3.2 Creating Data Types
Data Types

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

**Basic types.**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Set of Values</th>
<th>Some Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>true, false</td>
<td>not, and, or, xor</td>
</tr>
<tr>
<td>int</td>
<td>-2(^{31}) to 2(^{31} - 1)</td>
<td>add, subtract, multiply</td>
</tr>
<tr>
<td>String</td>
<td>sequence of Unicode characters</td>
<td>concatenate, compare</td>
</tr>
</tbody>
</table>

**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, we must specify:
• Set of values.
• Operations defined on them.

Java class. Allows us to define data types by specifying:
• Instance variables. (set of values)
• Methods. (operations defined on them)
• 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.

\[
V = k \frac{q}{r}
\]

\(r = \text{distance between } (x, y) \text{ and } (r_x, r_y)\)

\(k = \text{electrostatic constant} = 8.99 \times 10^9 \text{ N} \cdot \text{m}^2 / \text{C}^2\)
Point Charge Data Type

Goal. Create a data type to manipulate point charges.

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

API:

```java
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
}
```
Client program. Uses data type operations to calculate something.

```java
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); // automagically invokes the toString() method
    StdOut.println(c2);
    StdOut.println(v1 + v2);
}
```

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

```
public class Charge {
    private final double rx, ry;
    private final double q;
}
```
**Anatomy of a Constructor**

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

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

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

```java
Charge c1 = new Charge(.51, .63, 21.3);
Charge c2 = new Charge(.13, .94, 81.9);
```
Anatomy of an Instance Method

**Method.** Define operations on instance variables.

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

**Invoking a method.** Use dot operator to invoke a method in client code.

```
double v1 = c1.potentialAt(x, y);
double v2 = c2.potentialAt(x, y);
```
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));
    }
}
Potential visualization. Read in N point charges from a file; compute total potential at each point in unit square. (And make a Picture.)

```
% 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
```

[Image of potential visualization diagram]
Arrays of objects. Allocate memory for the array; then allocate memory for each individual object.

```java
// 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);
}
```
/ Plot the data.
int SIZE = 512;
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 i = 0; i < N; i++)
        {
            double x = 1.0 * col / SIZE;
            double y = 1.0 * row / SIZE;
            V += a[i].potentialAt(x, y);
        }
        Color color = getColor(V);  // Arbitrary double-Color map.
        pic.set(col, SIZE-1-row, color);
    }
pic.show();
Fix the serious bug in the following code.

```java
public class Charge {
    private double rx, ry;
    private double q;
    public Charge (double x0, double y0, double q0) {
        double rx = x0;
        double ry = y0;
        double q = q0;
    }
}
```
Turtle Graphics
Goal. Create a data type to manipulate a turtle moving in the plane.

Set of values. Location and orientation of turtle.

API.

```java
public class Turtle

    Turtle(double x0, double y0, double a0)  // create a new turtle at (x0, y0) facing a0 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);
    }
}
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);
        }
    }
}
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);
        }
    }
}
Spira Mirabilis in Nature
Complex Numbers

Be rational

Get real.

i π
**Complex Number Data Type**

**Goal.** Create a data type to manipulate complex numbers.

**Set of values.** Two real numbers: real and imaginary parts.

**API.**

```java
public class Complex {
    // Constructor
    Complex(double real, double imag)

    // Methods
    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
}
```

\[
\begin{align*}
a &= 3 + 4i, \\
b &= -2 + 3i, \\
a + b &= 1 + 7i, \\
a \times b &= -18 + i, \\
|a| &= 5
\end{align*}
\]
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.

```java
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);
}
```

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

Remark. Can't write \( a = b \times c \) since no operator overloading in 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 String toString() {    // methods
        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;
        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);
    }
}
Mandelbrot Set

Mandelbrot set. A particular set of complex numbers.

Plot. Plot \((x, y)\) black if \(z = x + y \cdot i\) is in the set, and white otherwise.

- No simple formula describes which complex numbers are in set.
- Instead, describe using an algorithm.
Mandelbrot set. Is complex number $z_0$ in set?

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

<table>
<thead>
<tr>
<th>$t$</th>
<th>$Z_t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$-\frac{1}{2} + 0i$</td>
</tr>
<tr>
<td>1</td>
<td>$-\frac{1}{4} + 0i$</td>
</tr>
<tr>
<td>2</td>
<td>$-\frac{7}{16} + 0i$</td>
</tr>
<tr>
<td>3</td>
<td>$-\frac{79}{256} + 0i$</td>
</tr>
<tr>
<td>4</td>
<td>$-\frac{26527}{65536} + 0i$</td>
</tr>
<tr>
<td>5</td>
<td>$-\frac{1443801919}{4294967296} + 0i$</td>
</tr>
</tbody>
</table>

$z = -\frac{1}{2} + 0i$ is in Mandelbrot set

<table>
<thead>
<tr>
<th>$t$</th>
<th>$Z_t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$1 + i$</td>
</tr>
<tr>
<td>1</td>
<td>$1 + 3i$</td>
</tr>
<tr>
<td>2</td>
<td>$-7 + 7i$</td>
</tr>
<tr>
<td>3</td>
<td>$1 - 97i$</td>
</tr>
<tr>
<td>4</td>
<td>$-9407 - 193i$</td>
</tr>
<tr>
<td>5</td>
<td>$88454401 + 3631103i$</td>
</tr>
</tbody>
</table>

$z = 1 + i$ is 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 $z$ in the Mandelbrot set?
• Returns white (definitely no) or black (probably yes).

```java
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;
}
```

More dramatic picture: replace `Color.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.

```java
class Complex {
    double real, imag;
    public Complex(double r, double i) {
        real = r; imag = i;
    }
    public Complex add(Complex c) {
        return new Complex(real + c.real, imag + c.imag);
    }
    public Complex multiply(Complex c) {
        return new Complex(real * c.real - imag * c.imag, real * c.imag + imag * c.real);
    }
    public boolean equals(Complex c) {
        return Math.abs(real - c.real) < 1e-6 && Math.abs(imag - c.imag) < 1e-6;
    }
}
```

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

% java Mandelbrot -.5 0 2

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

% java ColorMandelbrot -1.5 0 2 < mandel.txt