Previous Lecture:
- while loop
- Object and class
- Creating objects and calling their methods

Today’s Lecture:
- Calling instance methods

Assigned reading:
- GG Sec 2.1-2.3, Sec 3.1.0, 3.1.1

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);
    }
}

What we wrote in DrJava’s interaction pane

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

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

Instance methods are accessed through the instance

Primitive vs non-primitive values
int x = 2;
int y = 2;
JFrame f1 = new JFrame();
JFrame f2 = new JFrame();

Expression x==y gives
Expression f1==f2 gives
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

Class Definition

```java
public class class-name {
    // declaration (and initialization)
    constructor
    methods
}
```

Class definition: declarations

```java
class Interval {
    private double base; // low end
    private double width; // interval width
}
```

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.
Declarations Revisited
- Syntax: `type name;`
- Examples:
  ```java
  int count;
  Interval in1;
  Interval in2;
  ```
- Instance variables have default initial values
  - `int` variables: 0
  - Non-primitive (reference) variables: null
- Value null signifies that no object is referenced

Object instantiation
- An expression of the form `new class-name()` computes a reference to a newly created object of the given class
- Examples:
  ```java
  Interval in1;          //declaration
  in1 = new Interval();  //instantiation
  //Combined declaration & instantiation
  Interval in2 = new Interval();
  ```

Class definition: methods
```java
class Interval {
    private double base;  // low end
    private double width; // interval width

    /* =Get right end of interval */
    public double getEnd() {
        return base + width;
    }
}
```

Calling an instance method
```java
public class Client {
    public static void main(String[] args){
        Interval in1;
        in1 = new Interval();
        double x;
        x = in1.getEnd();
    }
}
```