3.2 Creating Data Types

Data Types

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

**Built-in “primitive” 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>any sequence of 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.

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 $(x_0, y_0)$ with electric charge $q$.
- Determine electric potential at $(x, y)$ due to point charge.
- Convert to string.

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

$r$ = distance between $(x, y)$ and $(x_0, y_0)$
$k$ = electrostatic constant = $8.99 \times 10^9$ N m$^2$/C$^2$
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);
    System.out.println(c1);
    System.out.println(c2);
    System.out.println(v1 + v2);
}
```

% java Charge .50 .50
21.3 at (0.51, 0.63)
81.0 at (0.13, 0.94)
2.74936907085912E12

Anatomy of a Constructor

Constructor. Invoke with new to create new objects.

Anatomy of Instance Variables

Instance variables. Specifies the set of values.
- Declare outside any method.
- Always use access modifier private. [stay tuned]

```
public class Charge() {
    private double rx;
    private double ry;
    private double q;
    ...
}
```

Anatomy of a Data Type Method

Method. Define operations on instance variables.

```
Charge c1 = new Charge(.51, .63, 21.3);
Charge c2 = new Charge(.13, .94, 81.9);
```
Arrays of objects. Allocate memory for the array; then allocate memory for each individual object.

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

Potential Visualization

Read in data.

```
int N = StdIn.readInt();
Charge[] a = new Charge[N];
for (int k = 0; k < N; k++) {
    double x0 = StdIn.readDouble();
    double y0 = StdIn.readDouble();
    double q0 = StdIn.readDouble();
    a[k] = new Charge(x0, y0, q0);
}
```

Arrays of objects. Allocate memory for the array; then allocate memory for each individual object.
Applications of Complex Numbers

Complex numbers are 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

Goal. Create a data type to manipulate complex numbers.

Set of values. Two real numbers. [real and imaginary parts]

Set of operations.
- Plus, times, absolute value.
- Subtract, divide, power, phase, ...
- Convert to string.

Ex. \( a = 3 + 4i, b = -2 + 3i \).
- \( a + b = 1 + 7i \).
- \( a \times b = -18 + i \).
- \(|a| = 5\).

Complex Number Data Type: Implementation

```java
public class Complex {
    private double re;
    private double im;

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

    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;
        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 set of complex numbers.
Plot. Plot (x, y) black if z = x + iy is in the set, and white otherwise.

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

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 |zₜ| ≥ 2 for any t, then z not in Mandelbrot set.
- Pseudo-fact: if |z₂₅₅| < 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).
- 7 lines of code with judicious use of data types.

```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);
        z = z² + z₀;
    }
    return Color.BLACK;
}
```

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

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

**Complex Number Data Type: Another Client**

Plot the Mandelbrot set in gray scale.

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

**Mandelbrot Set**

Goal. Create a data type to manipulate spatial vectors.

Set of values. Sequence of real numbers. [Cartesian coordinates]

Set of operations.
- Magnitude, direction.
- Plus, times, dot product, ...
- Convert to string.

Ex. $x = (0, 3, 4, 0), y = (0, -3, 1, -4)$.
- $x + y = (0, 0, 5, -4)$.
- $3x = (0, 9, 12, 0)$.
- $x \cdot y = (0 \cdot 0) + (3 \cdot -3) + (4 \cdot 1) + (0 \cdot -4) = -5$.
- $|x| = (0^2 + 3^2 + 4^2 + 0^2)^{1/2} = 5$.
- $\hat{x} = x / |x| = (0, 0.6, 0.8, 0)$. 
Vector Data Type Applications

Vectors are a quintessential mathematical abstraction.

Applications.
- Statistics.
- Linear algebra.
- Clustering and similarity search.
- Force, velocity, acceleration, momentum, torque.
- ... 

Vector Data Type: Implementation

```java
public class Vector {
    private int N;
    private double[] coords;  
    
    public Vector(double[] a) {
        N = a.length;
        coords = new double[N];
        for (int i = 0; i < N; i++)
            coords[i] = a[i];
    }
    
    public double dot(Vector b) {
        double sum = 0.0;
        for (int i = 0; i < N; i++)
            sum += coords[i] * b.coords[i];
        return sum;
    }
    
    public Vector plus(Vector b) {
        double[] c = new double[N];
        for (int i = 0; i < N; i++)
            c[i] = coords[i] + b.coords[i];
        return new Vector(c);
    }
    
    public Vector times(double t) {
        double[] c = new double[N];
        for (int i = 0; i < N; i++)
            c[i] = t * coords[i];
        return new Vector(c);
    }
    
    public double magnitude() {
        return Math.sqrt(this.dot(this));
    }
    
    public Vector direction() {
        return this.times(1.0 / this.magnitude());
    }
    
    ... 
}
```

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

Simulating the physical world.
- Java objects model real-world objects.
- Ex: point charge, COS 126 student.
- Not always easy to make model reflect reality.

Extending the Java language.
- Java doesn’t have a data type for every possible application.
- Data types enable us to add our own abstractions.
- Scientific applications: complex, vector, polynomial, matrix, ....

This. The keyword this is a reference to the invoking object.

Ex. When you invoke a.magnitude(), this is an alias for a.