Announcements:
- Project 4 due Sunday 4/3 at 6pm
- Use Keyboard class for reading input

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

Today’s Lecture:
- Defining a class:
  - Instance variables
  - Instance methods
  - Constructors
- Reading:
  - JV: Sec 4.3, 4.4

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

Object from class Rect

public class UseRect {
  public static void main (String[] args) {
    // attributes
    private double left;
    private double right;
    ...
    // drawRect method
    ...
    // area method
    ...
    // perimeter method
    ...
  }
}

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

class Rect {
  // attributes
  private double left;
  private double right;
  ...
  // drawRect method
  ...
  // area method
  ...
  // perimeter method
  ...
}

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()
  }
}
Class Definition

class class-name {
    declaration (and initialization)
    constructor
    methods
}

Class definition: declarations

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

- Declarations in a class define fields (instance variables) of the class
- Each class is a type. Classes are not primitive types.

Declarations Revisited

- Syntax: type name;
- Examples: 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();
  ```

Objects are referenced

```java
public class Client {
    public static void main(String[] args){
        Interval in1;
        in1 = new Interval();
        // Memory diagram
    }
}
```

Do not access fields directly

```java
public class Client {
    public static void main(String[] args){
        Interval in1;
        in1 = new Interval();
        System.out.println(in1.base+in1.width);
    }
}
```

Class definition

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

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

A method is a named, parameterized group of statements:

```java
modifier return-type method-name ( parameter-list ) {
    statement-list
}
```

- **Return type**
- **Modifier**
- **Method name**
- **Parameter list** (if any)

### Example

```java
public double getEnd() {
    return base + width;
}
```

**Return-type void** means nothing is returned from the method.

There must be a `return` statement, unless return-type is `void`.

## Class definition

```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();
    }
}
```
Calling an instance method

public class Client {
    public static void main(String[] args) {
        Interval in1;
        in1 = new Interval();
        double x;
        x = in1.base + in1.width;
    }
}

Calling an instance method

public class Client {
    public static void main(String[] args) {
        Interval in1;
        in1 = new Interval();
        double x;
        x = in1.getEnd();
        in1.setWidth();
    }
}

Constructor

- A constructor is used to create objects
- Each class has a default constructor
- Can define your own constructor:

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

    /* Default constructor */
    public Interval() {} // no arguments

    /* An Interval with base b, width w */
    public Interval(double b, double w) {
        base = b;
        width = w;
    }

    public double getEnd() {
        return base + width;
    }
}
```

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

    /* Default constructor */
    public Interval() {} // no arguments

    public double getEnd() {
        return base + width;
    }
}
```
Constructor invocation

\texttt{new class-name\( (\text{expression-list}) \)}

- The value of above expression is a reference to a \textit{new} object of the given \textit{class-name}

- The defined (or default) constructor is invoked on the new object created by \texttt{new}

Creating an object

```java
public class Client {
    public static void main(String[] args){
        Interval in1;
        in1 = new Interval(0.5, 1);
    }
}
```

```java
public Interval(double b, double w) {
    base = b;
    width = w;
}
```

![Diagram of object creation and constructor invocation]