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.
Reference variable

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

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

- Use the class name as a type:
  - A non-primitive type

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

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

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

Syntax:

```java
referenceVariableName . methodName (arguments )
```
Accessing a field

Syntax:
referenceVariableName . fieldName

Example:
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);
    }
}

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");

Primitive vs non-primitive values

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

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

A non-primitive type value

Local variables: "Live and die" inside a block (in a method)
**Class definition** vs. object **instantiation**

If you want to 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**

---

**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.

---

**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)**