

News

Last Day

- how to write good programs, debugging

Today

- objects

Objects

- way to 'collect' related data into one 'container'
- includes a collection of *methods* to 'operate' on the data

Defining Objects

- define related variables, methods inside a class
- class is like glue, or a container, holding things together
  - class acts as a *template* or *blueprint* for objects
- example of a simple object
  - save this in a file "Complex.java"

```
class Complex
{
    public double real;
    public double imag;
}
```

- this declares double variables *real*, *imag* inside the class
- for now, these are declared *public* (*private* on Friday)
- can include any number, type of variables inside the class...
  - even other objects!

Objects and References

- example of *USING* simple object
  - from previous slide, in "Complex.java"
 

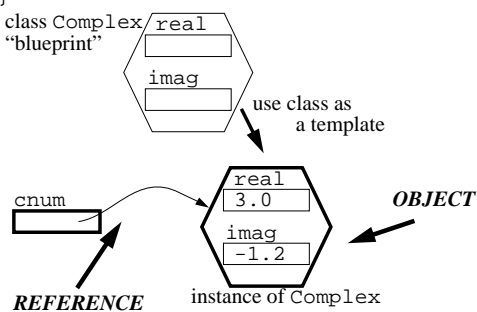
```
class Complex
{
    public double real;    // not static!!
    public double imag;   // not static!!
}
```

```
class ComplexTest
{
    public static void main(String[] args)
    {
        Complex cnum;
        cnum = new Complex();
        cnum.real = 3.0;
        cnum.imag = -1.2;
        System.out.print(cnum.real + "+" );
        System.out.print(cnum.imag + ")*i");
    }
}
```

- the *Complex* class is a *blueprint* for *Complex* objects
- *Complex cnum;*
  - a variable called *cnum*
  - it is a "*reference*" to an object of type *Complex*
  - initial value is "null"
- *cnum = new Complex();*
  - asks for a *new "instance"* or copy of a *Complex* object
  - *cnum* now *refers* to this object

Using Objects

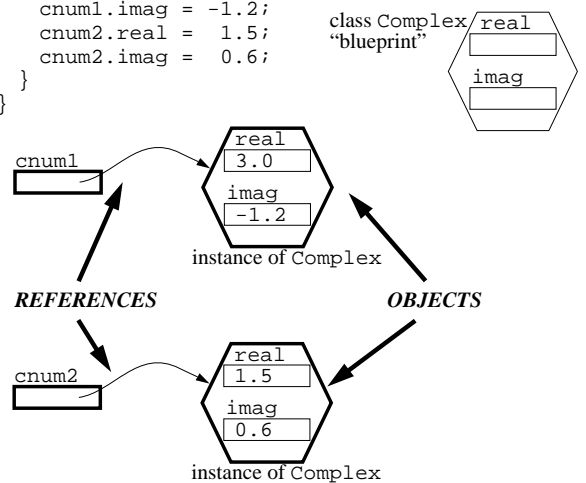
```
class ComplexTest
{
    public static void main(String[] args)
    {
        Complex cnum;
        cnum = new Complex();
        cnum.real = 3.0;
        cnum.imag = -1.2;
        System.out.print(cnum.real + "+" );
        System.out.print(cnum.imag + ")*i");
    }
}
```



- *cnum.real = 2.0;*
  - assigns value 2.0 to the *real field* or *member* of *cnum*
  - access *instance variables* like this:
    - *object-name.variable-name = 3;*
    - **note:** don't use *class-name* (i.e., *Complex*)

Using Objects

```
class ComplexTest
{
    public static void main(String[] args)
    {
        Complex cnum1 = new Complex();
        Complex cnum2 = new Complex();
        cnum1.real = 3.0;
        cnum1.imag = -1.2;
        cnum2.real = 1.5;
        cnum2.imag = 0.6;
    }
}
```



## Using Objects and Methods

- usually want to do interesting things to objects
  - objects have properties, a function of their data
    - complex numbers: magnitude, angle (polar coordinates)
    - circles: circumference, area
  - objects can be changed
    - complex numbers: add two of them
    - circles: scale the radius

- example: save in "Circle.java"

```
class Circle
{
    public double r;          // not static!!

    public double area()     // not static!!
    { return Math.PI * r * r; }

    public void scaleRadius(double factor)
    { r *= factor; }
}
```

- save in a file "CircleTest.java"

```
class CircleTest
{
    public static void main(String[] args)
    {
        Circle circ = new Circle();
        circ.r = 3.0;
        System.out.print( circ.area() );
        circ.scaleRadius(2.0);
        System.out.print( circ.area() );
    }
}
```

## Objects and Methods

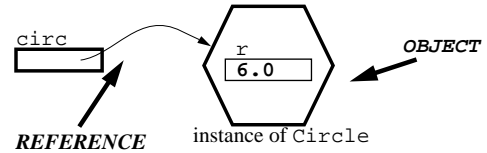
- refer to methods of objects as:
 

```
object-name.method-name(parameter-list);
```

Example:

```
circ.scaleRadius(2.0); // no result
```

- applies the scaleRadius() method to circ... result:



```
double area = circ.area();
//uses result of area() method
```

- applies the area() method to the circle... returns result:
 

```
(6.0*6.0*Math.PI) = 113.0973...
```

### Notice

- circ.area() has no parameters...
  - how does it know what the radius is????
  - it knows about the object referred to by circ
  - accesses radius of this circle as just r
- circ.scaleRadius(double factor) parameter...
  - inside method, can access the variable factor
  - also accesses radius of this circle as just r

## Constructors

- suppose we want to give initial values to real, imag inside Complex class

```
class Complex
{
    public double real, imag;

    // the following is a constructor
    public Complex( double r, double i )
    {
        real = r;
        imag = i;
    }

    // the following is a method
    public void addToReal( double amount )
    {
        real += amount;
    }
}
```

- use the **constructor** to give initial values!
  - do this inside ComplexTest:
 

```
cnum = new Complex( 3.0, -1.2 );
```
- notice how parameters match up with constructor parameters
- constructor has
  - same name as the class-name
  - no return type (not even void)
  - optional parameter list
- can have as many constructors, methods as you need...
  - but each one must have different parameter list!!!

## Bigger Test

```
class ComplexTest
{
    public static void main(String[] args)
    {
        Complex cnum1, cnum2;
        cnum1 = new Complex(3.0, -1.2);
        cnum2 = new Complex(1.5, 0.6);
        cnum1.addToReal(1.0);
        cnum2.addToReal(2.0);
    }
}
```

After running:

