

Last Day

- objects and methods

Today

- more objects: constructors, classes as data type, static

Objects

Important words from last day...

- **class** — collection of variables and methods, object blueprint
- **object** — a class (sometimes)
 - an instance of a class (usually)
- **instance** — a live/unique copy of a class, an object
- **member** — a method or variable inside a class
- **field** — a variable inside a class
- **reference** — a “pointer” to an instance
- **constructor** — a special method for initializing a new object

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

- saved in “Complex.java”, run javac to get “Complex.class”
- creates a new class called Complex
- can now use Complex as a type
 - just like int, double, char, ...
 - use it anywhere you may use int, double, char, ...

Constructor Concepts

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

    // The following is default constructor
    // it takes NO parameters.
    // We don't have to write one if we
    // don't want to do anything special,
    // Java will provide one automagically
    public Complex()
    {
        real = 0.0;
        imag = 0.0;
    }

    // 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;
    }
}
```

- real, imag are called the **instance variables**
- non-static methods inside Complex access by name

Constructor Concepts

- creating a new object is a two-step process:

STEP 1

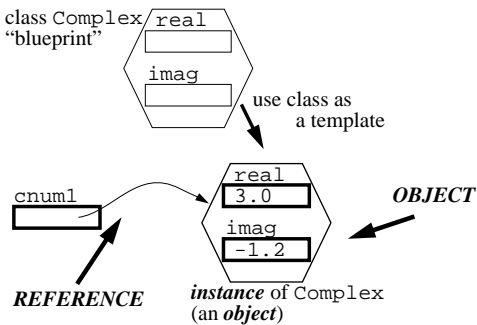
- ```
Complex cnum1;
```
- creates a **reference** to an object of type Complex
  - initial value of cnum1 is null, for “no object”
  - each reference uses memory (32 bits? 64 bits?)

STEP 2

- ```
cnum1 = new Complex();
cnum1 = new Complex(3.0, -1.2);
```
- creates a **new instance** of the Complex class
 - each instance uses memory (amount depends on object)
 - creation of instance calls proper constructor method

STEPS 1 AND 2

- ```
Complex cnum1 = new Complex(3.0, -1.2);
```
- combines both steps into one

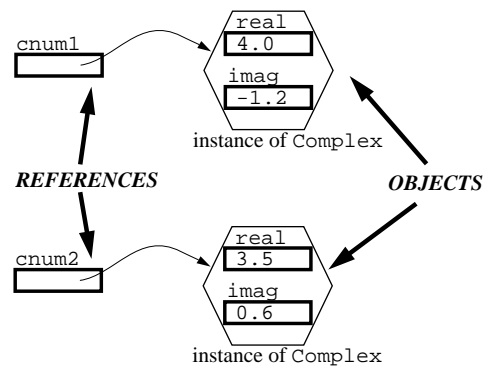


**bold rectangles:** need memory to store information!

**Constructor and Method 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:



## How to Add Complex Numbers

```
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);
 // cannot do:
 cnum1 = cnum1 + cnum2; // error!!!
 }
}
```

- **cannot** +, -, \*, / **references to objects!!!**
  - can only use +, -, \*, / on int, double, ...
- then how can we do this??
  - need a method to add Complex objects
  - place method inside Complex class
- new concept
  - can use Complex as a data type
  - just like int, double, ...
  - use a Complex type for the *parameter of the method*

```
public void add(Complex otherObj)
{ ...method body... }
```

- then to add one Complex object to another we can do  
cnum1.add( cnum2 );

## Using Complex as a data type

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

 // the following is default constructor
 public Complex()
 {
 real = 0.0;
 imag = 0.0;
 }

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

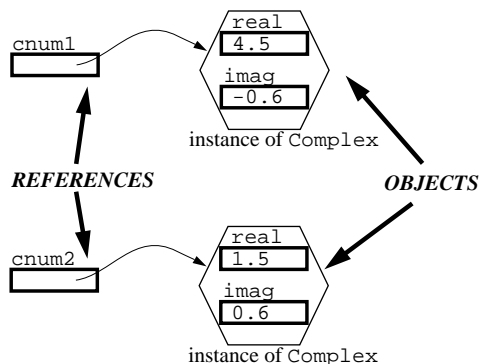
 // method to add complex numbers
 public void add(Complex otherObj)
 {
 real = real + otherObj.real;
 imag = imag + otherObj.imag;
 }
}
```

- here I have written a method that can be specified like this:  
public void add(Complex)
- in assignment 4 you must write methods like  
public void add(Rational)

## Using Complex as a data type

```
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);
 // cannot do:
 // cnum1 = cnum1 + cnum2;
 cnum1.add(cnum2);
 }
}
```

After running:



## Using Complex as a data type

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

 public Complex()
 { real = 0.0;
 imag = 0.0;
 }

 public Complex(double r, double i)
 { real = r;
 imag = i;
 }

 public Complex(Complex otherObj)
 { real = otherObj.real;
 imag = otherObj.imag;
 }

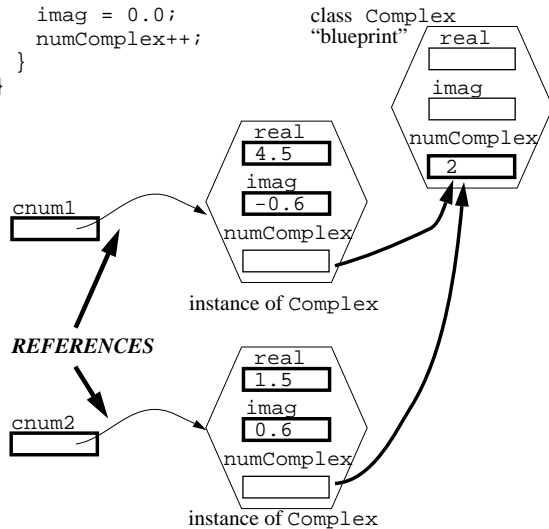
 // method to add complex numbers
 public void add(Complex otherObj)
 { real = real + otherObj.real;
 imag = imag + otherObj.imag;
 }
}
```

- inside main:  
Complex cnum1 = new Complex(3.0,-1.2);  
Complex cnum2 = new Complex( cnum1 );
- creates two objects: cnum2, based on cnum1

## What is “static”?

```
class Complex
{
 public double real, imag;
 public static int numComplex = 0;

 // the following is default constructor
 public Complex()
 {
 real = 0.0;
 imag = 0.0;
 numComplex++;
 }
}
```



## Using “static”

- static variables in a class
  - called “**class variables**”
  - don’t need an instance of an object to access variable
    - use `ClassName.variable`
    - eg: `Math.PI`, `Complex.numComplex`
  - only one copy of variable in the system
  - all instances of object access the same variable (same value)
  - any object can change value of “class variables”
  - all objects all see the same value
- variables in a class that aren’t static
  - called “**instance variables**”
  - must** have an instance of an object to access variable
    - use `objectName.variable`
    - eg: `cnum.real`, `cnum.imag`
  - one copy of variable per object
  - any object can change its own instance variable
  - cannot change instance variable of another object
    - unless it has a reference to that object
- static methods
  - called “**class methods**”, eg: `main()`
  - don’t need an instance of an object to call static methods
    - use `ClassName.method()`
    - eg: `Math.sqrt( 3.0 );`
- methods that aren’t static
  - called “**instance methods**”, eg: constructors
  - must** have an instance of an object to call the method
    - use `objectName.method()`
    - eg: `cnum1.addToReal( 1.0 );`

## Reading (Objects)

- Chapter 3
  - objects
  - 3.1, 3.2, 3.3
  - review questions
    - R3.1, R3.3, R3.4, R3.9
- Chapter 8
  - more about classes and objects
  - goes into more detail than I want to teach right now
  - read “lightly”: 8.1, 8.2, 8.3, 8.4, 8.5, 8.6, 8.7, 8.8
  - review questions
    - R8.3, R8.7, R8.8, R8.9, R8.10

## Monday

- assignment and cloning (not covered well in book)
- Chapter 3
  - 3.8 (cloning)
- strings as objects (not covered well in book)
- Chapter 2
  - 2.6 (Strings)

## Course Reading (so far... covered by Quiz)

Chapter 1: all  
 Chapter 2: 2.1, 2.3, 2.4, 2.5, (Appendix A2.3)  
 Chapter 3: 3.1, 3.2, 3.3  
 Chapter 4: 4.1, 4.2, 4.4, 4.5, 4.6, 4.7, 4.9, (Appendix A2.3)  
 Chapter 5: 5.1, 5.2, 5.4, 5.5, 5.6, 5.7, 5.9, 5.10, 5.11, 5.12  
 Chapter 6: 6.1, 6.2, 6.3, 6.5, 6.6  
 Chapter 7: (read lightly) 7.1, 7.2, 7.3, 7.4, 7.5, 7.7  
 Chapter 8: (read lightly) 8.1, 8.2, 8.3, 8.4, 8.5, 8.6, 8.7, 8.8

## Review Problems

- Chapter 2
  - R2.3, 2.8, 2.10, 2.15, 2.17abcd
  - P2.2, 2.3 (don’t use if), 2.5, 2.9, 2.12, 2.16, 2.20
- Chapter 3
  - R3.1, 3.3, 3.4, 3.9
- Chapter 4
  - R4.1, 4.2, 4.3, 4.4, 4.6, 4.8, 4.9
  - P4.3, 4.6, 4.9, 4.12, 4.15, 4.17
- Chapter 5
  - R5.1, 5.2, 5.4, 5.7, 5.9, 5.16
  - P5.2, 5.6, 5.12, 5.16
- Chapter 6
  - R6.1, 6.2, 6.3, 6.4, 6.5, 6.6, 6.11, 6.12, 6.15, 6.18, 6.20, 6.21
  - P6.2, 6.5, 6.9, 6.12, 6.13, 6.18
- Chapter 7
  - R7.16
- Chapter 8
  - R8.3, R8.7, R8.8, R8.9, R8.10