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, define:
• 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.

\[ V = \frac{kq}{r} \]

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

**Goal.** Create a data type to manipulate point charges.

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

**API:**

- **public class Charge**
  - `Charge(double x0, double y0, double q0)`: Constructor specifying what happens when you create a new object.
  - `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);
    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
```

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.
  - makes data type abstract
  - makes objects immutable (stay tuned)

```java
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.

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

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

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

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

```java
public double potentialAt(double x, double y)
{
    double k = 8.99e99;
    double dx = x - rx;
    double dy = y - ry;
    return k * q / Math.sqrt(dx*dx + dy*dy);
}
```

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

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

Anatomy of a Class

```java
public class Charge
{
    // instance variables
    private final double rx, ry;
    private final double q;

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

    // instance methods
    public double potentialAt(double x, double y)
    { double k = 8.99e99;
        double dx = x - rx;
        double dy = y - ry;
        return k * q / Math.sqrt(dx*dx + dy*dy);
    }
}
```

Charge Client Example: Potential Visualization

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

```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);
}
```

**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);
}
```
Potential Visualization

// 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();

Data Type Challenge

Fix the serious bug in the following code.

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. 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 Implementation

```java
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);
    }
}
```

Turtle client example: N-gon

```java
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 1.0
3 1.2
1440 1.00004
1440 1.0004
```

Turtle client example: Spira Mirabilis

```java
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

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

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

**API.**

public class Complex

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complex(double real, double imag)</td>
<td>sum of this number and b</td>
</tr>
<tr>
<td>Complex plus(Complex b) &amp; Complex times(Complex b)</td>
<td>product of this number and b</td>
</tr>
<tr>
<td>double abs()</td>
<td>magnitude</td>
</tr>
<tr>
<td>String toString()</td>
<td>string representation</td>
</tr>
</tbody>
</table>

**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.
- ...

**Remark.** Can’t write \( a = b \times c \) since no operator overloading in Java.

**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
```

% java TestClient -c.toString()

```
result of c.toString()
```
Complex Number Data Type: Implementation

```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() {  // returns the absolute value
        return Math.sqrt(re*re + im*im);
    }

    public Complex plus(Complex b) {  // ref to b's instance variables
        double real = re + b.re;
        double imag = im + b.im;
        return new Complex(real, imag);
    }

    public Complex times(Complex b) {  // ref to b's instance variables
        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 \(z = x + y \, i\) 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>-1/2 + 0i</td>
</tr>
<tr>
<td>1</td>
<td>-1/4 + 0i</td>
</tr>
<tr>
<td>2</td>
<td>-7/16 + 0i</td>
</tr>
<tr>
<td>3</td>
<td>-79/256 + 0i</td>
</tr>
<tr>
<td>4</td>
<td>-26527/65536 + 0i</td>
</tr>
<tr>
<td></td>
<td>-1431601136/629145024 + 0i</td>
</tr>
</tbody>
</table>

\(z = -1/2\) 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>

\(z = 1 + i\) not in 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 \(i\), then \(z\) not in Mandelbrot set.
- Pseudo-fact: if \(|z_{255}| \approx 2\) then \(z\) "likely" in Mandelbrot set.

Plotting the Mandelbrot Set

- Sample from an \(N\)-by-\(N\) grid of points in the plane.
- Fact: if \(|z_t| > 2\) for any \(i\), then \(z\) not in Mandelbrot set.
- Pseudo-fact: if \(|z_{255}| \approx 2\) then \(z\) "likely" in Mandelbrot set.
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 \( \text{Color.WHITE} \) with grayscale or color.

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 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();
}
```
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, ....