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

Variables
TWO main types of variables:
- Primitive type
- Reference to object

Some variables with different properties:
- Local: live and die inside a method
- Instance variable: owned by and accessed through individual instances (objects)
- Static variable: class variable shared by all instances—only one copy in a class

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

Class definition: declarations
class Interval {
    private double base; // low end
    private double range; // interval range
}

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

- Syntax: \texttt{type name;}
- Examples:
  
  \begin{verbatim}
  int count;
  Interval in1;
  Interval in2;
  \end{verbatim}

- Instance variables have default initial values
  - int variables: \texttt{0}
  - Non-primitive (reference) variables: \texttt{null}
- Value \texttt{null} signifies that no object is referenced

Object instantiation

- An expression of the form
  
  \begin{verbatim}
  new class-name()
  \end{verbatim}
  
  computes a reference to a newly created object of the given class

- Examples:
  
  \begin{verbatim}
  Interval in1; //declaration
  in1 = new Interval(); //instantiation
  //Combined declaration & instantiation
  Interval in2 = new Interval();
  \end{verbatim}

Objects are \textit{referenced}

\begin{verbatim}
public class Client {
    public static void main(String[] args){
        Interval in1;
        in1 = new Interval();
    }
}
\end{verbatim}

Manipulating references

\begin{verbatim}
public class Client {
    public static void main(String[] args){
        Interval in1;
        in1 = new Interval();
        Interval in2;
        in2 = in1;
    }
}
\end{verbatim}

Class definition: methods

\begin{verbatim}
class Interval {
    private double base; // low end
    private double range; // interval range

    // Getter method
    public double getEnd() {
        return base + range;
    }
}
\end{verbatim}
Methods

A method is a named, parameterized group of statements

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

 statement-list

 }
```

Return-type void means nothing is returned from the method

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

Class definition:

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

    // Getter methods
    public double getEnd() {
        return base + range;
    }
    public double getBase() { return base; }
}
```

Class definition:

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

    // Getter methods
    public double getEnd() {
        return base + range;
    }
    public double getBase() { return base; }
}
```

Calling an instance method

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

```
Interval in1 = new Interval();
Interval in2 = new Interval();
if ( in1.getEnd() > in2.getEnd() )
    System.out.println("blah...");
```