COMPUTER
SCIENCE
An Interdisciplinary Approach

Section 3.2

ROBERT SEDGEWICK
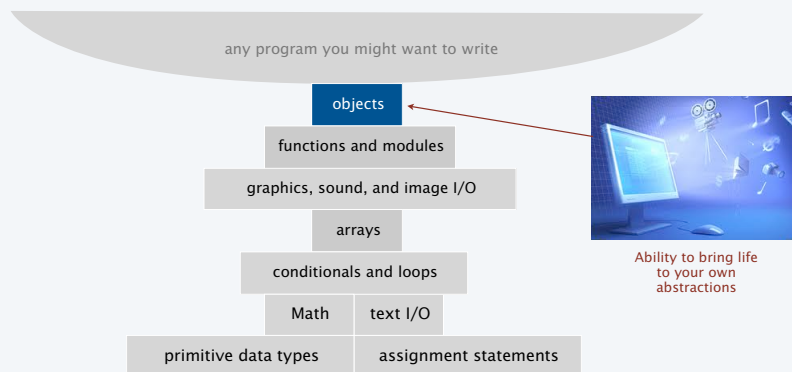KEVIN WAYNE

http://introcs.cs.princeton.edu

9. Creating Data Types

---

9. Creating Data Types

- **Overview**
- Point charges
- Turtle graphics
- Complex numbers

CS.9.A.CreatingDTs.Overview

Basic building blocks for programming

any program you might want to write

objects

functions and modules

graphics, sound, and image I/O

arrays

conditionals and loops

Math   text I/O

primitive data types   assignment statements

Ability to bring life
to your own
abstractions

3

Object-oriented programming (OOP)

Object-oriented programming (OOP).
- Create your own data types.
- Use them in your programs (manipulate *objects*).

An object holds a data type value.
Variable names refer to objects.

Examples

data type	set of values	examples of operations
Color	three 8-bit integers	get red component, brighten
Picture	2D array of colors	get/set color of pixel
String	sequence of characters	length, substring, compare

C A T A G C G C

An abstract data type is a data type whose representation is *hidden from the client*.

Impact: We can use ADTs without knowing implementation details.
- Previous lecture: how to write client programs for several useful ADTs
- This lecture: how to implement your own ADTs

4

Implementing a data type

To create a data type, you need provide code that
- Defines the set of values (instance variables).
- Implements operations on those values (methods).
- Creates and initialize new objects (constructors).

Instance variables
- Declarations associate variable names with types.
- Set of type values is "set of values".

Methods
- Like static methods.
- Can refer to instance variables.

Constructors
- Like a method with the same name as the type.
- No return type declaration.
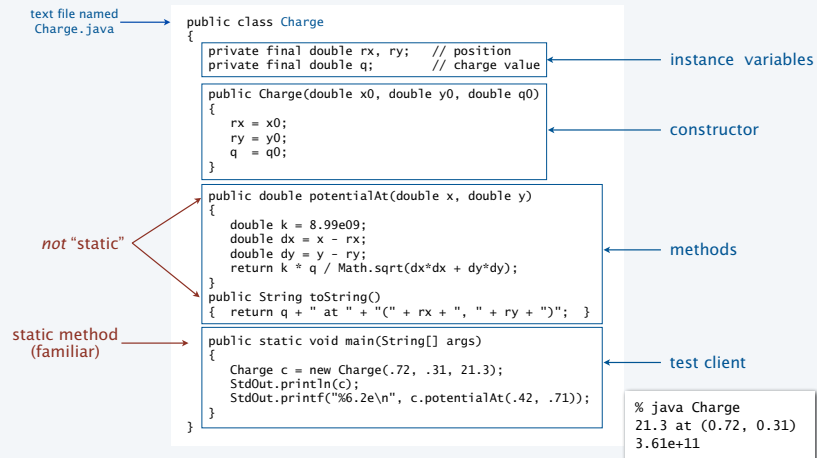- Invoked by new, returns object of the type.

In Java, a data-type implementation is known as a *class*.

A Java class

instance variables
constructors
methods
test client

5

Anatomy of a Class

text file named
Charge.java

```
public class Charge
{
   private final double rx, ry;   // position
   private final double q;        // charge value

   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)
   {
      Charge c = new Charge(.72, .31, 21.3);
      StdOut.println(c);
      StdOut.printf("%6.2e\n", c.potentialAt(.42, .71));
   }
}
```

instance variables

constructor

not "static"

methods

static method
(familiar)

test client

```
% java Charge
21.3 at (0.72, 0.31)
3.61e+11
```

6

COMPUTER SCIENCE
SEDGEWICK/WAYNE
PART I: PROGRAMMING IN JAVA

CS.9.A.CreatingDTs.Overview

COMPUTER SCIENCE
SEDGEWICK/WAYNE
PART I: PROGRAMMING IN JAVA

9. Creating Data Types

- Overview
- **Point charges**
- Turtle graphics
- Complex numbers

CS.9.B.CreatingDTs.Charges

ADT for point charges

A point charge is an idealized model of a particle that has an electric charge.

An ADT allows us to write Java programs that manipulate point charges.
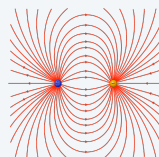
Values

	examples	
position (x, y)	$(.53, .63)$	$(.13, .94)$
electrical charge	20.1	81.9

API (operations)

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 of this charge*

Crash course on electric potential

Electric potential is a measure of the effect of a point charge on its surroundings.
- It increases in proportion to the charge value.
- It decreases in proportion to the *inverse of the distance* from the charge (2D).

Mathematically,
- Suppose a point charge c is located at (r_x, r_y) and has charge q.
- Let r be the distance between (x, y) and (r_x, r_y).
- Let $V_c(x,y)$ be the potential at (x, y) due to c.
- Then $V_c(x, y) = k\dfrac{q}{r}$ where $k = 8.99{\times}10^9$ is a normalizing factor.
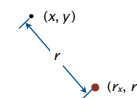
Q. What happens when multiple charges are present?

A. The potential at a point is the *sum* of the potentials due to the individual charges.

Note: Similar laws hold in many other situations. ←—— Example. *N*-body (3D) is an inverse *square* law.

Point charge implementation: Test client

Best practice. Begin by implementing a simple test client.

```
public static void main(String[] args)
{
    Charge c = new Charge(.72, .31, 20.1);
    StdOut.println(c);          ←—— Reminder: automatically invokes c.toString()
    StdOut.printf("%6.2e\n", c.potentialAt(.42, .71));
}
```

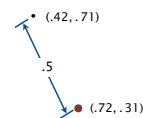| instance variables |
|---|
| constructors |
| methods |
| **test client** |
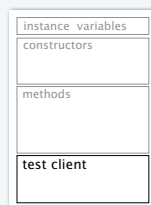
$$V_c(x,y) = k\frac{q}{r}$$

$$r = \sqrt{(r_x - x)^2 + (r_y - y)^2}$$
$$= \sqrt{.3^2 + .4^2} = .5$$
$$V_c(.42, .71) = 8.99 \times 10^9 \frac{20.1}{.5}$$
$$= 3.6 \times 10^{11}$$


```
% java Charge
20.1 at (0.72, 0.31)
3.61e+11
```
←—— What we *expect*, once the implementation is done.

Point charge implementation: Instance variables

Instance variables define data-type values.

instance variables
constructors
methods
test client

Values

|	examples	
position (x, y)	$(.53, .63)$	$(.13, .94)$
electrical charge	20.1	81.9

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

Modifiers control access.
- private denies clients access and therefore makes data type abstract.
- final disallows any change in value and documents that data type is *immutable*.

stay tuned

Key to OOP. Each *object* has instance-variable values.
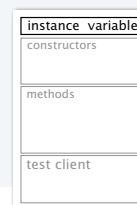
Point charge implementation: Constructor

Constructors create and initialize new objects.

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

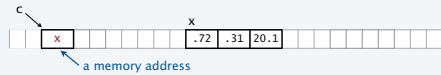references to instance variables, which are *not* declared within the constructor

instance variables
constructors
methods
test client

Clients use new to invoke constructors.
- Pass arguments as in a method call.
- Return value is reference to new object.

Possible memory representation of
`Charge c = new Charge(.72, .31, 20.1);`

c

x

| x |

| .72 | .31 | 20.1 |

a memory address

13

Point charge implementation: Methods

Methods define data-type operations (implement APIs).

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 of this charge*

instance variables
constructors
methods
test client

```
public class Charge
{
   ...
   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 + ")";  }
   ...
}
```

$$V_c(x, y) = k\frac{q}{r}$$

Key to OOP. An instance variable reference in an instance method *refers to the value for the object that was used to invoke the method.*

14

Point charge implementation

text file named
Charge.java

```
public class Charge
{
   private final double rx, ry;   // position
   private final double q;        // charge value

   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)
   {
       Charge c = new Charge(.72, .31, 20.1);
       StdOut.println(c);
       StdOut.printf("%6.2e\n", c.potentialAt(.42, .71));
   }
}
```

instance variables

constructor

methods

test client

```
% java Charge
20.1 at (0.72, 0.31)
3.61e+11
```

15

Point charge client: Potential visualization (helper methods)

Read point charges from StdIn.
- Uses Charge like any other type.
- Returns an array of Charges.

```
public static Charge[] readCharges()
{
   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);
   }
   return a;
}
```

Convert potential values to a color.
- Convert V to an 8-bit integer.
- Use grayscale.

```
public static Color toColor(double V)
{
   V = 128 + V / 2.0e10;
   int t = 0;
   if (V > 255) t = 255;
   else if (V >= 0) t = (int) V;
   return new Color(t, t, t);
}
```

| V											
t	0	1	...	37	38	39	...	128	...	254	255
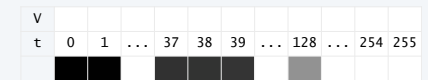
16

Point charge client: Potential visualization

```
import java.awt.Color;
public class Potential
{
    public static Charge[] readCharges()
    { // See previous slide. }
    public static Color toColor()
    { // See previous slide. }
    public static void main(String[] args)
    {
        Charge[] a = readCharges();
        int SIZE = 800;
        Picture pic = new Picture(SIZE, SIZE);
        for (int col = 0; col < SIZE; col++)
            for (int row = 0; row < SIZE; row++)
            {
                double V = 0.0;
                for (int k = 0; k < a.length; k++)
                {
                    double x = 1.0 * col / SIZE;
                    double y = 1.0 * row / SIZE;
                    V += a[k].potentialAt(x, y);
                }
                pic.set(col, SIZE-1-row, toColor(V));
            }
        pic.show();
    }
}
```

```
% more charges3.txt
3
.51 .63 -100
.50 .50   40
.50 .72   20
% java Potential < charges3.txt
```
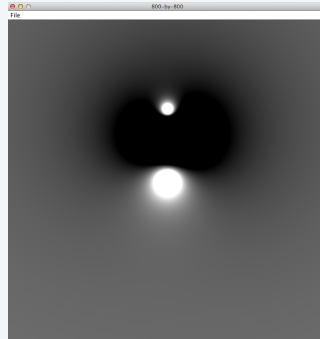

17

Potential visualization I

```
% more charges9.txt
9
.51 .63 -100
.50 .50   40
.50 .72   20
.33 .33    5
.20 .20  -10
.70 .70   10
.82 .72   20
.85 .23   30
.90 .12  -50

% java Potential < charges9.txt
```
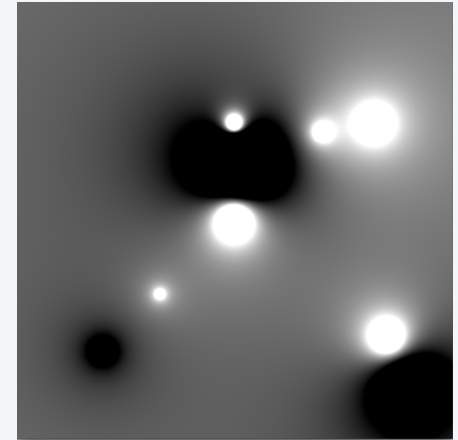

18

Potential visualization II: A moving charge

```
% more charges9.txt
9
.51 .63 -100
.50 .50   40
.50 .72   20
.33 .33    5
.20 .20  -10
.70 .70   10
.82 .72   20
.85 .23   30
.90 .12  -50

% java PotentialWithMovingCharge < charges9.txt
```
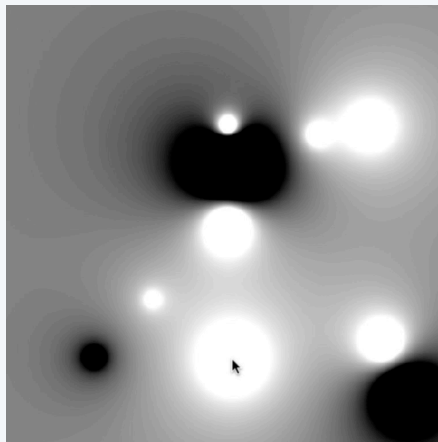

19

Potential visualization III: Discontinuous color map

```
public static Color toColor(double V)
{
    V = 128 + V / 2.0e10;
    int t = 0;
    if (V > 255) t = 255;
    else if (V >= 0) t = (int) V;
    t = t*37 % 255
    return new Color(t, t, t);
}
```
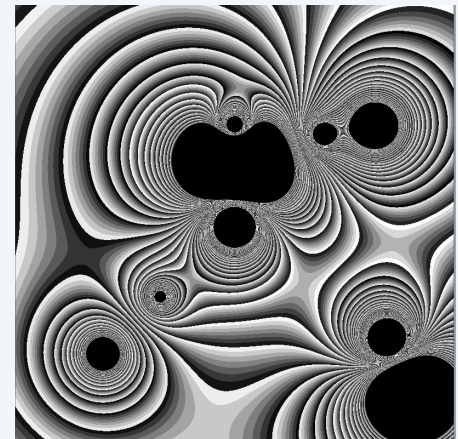
V											
t	0	1	2	3	4	5	6	7	8	9	...

20

9. Creating Data Types

- Overview
- Point charges
- **Turtle graphics**
- Complex numbers

ADT for turtle graphics

A turtle is an idealized model of a plotting device.

An ADT allows us to write Java programs that manipulate turtles.
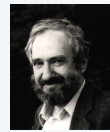
Seymour Papert
1928–

	position (x, y)	(.5, .5)	(.25, .75)	(.22, .12)
Values	orientation	90°	135°	10°

public class Turtle	
Turtle(double x0, double y0, double q0)	
void turnLeft(double delta)	*rotate* delta *degrees counterclockwise*
void goForward(double step)	*move distance* step, *drawing a line*

API (operations)

23

Turtle graphics implementation: Test client

Best practice. Begin by implementing a simple test client.

instance variables
constructors
methods
test client

```
public static void main(String[] args)
{
    Turtle turtle = new Turtle(0.0, 0.0, 0.0);
    turtle.goForward(1.0);
    turtle.turnLeft(120.0);
    turtle.goForward(1.0);
    turtle.turnLeft(120.0);
    turtle.goForward(1.0);
    turtle.turnLeft(120.0);
}
```

% java Turtle
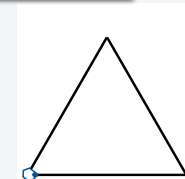
Note: Client drew triangle without computing √3

What we *expect*, once the implementation is done.

24

Turtle implementation: Instance variables and constructor

Instance variables define data-type values.

Constructors create and initialize new objects.

```
public class Turtle
{
    private double x, y;           ← instance variables
    private double angle;            are not final
    public Turtle(double x0, double y0, double a0)
    {
        x = x0;
        y = y0;
        angle = a0;
    }
...
}
```

| instance variables |
| constructor |
| methods |
| test client |

Values

position (x, y)	(.5, .5)	(.75, .75)	(.22, .12)
orientation	90°	135°	10°

25

Turtle implementation: Methods

Methods define data-type operations (implement APIs).

```
public class Turtle
{
...
    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);
    }
...
}
```

| instance variables |
| constructors |
| methods |
| test client |

API

public class Turtle	
Turtle(double x0, double y0, double q0)	
void turnLeft(double delta)	
void goForward(double step)	

$(x_0 + d \cos \alpha, \ y_0 + d \sin \alpha)$

$d \sin \alpha$

(x_0, y_0) $d \cos \alpha$
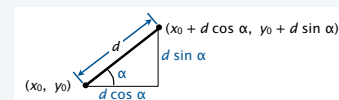
26

Turtle implementation

text file named
Turtle.java

```
public class Turtle
{
    private double x, y;                    ← instance variables
    private double angle;
    public Turtle(double x0, double y0, double a0)
    {
        x = x0;
        y = y0;                             ← constructor
        angle = a0;
    }
    public void turnLeft(double delta)
    { angle += delta; }
    public void goForward(double d)
    {
        double oldx = x;
        double oldy = y;                    ← methods
        x += d * Math.cos(Math.toRadians(angle));
        y += d * Math.sin(Math.toRadians(angle));
        StdDraw.line(oldx, oldy, x, y);
    }
    public static void main(String[] args)
    {
        Turtle turtle = new Turtle(0.0, 0.0, 0.0);
        turtle.goForward(1.0); turtle.turnLeft(120.0);   ← test client
        turtle.goForward(1.0); turtle.turnLeft(120.0);
        turtle.goForward(1.0); turtle.turnLeft(120.0);
    }
}
```

% java Turtle
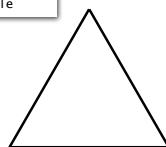
27

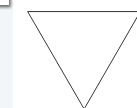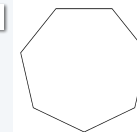
Turtle client: N-gon

```
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);
        }
    }
}
```
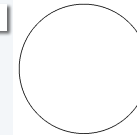
% java Ngon 3

% java Ngon 7

% java Ngon 1440

28

Turtle client: 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);
      }
   }
}
```

% java Spiral 3 1.2

% java Spiral 7 1.2

% java Spiral 1440 1.0004

Spira Mirabilis in the wild
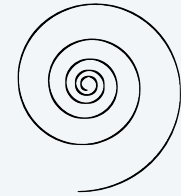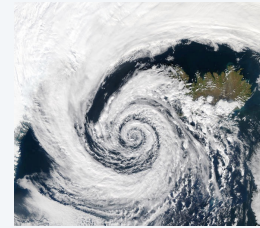
Pop quiz 1 on OOP

Q. Fix the serious bug in this code:

```
public class Turtle
{
   private double x, y;
   private double angle;

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

Pop quiz 1 on OOP

Q. Fix the serious bug in this code:

```
public class Turtle
{
   private double x, y;
   private double angle;

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

A. Remove type declarations. They create local variables, which are *different* from the instance variables!

Object-oriented programmers pledge. "I *will not* shadow instance variables"

Every programmer makes this mistake, and it is a difficult one to detect.

Image sources

http://web.media.mit.edu/~papert/
http://en.wikipedia.org/wiki/Logarithmic_spiral
http://en.wikipedia.org/wiki/Logarithmic_spiral#/media/File:Nautilus_Cutaway_with_Logarithmic_Spiral.png
http://en.wikipedia.org/wiki/File:Low_pressure_system_over_Iceland.jpg

---

9. Creating Data Types

- Overview
- Point charges
- Turtle graphics
- **Complex numbers**

Crash course in complex numbers

A complex number is a number of the form $a + bi$ where a and b are real and $i \equiv \sqrt{-1}$.

Complex numbers are a *quintessential mathematical abstraction* that have been used for centuries to give insight into real-world problems not easily addressed otherwise.

Leonhard Euler
1707–1783

A. L. Cauchy
1789–1857

To perform *algebraic operations* on complex numbers, use real algebra, replace i^2 by -1 and collect terms.
- Addition example: $(3 + 4i) + (-2 + 3i) = 1 + 7i$.
- Multiplication example: $(3 + 4i) \times (-2 + 3i) = -18 + i$.

Example: $|3 + 4i| = 5$

The *magnitude* or *absolute value* of a complex number $a + bi$ is $|a + bi| = \sqrt{a^2 + b^2}$.

Applications: Signal processing, control theory, quantum mechanics, analysis of algorithms...

35

ADT for complex numbers

A complex number is a number of the form $a + bi$ where a and b are real and $i \equiv \sqrt{-1}$.

An ADT allows us to write Java programs that manipulate complex numbers.

	complex number	$3 + 4i$	$-2 + 2i$
Values	real part	3.0	−2.0
	imaginary part	4.0	2.0

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*

API (operations)

36

Slide 37

Complex number data type implementation: Test client

Best practice. Begin by implementing a simple test client.

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

```
% java Complex
a = 3.0 + 4.0i
b = -2.0 + 3.0i
a * b = -18.0 + 1.0i
```

What we *expect*, once the implementation is done.

instance variables
constructors
methods
test client

Slide 38

Complex number data type implementation: Instance variables and constructor

Instance variables define data-type values.

Constructors create and initialize new objects.

```
public class Complex
{
    private final double re;      ← instance variables
    private final double im;         are final

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

instance variables
constructor
methods
test client
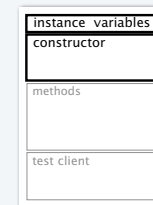
complex number	$3 + 4i$	$-2 + 2i$
real part	3.0	−2.0
imaginary part	4.0	2.0

Values

Slide 39

Complex number data type implementation: Methods

Methods define data-type operations (implement APIs).

```
public class Complex
{
 ...
    public Complex plus(Complex b)
    {
        double real = re + b.re;
        double imag = im + b.im;
        return new Complex(real, imag);
    }
    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);
    }
    public double abs()
    {  return Math.sqrt(re*re + im*im);  }

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

might also write "this.re" or use Complex a = this

Java keyword "this" is a reference to "this object" and is implicit when an instance variable is directly referenced

$a = v + wi$
$b = x + yi$
$a \times b = vx + vyi + wxi + wyi^2$
$\quad = vx - wy + (vy + wx)i$

API

instance variables
constructors
methods
test client

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*

Slide 40

Complex number data type implementation

text file named Complex.java

```
public class Complex
{
    private final double re;              ← instance variables
    private final double im;

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

    public Complex plus(Complex b)
    {
        double real = re + b.re;
        double imag = im + b.im;
        return new Complex(real, imag);
    }
    public Complex times(Complex b)           ← methods
    {
        double real = re * b.re - im * b.im;
        double imag = re * b.im + im * b.re;
        return new Complex(real, imag);
    }
    public double abs()
    {  return Math.sqrt(re*re + im*im);  }
    public String toString()
    {  return re + " + " + im + "i";  }

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

```
% java Complex
a = 3.0 + 4.0i
b = -2.0 + 3.0i
a * b = -18.0 + 1.0i
```

The Mandelbrot set

The *Mandelbrot set* is a set of complex numbers.
- Represent each complex number $x + yi$ by a point (x, y) in the plane.
- If a point is *in* the set, we color it BLACK.
- If a point is *not* in the set, we color it WHITE.
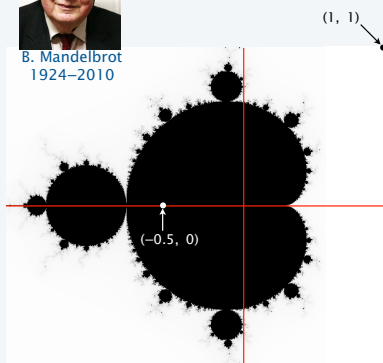
B. Mandelbrot
1924–2010

(1, 1)

(−0.5, 0)

Examples
- *In* the set: $-0.5 + 0i$.
- *Not* in the set: $1 + i$.

Challenge
- No simple formula exists for testing whether a number is in the set.
- Instead, the set is defined by an *algorithm*.

Determining whether a point is in the Mandelbrot set

Is a complex number z_0 in the set?
- Iterate $z_{t+1} = (z_t)^2 + z_0$.
- If $|z_t|$ *diverges to infinity*, z_0 is *not* in the set.
- If not, z_0 is *in* the set.

(1, 1)

(−0.5, 0)

t	z_t	t	z_t
0	$-1/2 + 0i$	0	$1 + i$
1	$-1/4 + 0i$	1	$1 + 3i$
2	$-7/16 + 0i$	2	$-7 + 7i$
3	$-79/256 + 0i$	3	$1 - 97i$
4	$-26527/65536 + 0i$	4	$-9407 - 193i$

always between $-1/2$ and 0
$z = -1/2 + 0i$ is *in* the set

diverges to infinity
$z = 1 + i$ is *not* in the set
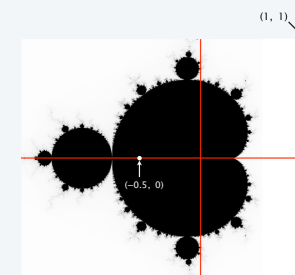
$(1+i)^2 + (1+i) = 1 + 2i + i^2 + 1 + i = 1+3i$

$(1+3i)^2 + (1+i) = 1 + 6i + 9i^2 + 1 + i = -7+7i$

Plotting the Mandelbrot set

Practical issues
- Cannot plot infinitely many points.
- Cannot iterate infinitely many times.

Approximate solution for first issue
- Sample from an N-by-N grid of points in the plane.
- Zoom in to see more detail (stay tuned!).

Approximate solution for second issue
- Fact: if $|z_t| > 2$ for any t, then z *is not* in the set.
- Pseudo-fact: if $|z_{255}| \leq 2$ then z is "likely" in the set.

(1, 1)

(−0.5, 0)
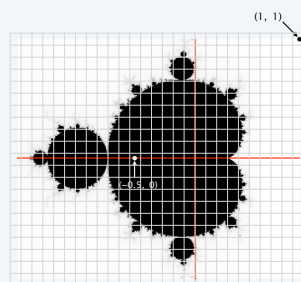
Important note: Solutions imply significant computation.

Complex number client: Mandelbrot set visualization (helper method)

Mandelbrot function of a complex number.
- Returns WHITE if the number is not in the set.
- Returns BLACK if the number is (probably) in the set.

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

For a more dramatic picture,
return new Color(255-t, 255-t, 255-t)
or colors picked from a color table.

Complex number client: Mandelbrot set visualization

```java
import java.awt.Color;
public class Mandelbrot
{
    public static Color mand(Complex z0)
    { // See previous slide. }
    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    = Integer.parseInt(args[3]);
        Picture pic = new Picture(N, N);

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

scale to screen coordinates

(0, 0) is upper left corner

% java Mandelbrot -.5 0 2 32

Mandelbrot Set

% java GrayscaleMandelbrot -.5 0 2
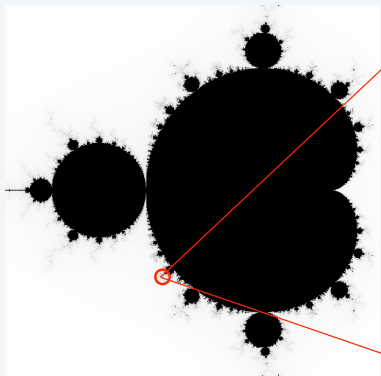
% java GrayscaleMandelbrot .1045 -.637 .01
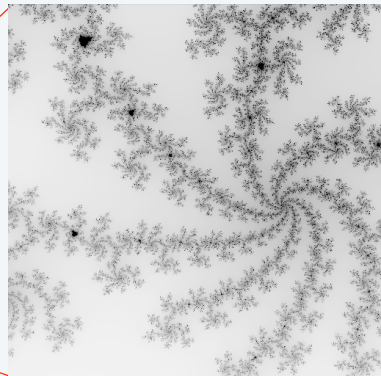
Mandelbrot Set

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

color map

-1.5 0 2

-1.5 0 .002
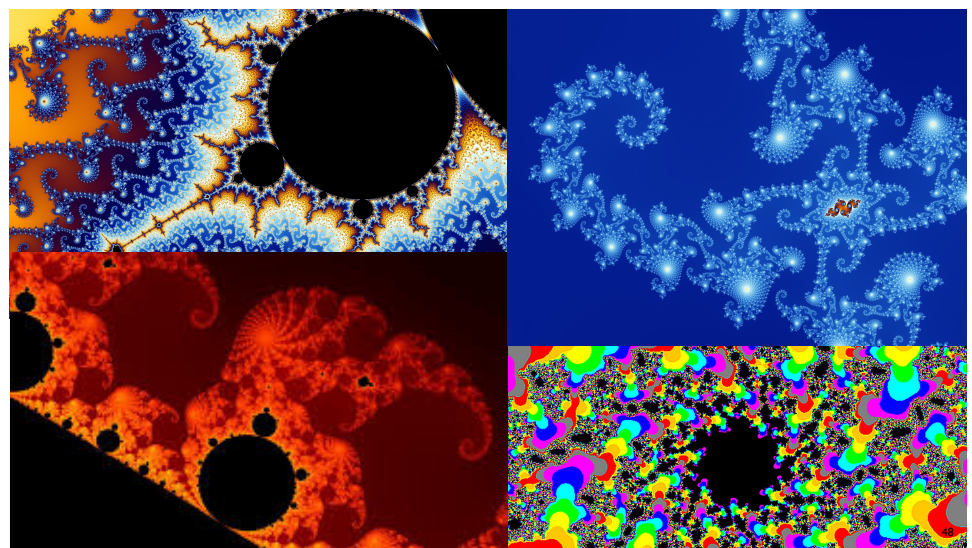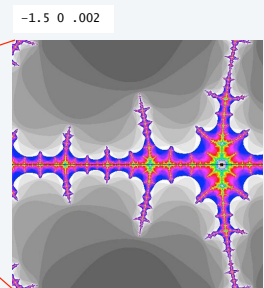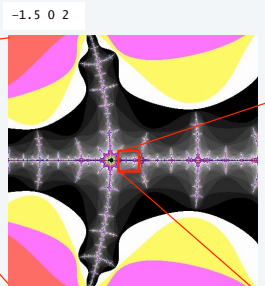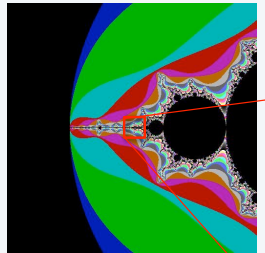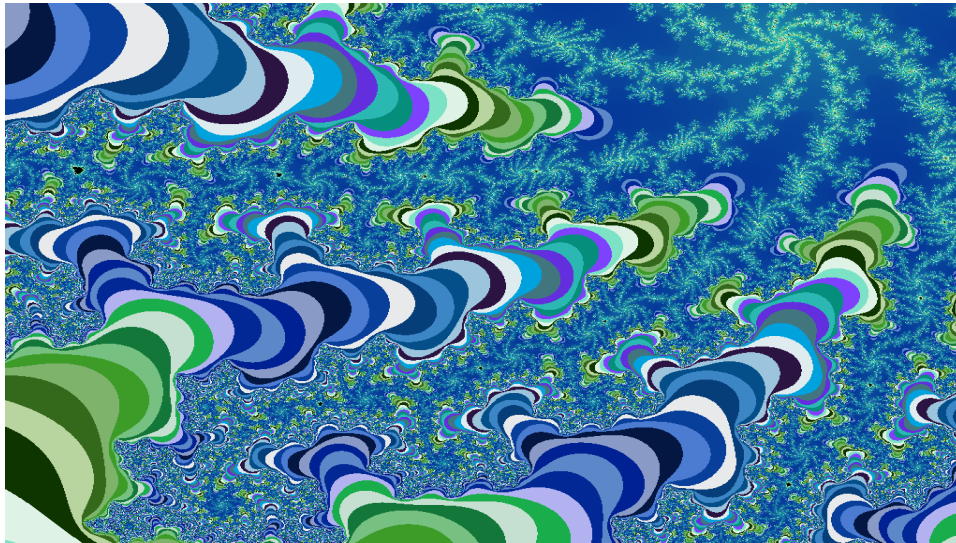
OOP summary

Object-oriented programming (OOP)
- Create your own data types (sets of values and ops on them).
- Use them in your programs (manipulate *objects*).

OOP helps us simulate the physical world
- Java objects model real-world objects.
- Not always easy to make model reflect reality.
- Examples: charged particle, color, sound, genome....
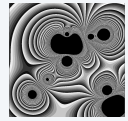
OOP helps us extend the Java language
- Java doesn't have a data type for every possible application.
- Data types enable us to add our own abstractions.
- Examples: complex, vector, polynomial, matrix, picture....

T A G A T G **T G C** T A G C

50

You have come a long way

any program you might want to write

objects

```
public class HelloWorld
{
  public static void main(String[] args)
  {
    System.out.println("Hello, World");
  }
}
```

functions and modules

graphics, sound, and image I/O

arrays

T A G A T G **T G C** T A G C

conditionals and loops

Math   text I/O

primitive data types   assignment statements
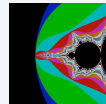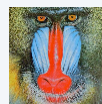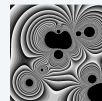
Course goal. Open a *whole new world* of opportunity for you (programming). ✔

51

C OMPUTER
S CIENCE
An Interdisciplinary Approach

Section 3.2

ROBERT SEDGEWICK
KEVIN WAYNE

http://introcs.cs.princeton.edu

9. Creating Data Types