3.1 Objects



Emgineeting Programming in Java: An Interdisciplinary Approach · Robert Sedgewick and Kevin Wayne · Copyright © 2002–2010 · 7/28/2015 3:46:55 PM

Penn

A Foundation for Programming



Data Types

Data Types: set of values and associated operations

Primitive Types:

- values map directly to the machine representation
- ops map directly to machine instructions

Data Type	Set of Values	Operations
boolean	true, false	not, and, or, xor
int	-2 ³¹ to 2 ³¹ -1	add, subtract, multiply
double	any of 2 ⁶⁴ possible reals	add, subtract, multiply

We want to write programs that handle other data types

- colors, pictures, strings, input streams, ...
- complex numbers, vectors, matrices, polynomials, ...
- points, polygons, charged particles, celestial bodies, ...



Objects

Objects: represent values and operations for more complex data types

- Object variables are called <u>fields</u>
- Object operations are called <u>methods</u>

Data Type	Set of Values	Operations
Color	24 bits	get red component, brighten
Picture	2D array of colors	get/set color of pixel (i, j)
String	sequence of characters	length, substring, compare

Objects are said to <u>encapsulate</u> (hide) its detail

- How an object is implemented is not important
- What it does is important

Objects can be created and referenced with variables



Object-Oriented Programming

Programming paradigm that views a program as a collection of interacting objects

In contrast, the conventional model views the program as a list of tasks (subroutines or functions)

We'll talk about how to:

- Create your own data types (set of values and operations)
- Use objects in your programs (e.g., manipulate objects)

Why would I want to use objects in my programs?

- Simplify your code
- Make your code easier to modify
- Share an object with a friend



The String Object

Fields:

■???

Methods:

- ■boolean equals(String anotherString)
- int length()
- String substring(int beginIdx, int endIdx)
- String toLowerCase()
- String toUpperCase()

• • • •

http://download.oracle.com/javase/1.4.2/docs/api/



Constructors and Methods

To construct a new object:

- Use keyword new (to invoke constructor)
- Use name of data type (to specify which type of object) with associated parameters for the constructor

To apply an operation:

- Use name of object (to specify which object)
- Use the dot operator (to access a member of the object)
- Use the name of the method (to specify which operation)





Defining Your Own Objects with Classes

- Classes are blueprints or <u>prototypes</u> for new objects
- Classes define all <u>field</u> and <u>method</u> declarations ... which are repeated for each new object created
- Using a class to create a new object is called <u>instantiating</u> an object

... creating a new object instance of the class

Classes often model real-world items



Constructors

 A special method that is used in order to instantiate an object

```
String s = new String("Hello World");
```

 If we made a Person class where you could create people with different names then you create a new person object by doing

Person p = new Person("Arvind");

Rule – Constructor has the same name as the name of the class.



 What do we want to have the ball <u>do</u>? (i.e., what <u>methods</u> should it have?)

• What initial parameters should we specify in the constructor?







• What do we want to have the ball <u>do</u>?

(i.e., what methods should it have?)

- void draw() : "Ball, draw thyself!"
- void update() : simulate the ball's motion

• What initial parameters should we specify in the constructor?





• What do we want to have the ball <u>do</u>?

(i.e., what methods should it have?)

- void draw() : "Ball, draw thyself!"
- void update() : simulate the ball's motion

- What initial parameters should we specify in the constructor?
 - Ball() : creates a ball at a random location
 - Ball (int x, int y) : creates a ball at (x, y)

These methods constitute the ball's API (Application Programming Interface)



Given only the API, we can use the object in a program:

```
Ball
                                        an array
public class BouncingBallStdDraw {
                                        of Balls.
                                                    Ball()
                                                    Ball(int x, int y)
   public static void main(String[] args) {
       for (int i=0; i< balls.length; i++){</pre>
                                                    void draw()
           balls[i] = new Ball();
                                                    void update()
       }
       for (int i =0; i <300; i++){</pre>
           StdDraw.clear();
                                                  New objects are
           for (int j=0; j < balls.length; j++)</pre>
                                                  created with the
              balls[j].draw();
                                                  new keyword.
           StdDraw.show(200);
           for (int j=0; j< balls.length; j++)</pre>
              balls[j].update();
       }
    }
```

Methods of objects stored in the array are accessed using dot-notation.



Where to Write Your Class

- Generally put each class in a separate file
- A class named MyClass is expected to be found in a file named MyClass.java
- Declare the class to be public
- This class can now be used as a 'data type' in your other programs



Comparing Declarations and Initializers

```
int i;
int j = 3;
float f = 0.1;
float[] f2 = new float[20];
String s1 = "abc";
String s2 = new String("abc");
Ball b = new Ball();
Ball[] b2 = new Ball[20];
for (int i = 0; i < b2.length; i++) {
    b2[i] = new Ball();
}
```



- Allow client to manipulate an object as a single entity
- Essentially a machine address (pointer)

```
Ball b1 = new Ball();
b1.update();
b1.update();
Ball b2 = new Ball();
b2.update();
b2 = b1;
b2.update();
```

addr	value
C0	0
C1	0
C2	0
С3	0
C4	0
C5	0
C6	0
C7	0
C8	0
С9	0
CA	0
СВ	0
CC	0

- Allow client to manipulate an object as a single entity
- Essentially a machine address (pointer)

Ball b1 = new Ball();
b1.update();
b1.update();
<pre>Ball b2 = new Ball(); b2.update();</pre>
<pre>b2 = b1; b2.update();</pre>

addr	valu
C0	0.50
C1	0.50
C2	0.05
 C3	0.01
 C4	0.03
C5	0
C6	0
C7	0
C8	0
С9	0
CA	0
CB	0
CC	0

b1

C0

- Allow client to manipulate an object as a single entity
- Essentially a machine address (pointer)

Ball b1 = new Ball();
b1.update();
b1.update();
<pre>Ball b2 = new Ball();</pre>
b2.update();
b2 = b1;
b2.update();



	addr	value
->	C0	0.55
	C1	0.51
	C2	0.05
	C3	0.01
	C4	0.03
	C5	0
	C6	0
	C7	0
	C8	0
	С9	0
	CA	0
	СВ	0
	CC	0

registers

- Allow client to manipulate an object as a single entity
- Essentially a machine address (pointer)

```
Ball b1 = new Ball();
b1.update();
b1.update();
Ball b2 = new Ball();
b2.update();
b2 = b1;
b2.update();
```



	addr	value
-	C0	0.60
	C1	0.52
	C2	0.05
	C3	0.01
	C4	0.03
	C5	0
	C6	0
	C7	0
	C8	0
	С9	0
	CA	0
	CB	0
	CC	0

registers

- Allow client to manipulate an object as a single entity
- Essentially a machine address (pointer)

```
Ball b1 = new Ball();
b1.update();
b1.update();
Ball b2 = new Ball();
b2.update();
b2 = b1;
b2.update();
```



registers

- Allow client to manipulate an object as a single entity
- Essentially a machine address (pointer)

```
Ball b1 = new Ball();
b1.update();
b1.update();
Ball b2 = new Ball();
b2.update();
b2 = b1;
b2.update();
```



registers

- Allow client to manipulate an object as a single entity
- Essentially a machine address (pointer)

```
Ball b1 = new Ball();
b1.update();
b1.update();
Ball b2 = new Ball();
b2.update();
b2 = b1;
b2.update();
```

C7 – CB can be reused for other variables. Known as garbage collection in java.



registers

b1

C0

b2

C0

- Allow client to manipulate an object as a single entity
- Essentially a machine address (pointer)

```
Ball b1 = new Ball();
b1.update();
b1.update();
Ball b2 = new Ball();
b2.update();
b2 = b1;
b2.update();
```

Moving b2 also moves b1 since they are aliases that reference the same object.



registers

addr

value

0.65

0.53

0.05

0.01

0.03

0

0

0.57

0.54

0.07

0.04

0.04

0

main memory (64-bit machine)

CC

Pass-By-Value

Arguments to methods are always passed by value.

- Primitive types: passes copy of value of actual parameter.
- Objects: passes copy of reference to actual parameter.

```
public class PassByValue {
   static void update(int a, int[] b, String c) {
           = 7;
      а
     b[3] = 7;
          = "seven";
      С
      StdOut.println(a + " " + b[3] + " " + c);
   }
   public static void main(String[] args) {
      int a = 3;
      int[] b = { 0, 1, 2, 3, 4, 5 };
      String c = "three";
      StdOut.println(a + " " + b[3] + " " + c);
      update(a, b, c);
      StdOut.println(a + " " + b[3] + " " + c);
   }
}
                                          % java PassByValue
```

- 3 3 three
- 7 7 seven
- 3 7 three

Encapsulation



Access Control

- Encapsulation is implemented using *access control*.
 - Separates interface from implementation
 - Provides a boundary for the client programmer
- Visible parts of the class (the *interface*)
 - can be used and/or changed by the client programmer.
- Hidden parts of the class (the *implementation*)
 - Can be changed by the class creator without impacting any of the client programmer's code
 - Can't be corrupted by the client programmer



Access Control in Java

- Visibility modifiers provide access control to instance variables and methods.
 - *public* visibility accessible by everyone, in particular the client programmer
 - A class' interface is defined by its public methods.
 - *private* visibility accessible only by the methods within the class
 - Two others—*protected* and package outside the scope of this course



Good Programming Practice

- Combine methods and data in a single class
- Label <u>all</u> instance variables as private for information hiding
 - The class has complete control over how/when/if the instance variables are changed
 - Fields primarily support class behavior
- Minimize the class' public interface
- Public interface should offer only those methods that a client needs in order to 'interact' with the class



Using this

You can think of this as an implicit private reference to the current instance.



Note that b1.year and b1.this.year refer to the same field



Overloaded Constructors

```
public class Date {
  private int month; // 1 - 12
  private int day; // 1 - 31
  private int year; // 4 digits
  // no-argument constructor
                                // 1 Jan 1900
  public Date() {
    month = 1;
                               Date d1 = new Date();
    day = 1;
    year = 1900;
                                // 30 Oct 2013
  }
                               Date d2 = new Date(10, 30, 2013);
  // alternative constructor
  public Date(int month, int day, int year) {
```

this.month = month; this.day = day; this.year = year; } Note the us keyword to

Note the usage of the this keyword to avoid the obvious ambiguity



}

. . .

Accessors & Mutator

- Class *behavior* <u>may</u> allow access to, or modification of, individual private instance variables.
- Accessor method
 - retrieves the value of a private instance variable
 - conventional to start the method name with get
- Mutator method
 - changes the value of a private instance variable
 - conventional to start the name of the method with set
- Gives the client program <u>indirect</u> access to the instance variables.



More Accessors and Mutators

Question: Doesn't the use of accessors and mutators defeat the purpose of making the instance variables **private**?

Answer: No

- The class implementer decides which instance variables will have accessors.
- Mutators can:
 - validate the new value of the instance variable, and
 - decide whether or not to actually make the requested change.



Accessor and Mutator Example

```
public class Date {
  private int month; // 1 - 12
  private int day; // 1 - 31
  private int year; // 4-digit year
  // accessors return the value of private data
  public int getMonth() { return month; }
  // mutators can validate the new value
  public boolean setMonth(int month) {
   if (1 <= month && month <= 12) {
       this.month = month;
       return true;
   }
   else // this is an invalid month
      return false;
   }
  // rest of class definition follows
```

ngineering

Accessor/Mutator Caution

- In general you should NOT provide accessors and mutators for all private instance variables.
 - Recall that the principle of encapsulation is best served with a *limited class interface*.



Private Methods

- Methods may be private.
 - Cannot be invoked by a client program
 - Can only be called by other methods within the same class definition
 - Most commonly used as "helper" methods to support top-down implementation of a public method



Private Method Example

```
public class Date {
   private int month; // 1 - 12
   private int day; // 1 - 31
   private int year; // 4-digit year
   // accessors return the value of private data
   public int getMonth() { return month; }
   // mutators can validate the new value
   public boolean setMonth(int month) {
    if (isValidMonth(month)) {
        this.month = month;
        return true;
    }
    else // this is an invalid month
       return false;
   }
   // helper method - internal use only
  private boolean isValidMonth(int month) {
     return 1 <= month && month <= 12;
   }
```

Ingineering

Static and Final



Static Variable

- A static variable belongs to the class as a whole, not just to one object.
- There is only one copy of a static variable per class.
 - All objects of the class can read and change this static variable.
- A static variable is declared with the addition of the modifier static.
 static int myStaticVariable = 0;



Static Variable

- The most common usage of a static variable is in order to keep track of the number of instances of an object.
- Assume class called Human. There is some 'controlling' class which creates humans (new Human()) and it also is responsible for the death of humans.
- We would like to keep track of the number of Humans. One way to do this would be have a static variable in the Human class which gets incremented upon child birth and decremented upon death.



Static Constants

- A *static constant* is used to symbolically represent a constant value.
 - The declaration for a static constant includes the modifier final, which indicates that its value cannot be changed:
 public static final float PI = 3.142;
- It is not necessary to instantiate an object to access a static variable, constant or method.
- When referring to such a constant outside its class, use the name of its class in place of a calling object.

float radius = MyClass.PI * radius * radius;



Rules for Static Methods

- Static methods have no calling/host object (they have no this).
- Therefore, static methods <u>cannot</u>:
 - Refer to any instance variables of the class
 - Invoke any method that has an implicit or explicit this for a calling object
- Static methods <u>may</u> invoke other static methods or refer to static variables and constants.
- A class definition may contain both static methods and non-static methods.



main = Static Method

Note that the method header for main() is

public static void main(String[] args)

Being static has two effects:

- main can be executed without an object.
- "Helper" methods called by main must also be static.
 - Hence public static when you were first introduced to functions



Any Class Can Have a main()

- Every class can have a public static method name main().
- Java will execute the main that exists in whichever class you choose to run

java <className>

• A convenient way to write test code for your class.



Static Review

Given the skeleton class definition below

```
public class C {
  public int a = 0;
  public static int b = 1;
  public void f() {...}
  public static void g() {...}
}
```

- Can body of f() refer to a?
- Can body of f() refer to b?
- Can body of g() refer to a?
- Can body of g() refer to b?
- Can f() call g()?
- Can g() call f()?

For each, explain why or why not.



BACKUP/EXTRA SLIDES

Complex Numbers

Complex Number Data Type

Goal: Create a data type for complex numbers Values: the real and imaginary parts (doubles)

public class Complex

Complex(double real, double imag)Complexplus(Complex b)String toString()sum of this number and bstring representation

$$a = 3 + 4i, b = -2 + 3i$$

 $a + b = 1 + 7i$
 $a \times b = -18 + i$
 $|a| = 5$

Applications of Complex Numbers

- Fractals
- Impedance in RLC circuits
- Signal processing and Fourier analysis
- Control theory and Laplace transforms
- Quantum mechanics and Hilbert spaces
- •••



Complex Number Data Type: A Simple Client



Complex Number Data Type: Implementation

fields

public class Complex {

```
private final double re;
private final double im;
```

```
public Complex(double real, double imag) {
    re = real;
    im = imag;
}
```

Mandelbrot Set

Mandelbrot set: A set of complex numbers



Mandelbrot Set

Is complex number z_0 in the set?

- Iterate $z_{t+1} = (z_t)^2 + z_0$
- If | z_t | diverges to infinity, then z₀ is not in set;
 otherwise z₀ is in set

t	z _t
0	-1/2 + Oi
1	-1/4 + Oi
2	-7/16 + Oi
3	-79/256 + Oi
4	-26527/65536 + Oi
5	-1443801919/4294967296 + Oi

z = -1/2 is in Mandelbrot set

†	Z _†
0	1 + i
1	1 + 3i
2	-7 + 7i
3	1 - 97i
4	-9407 - 193i
5	88454401 + 3631103i

z = 1 + i not in Mandelbrot set

0.5

Plotting the Mandelbrot Set

Practical issues:

- Cannot plot infinitely many points
- Cannot iterate infinitely many times

Approximate solution:

- Sample from an *N*-by-*N* grid of points in the plane
- Fact: if $|z_t| > 2$ for any *t*, then *z* not in Mandelbrot set
 - $-if |z_{255}| \le 2$ then z "likely" in Mandelbrot set





0.5

Complex Number Data Type: Another Client

Mandelbrot function with complex numbers

- Is z_0 in the Mandelbrot set?
- Returns white (definitely no) or black (probably yes)



More dramatic picture: replace stdDraw.WHITE with grayscale or color

Complex Number Data Type: Another Client

Plot the Mandelbrot set in gray scale.

```
public static void main(String[] args) {
   double xc = Double.parseDouble(args[0]);
   double yc = Double.parseDouble(args[1]);
   double size = Double.parseDouble(args[2]);
   int N = 512;
   Picture pic = new Picture (N, N);
   for (int i = 0; i < N; i++) {</pre>
      for (int j = 0; j < N; j++) {</pre>
         double x0 = xc - size/2 + size*i/N;
         double y0 = yc - size/2 + size*j/N;
         Complex z0 = new Complex (x0, y0); \checkmark
         Color color = mand(z0);
                                                 scale to screen
         pic.set(i, N-1-j, color);
                                                 coordinates
                           (0, 0) is upper left
   pic.show();
```

Mandelbrot Set

% java Mandelbrot -.5 0 2



% java Mandelbrot .1045 -.637 .01

