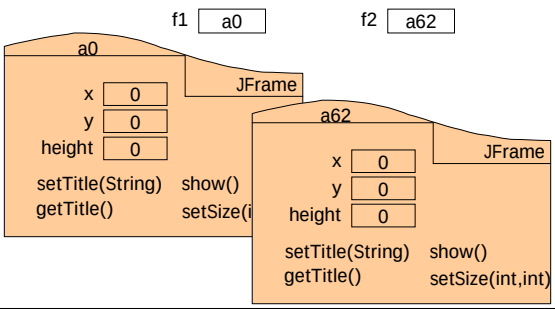
```
JFrame f= new JFrame();
```

Use the class name as the type name


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




```
JFrame f1 = new JFrame();
JFrame f2 = new JFrame();
```



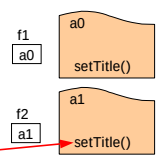
Object

- *Object*: contains variables (fields, instance variables) and methods (instance methods)
  - ***Variables***: "state" or "characteristics" e.g., dimensions, title, positions,
  - ***Methods***: "behavior" or "action" e.g., show (draw), setTitle, getWidth

How to access the methods (and variable values) in an object?

Instance methods are accessed through the instance

```
JFrame f1= new JFrame();

JFrame f2= new JFrame();

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



Calling instance methods

```
JFrame f= new JFrame();

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

Syntax :

referenceVariableName . *methodName* (*arguments*)

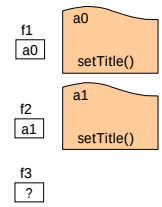
Accessing a field (instance variable)

Syntax:

referenceVariableName . *fieldName*

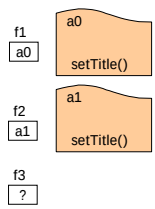
Reference ≠ Object

```
JFrame f1= new JFrame();

JFrame f2= new JFrame();

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



Reference ≠ Object

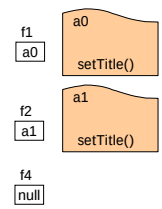
```
JFrame f1= new JFrame();

JFrame f2= new JFrame();

JFrame f3;

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



f1, f2, f3 are reference variables—not objects. A reference variable *may* store a reference to an object.

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

x==y gives

f1==f2 gives

f1==f3 gives

Class *definition* vs. object *instantiation*

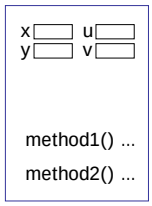
If you want make a whole lot of cookies, you may want to

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

Making a cookie cutter
≠
Getting a cookie

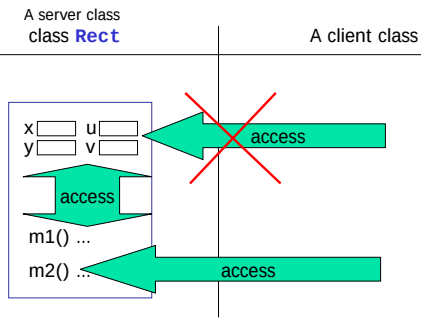
```
class Rect {

  // 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, Scanner**
  - **JFrame**
- Above classes provide various *services* (related services are grouped in same class)
- Implementation details of the class are hidden from the *client* (user)

Object & Class

- *Object*: contains variables (fields, instance variables) and methods (instance methods)
  - ***Variables***: "state" or "characteristics" e.g., dimensions, title, positions
  - ***Methods***: "behavior" or "action" e.g., show (draw), setTitle, getWidth
- *Class*: blueprint (definition) of an object
  - *No memory space* is reserved for object data
- An object is an instance of a class