Announcements:
- Project 4 due Sunday 4/3 at 6pm
- Use Keyboard class for reading input
- Section in classrooms this week

Previous Lecture:
- Selection statement
- Reading input using Keyboard class
- Iteration with while loop

Today’s Lecture:
- Object and class
- Creating objects and calling their methods

Reading:
- JV: Sec 3.8, Sec 4.1, 4.2

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:
  ```java
  import javax.swing.*;
  ```
- To find out what the classes do, read the API specifications:
  http://java.sun.com/j2se/1.4.2/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
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
### 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
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);
    }
}
```

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

### Accessing a field

Syntax:
```
referenceVariableName . fieldName
```

### 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)

**Local variables:** "Live and die" inside a block (in a method)
Instance methods are accessed through the instance

```java
JFrame f1 = new JFrame();
JFrame f2 = new JFrame();
setTitle("x");
```

Whose setTitle method???

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

Reference ≠ Object

```java
JFrame f1 = new JFrame();
JFrame f2 = new JFrame();
JFrame f3; // local variable
// no default value
f3.setTitle("x");
```

null

```java
JFrame f1 = new JFrame();
JFrame f2 = new JFrame();
JFrame f4 = null;
f4.setTitle("x");
```

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

Reference ≠ Object

```java
JFrame f1 = new JFrame();
JFrame f2 = new JFrame();
JFrame f3;
JFrame f3;
```

Primitive vs non-primitive values

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

Expression \( x == y \) gives
### Primitive vs non-primitive values

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

Expression f1 == f2 gives false
```

### 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

### Prelim 2

- Q1: “Tracing” code (2-d 😊, 1-d 😊, function 😊)
- Q2: short fragments (vector index expression, calling f with 2 return values, vectorized code 😊)
- Q3: vector, loop, algorithm to find min 😊
- Q4: nested loop 😊
- Q5: algorithm to sum triangle areas 😊

### Prelim 2

- Median 90
- Mean 84.1; Standard Deviation 16.0
- A rough mapping of grades:
  - Q (A-like) 90-100
  - R (B-like) 75-85
  - S (C-like) 60-70