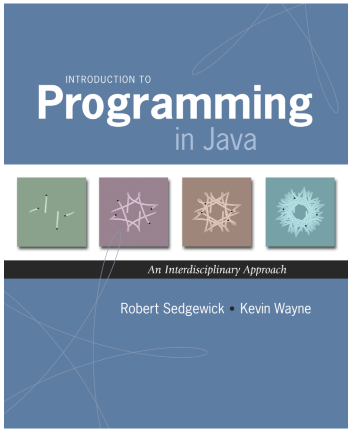
3.2 Creating Data Types



Introduction to Programming in Java: An Interdisciplinary Approach · Robert Sedgewick and Kevin Wayne · Copyright © 2008 · March 26, 2009 3:51 PM

Data Types

Data type. Set of values and operations on those values.

Basic types.

Data Type	Set of Values	Some Operations
`boolean`	`true`, `false`	not, and, or, xor
`int`	-2^{31} to $2^{31} - 1$	add, subtract, multiply
`String`	sequence of Unicode characters	concatenate, compare

Last time. Write programs that use data types.
Today. Write programs to create our own data types.

Defining Data Types in Java

To define a data type, specify:

- Set of values.
- Operations defined on those values.

Java class. Defines a data type by specifying:

- Instance variables. (set of values)
- Methods. (operations defined on those values)
- Constructors. (create and initialize new objects)

Point Charge Data Type

Goal. Create a data type to manipulate point charges.

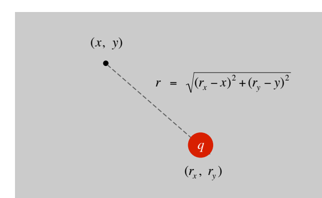
Set of values. Three real numbers. [position and electrical charge]

Operations.

- Create a new point charge at (r_x, r_y) with electric charge q.
- Determine electric potential V at (x, y) due to point charge.
- Convert to string.

$$V = k \frac{q}{r}$$

r = distance between (x, y) and (r_x, r_y)
k = electrostatic constant = $8.99 \times 10^9 \ \mathrm{N \cdot m^2 / C^2}$



Point Charge Data Type

Goal. Create a data type to manipulate point charges.

Set of values. Three real numbers. [position and electrical charge]

API.

public class Charge		
	Charge(double x0, double y0, double q0)	
double	potentialAt(double x, double y)	*electric potential at (x, y) due to charge*
String	toString()	*string representation*

Charge Data Type: A Simple Client

Client program. Uses data type operations to calculate something.

```
public static void main(String[] args) {
   double x = Double.parseDouble(args[0]);
   double y = Double.parseDouble(args[1]);
   Charge c1 = new Charge(.51, .63, 21.3);
   Charge c2 = new Charge(.13, .94, 81.9);
   double v1 = c1.potentialAt(x, y);
   double v2 = c2.potentialAt(x, y);
   StdOut.println(c1);
   StdOut.println(c2);
   StdOut.println(v1 + v2);
}
```

automagically invokes the toString() method

```
% java Charge .50 .50
21.3 at (0.51, 0.63)
81.9 at (0.13, 0.94)
2.74936907085912e12
```


Anatomy of Instance Variables

Instance variables. Specifies the set of values.

- Declare outside any method.
- Always use access modifier `private`.
- Use modifier `final` with instance variables that never change.

stay tuned

```
public class Charge
{
   private final double rx, ry;
   private final double q;
   .
   .
   .
}
```

instance variable declarations

modifiers

Anatomy of a Constructor

Constructor. Specifies what happens when you create a new object.

access modifier

no return type

constructor name (same as class name)

argument variables

```
public Charge ( double x0 , double y0 , double q0 )
{
   rx = x0;
   ry = y0;
   q = q0;
}
```

signature

instance variable names

body

Invoking a constructor. Use `new` operator to create a new object.

```
Charge c1 = new Charge(.51, .63, 21.3);
Charge c2 = new Charge(.13, .94, 81.9);
```

invoke constructor

Anatomy of a Data Type Method

Method. Define operations on instance variables.

access modifier · return type · method name · argument variables · signature

```
public double potentialAt(double x, double y)
{
   double k  = 8.99e09;
   double dx = x - rx;
   double dy = y - ry;
   return k * q / Math.sqrt(dx*dx + dy*dy);
}
```

local variables · argument variable name · instance variable name · call on a static method · local variable name

Invoking a method. Use dot operator to invoke a method.

```
double v1 = c1.potentialAt(x, y);
double v2 = c2.potentialAt(x, y);
```

object name · invoke method

Anatomy of a Class

```
public class Charge
{
   private final double rx, ry;
   private final double q;

   public Charge(double x0, double y0, double q0)
   {  rx = x0; ry = y0; q = q0;  }

   public double potentialAt(double x, double y)
   {
      double k = 8.99e09;
      double dx = x - rx;
      double dy = y - ry;
      return k * q / Math.sqrt(dx*dx + dy*dy);
   }

   public String toString()
   {  return q +" at " + "("+ rx + ", " + ry +")";  }

   public static void main(String[] args)
   {
      double x = Double.parseDouble(args[0]);
      double y = Double.parseDouble(args[1]);
      Charge c1 = new Charge(.51, .63, 21.3);
      Charge c2 = new Charge(.13, .94, 81.9);
      double v1 = c1.potentialAt(x, y);
      double v2 = c2.potentialAt(x, y);
      StdOut.printf("%.1e\n", (v1 + v2));
   }
}
```

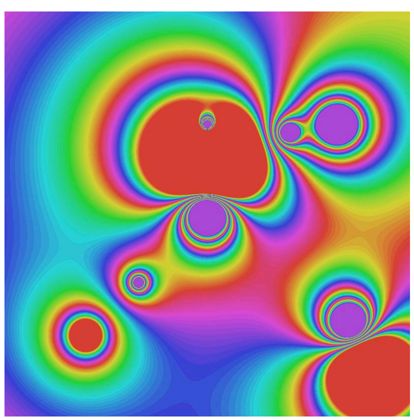
class name · instance variables · constructor · instance methods · instance variable names · test client · create and initialize object · invoke constructor · object name · invoke method

Potential Visualization

Potential visualization. Read in N point charges from standard input; compute total potential at each point in unit square.

```
% more charges.txt
9
.51 .63 -100
.50 .50   40
.50 .72   10
.33 .33    5
.20 .20  -10
.70 .70   10
.82 .72   20
.85 .23   30
.90 .12  -50
```

```
% java Potential < charges.txt
```



Potential Visualization

Arrays of objects. Allocate memory for the array with `new`; then allocate memory for each individual object with `new`.

```
// read in the data
int N = StdIn.readInt();
Charge[] a = new Charge[N];
for (int i = 0; i < N; i++) {
   double x0 = StdIn.readDouble();
   double y0 = StdIn.readDouble();
   double q0 = StdIn.readDouble();
   a[i] = new Charge(x0, y0, q0);
}
```

Potential Visualization

```
// plot the data
int SIZE = 512;
Picture pic = new Picture(SIZE, SIZE);
for (int i = 0; i < SIZE; i++) {
   for (int j = 0; j < SIZE; j++) {
      double V = 0.0;
      for (int k = 0; k < N; k++) {
         double x = 1.0 * i / SIZE;
         double y = 1.0 * j / SIZE;
         V += a[k].potentialAt(x, y);
      }
      Color color = getColor(V);
      pic.set(i, SIZE-1-j, color);
   }
}
pic.show();
```

$$V = \sum_k (k\, q_k \;/ r_k)$$

compute color as a function of potential V

(0, 0) is upper left

Turtle Graphics

Turtle Graphics

Goal. Create a data type to manipulate a turtle moving in the plane.
Set of values. Location and orientation of turtle.

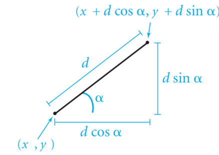
API.

public class Turtle	
Turtle(double x0, double y0, double a0)	*create a new turtle at (x_0, y_0) facing a_0 degrees counterclockwise from the x-axis*
void turnLeft(double delta)	*rotate* `delta` *degrees counterclockwise*
void goForward(double step)	*move distance* `step`*, drawing a line*

```
// draw a square
Turtle turtle = new Turtle(0.0, 0.0, 0.0);
turtle.goForward(1.0);
turtle.turnLeft(90.0);
turtle.goForward(1.0);
turtle.turnLeft(90.0);
turtle.goForward(1.0);
turtle.turnLeft(90.0);
turtle.goForward(1.0);
turtle.turnLeft(90.0);
```

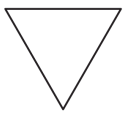

Turtle Graphics

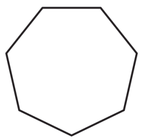
```
public class Turtle {
   private double x, y;   // turtle is at (x, y)
   private double angle;  // facing this direction

   public Turtle(double x0, double y0, double a0) {
      x = x0;
      y = y0;
      angle = a0;
   }

   public void turnLeft(double delta) {
      angle += delta;
   }

   public void goForward(double d) {
      double oldx = x;
      double oldy = y;
      x += d * Math.cos(Math.toRadians(angle));
      y += d * Math.sin(Math.toRadians(angle));
      StdDraw.line(oldx, oldy, x, y);
   }
}
```

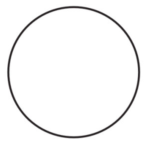


N-gon

```
public class Ngon {
   public static void main(String[] args) {
      int N         = Integer.parseInt(args[0]);
      double angle = 360.0 / N;
      double step  = Math.sin(Math.toRadians(angle/2.0));
      Turtle turtle = new Turtle(0.5, 0, angle/2.0);
      for (int i = 0; i < N; i++) {
         turtle.goForward(step);
         turtle.turnLeft(angle);
      }
   }
}
```

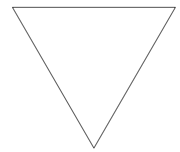

3

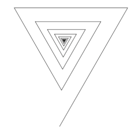

7

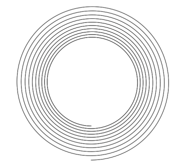

1440

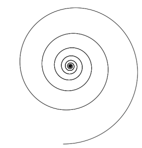
Spira Mirabilis

```
public class Spiral {
   public static void main(String[] args) {
      int N         = Integer.parseInt(args[0]);
      double decay = Double.parseDouble(args[1]);
      double angle = 360.0 / N;
      double step  = Math.sin(Math.toRadians(angle/2.0));
      Turtle turtle = new Turtle(0.5, 0, angle/2.0);
      for (int i = 0; i < 10 * N; i++) {
         step /= decay;
         turtle.goForward(step);
         turtle.turnLeft(angle);
      }
   }
}
```

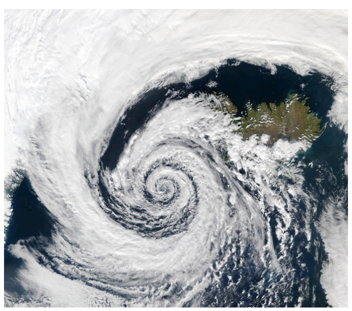

3 1.0


3 1.2


1440 1.00004


1440 1.0004

Spira Mirabilis in Nature



Complex Numbers

Complex Number Data Type

Goal. Create a data type to manipulate complex numbers.
Set of values. Two real numbers: real and imaginary parts.

API.

public class Complex		
	Complex(double real, double imag)	
Complex	plus(Complex b)	*sum of this number and* b
Complex	times(Complex b)	*product of this number and* b
double	abs()	*magnitude*
String	toString()	*string representation*

$a = 3 + 4i,\ b = -2 + 3i$
$a + b = 1 + 7i$
$a \times b = -18 + i$
$|a| = 5$

Applications of Complex Numbers

Relevance. A quintessential mathematical abstraction.

Applications.
- 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

Client program. Uses data type operations to calculate something.

```
public static void main(String[] args) {
   Complex a = new Complex( 3.0, 4.0);
   Complex b = new Complex(-2.0, 3.0);
   Complex c = a.times(b);
   StdOut.println("a = " + a);
   StdOut.println("b = " + b);
   StdOut.println("c = " + c);
}
```

result of c.toString()

```
% java TestClient
a = 3.0 + 4.0i
b = -2.0 + 3.0i
c = -18.0 + 1.0i
```

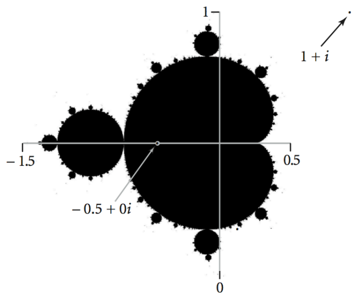
Remark. Can't write a = b*c since no operator overloading in Java.

Complex Number Data Type: Implementation

```
public class Complex {

   private final double re;
   private final double im;                      instance variables

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

   public String toString() { return re + " + " + im + "i"; }

   public double abs() { return Math.sqrt(re*re + im*im); }

   public Complex plus(Complex b) {
      double real = re + b.re;
      double imag = im + b.im;                   creates a Complex object,
      return new Complex(real, imag);            and returns a reference to it
   }
                                                 refers to b's instance variable
   public Complex times(Complex b) {
      double real = re * b.re – im * b.im;
      double imag = re * b.im + im * b.re;
      return new Complex(real, imag);
   }                                             methods
}
```

Mandelbrot Set

Mandelbrot set. A set of complex numbers.
Plot. Plot (x, y) black if $z = x + y\,i$ is in the set, and white otherwise.

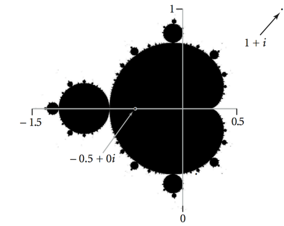


- No simple formula describes which complex numbers are in set.
- Instead, describe using an algorithm.

Mandelbrot Set

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 z_0 is not in set; otherwise z_0 is in set.



t	z_t
0	-1/2 + 0i
1	-1/4 + 0i
2	-7/16 + 0i
3	-79/256 + 0i
4	-26527/65536 + 0i
5	-1443801919/4294967296 + 0i

z = -1/2 is in Mandelbrot set

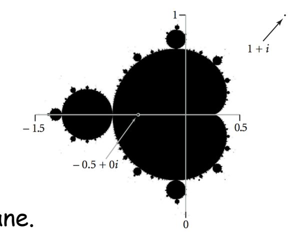
t	z_t
0	1 + i
1	1 + 3i
2	-7 + 7i
3	1 – 97i
4	-9407 – 193i
5	88454401 + 3631103i

z = 1 + i not in Mandelbrot set

Plotting the Mandelbrot Set

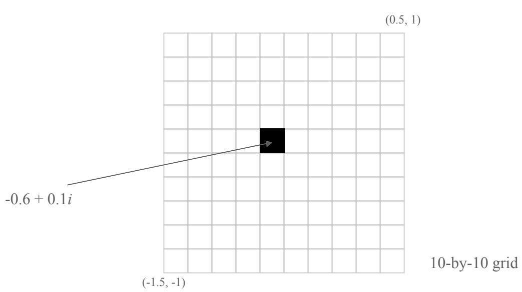
Practical issues.

- 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.
- Pseudo-fact: if $|z_{255}| \leq 2$ then z "likely" in Mandelbrot set.



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).

```
public static Color mand(Complex z0) {
   Complex z = z0;
   for (int t = 0; t < 255; t++) {
      if (z.abs() > 2.0) return StdDraw.WHITE;
      z = z.times(z);
      z = z.plus(z0);
   }
   return StdDraw.BLACK;
}
```

$z = z^2 + z_0$

More dramatic picture: replace `StdDraw.WHITE` with grayscale or color.

`new Color(255-t, 255-t, 255-t)`

Complex Number Data Type: Another Client

Plot the Mandelbrot set in gray scale.

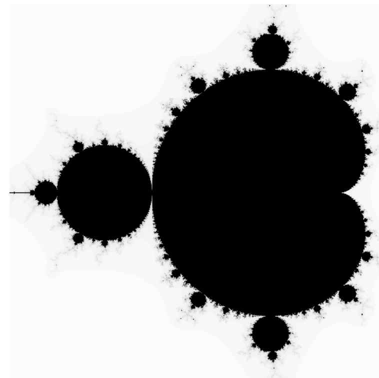
```
public static void main(String[] args) {
   double xc   = Double.parseDouble(args[0]);
   double yc   = Double.parseDouble(args[1]);
   double size = Double.parseDouble(args[2]);
   int N = 512;
   Picture pic = new Picture(N, N);

   for (int i = 0; i < N; i++) {
      for (int j = 0; j < N; j++) {
         double x0 = xc - size/2 + size*i/N;
         double y0 = yc - size/2 + size*j/N;
         Complex z0 = new Complex(x0, y0);
         Color color = mand(z0);
         pic.set(i, N-1-j, color);
      }
   }
   pic.show();
}
```

scale to screen coordinates

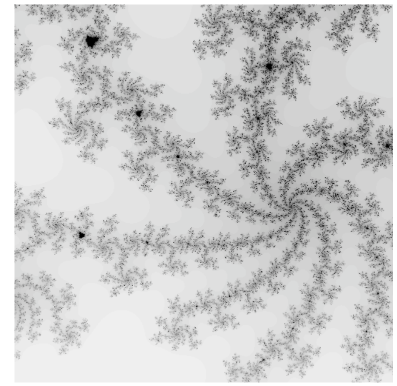
(0, 0) is upper left

Mandelbrot Set

`% java Mandelbrot -.5 0 2`

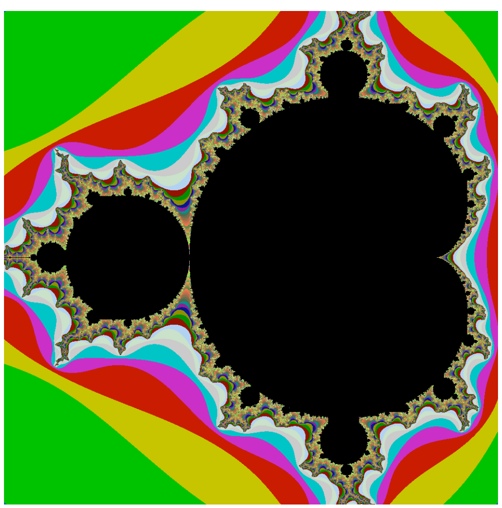


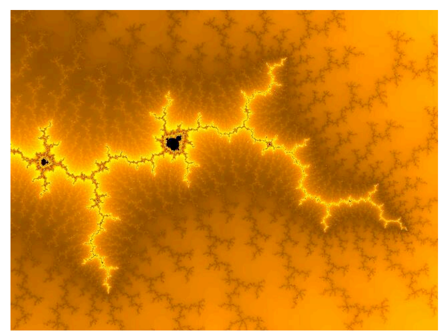
`% java Mandelbrot .1045 -.637 .01`

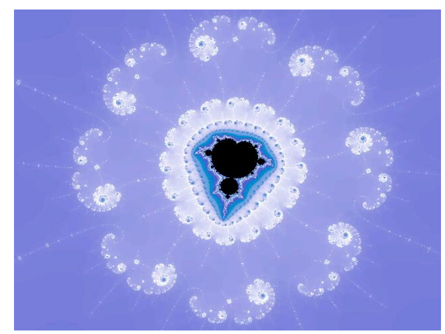


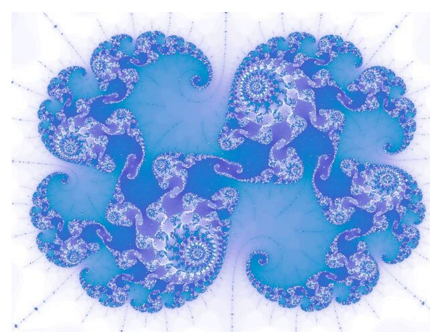
Mandelbrot Set

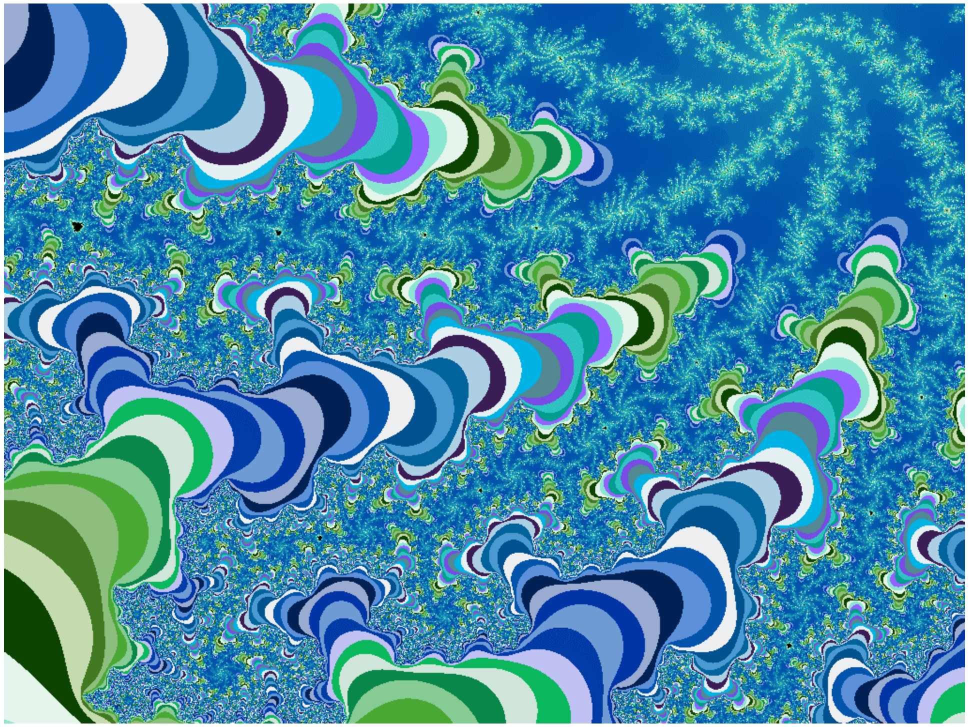
`% java ColorMandelbrot -.5 0 2 < mandel.txt`











Applications of Data Types

Data type. Set of values and collection of operations on those values.

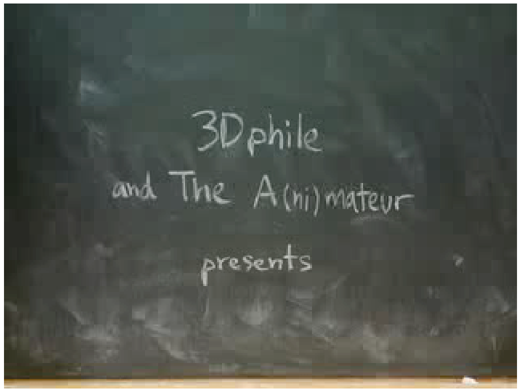
Simulating the physical world.

- Java objects model real-world objects.
- Not always easy to make model reflect reality.
- Ex: charged particle, molecule, COS 126 student, ...

Extending the Java language.

- Java doesn't have a data type for every possible application.
- Data types enable us to add our own abstractions.
- Ex: complex, vector, polynomial, matrix, ...

Mandelbrot Set Music Video



http://www.jonathancoulton.com/songdetails/Mandelbrot Set