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, 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 (rx, ry) 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 (rx, ry) and (x, y)

k = electrostatic constant = \(8.99 \times 10^9\) 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.

```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
```

Point Charge Data Type: A Simple Client

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.

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

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;
}
```

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

```java
Charge c1 = new Charge(51.63, 21.33);
Charge c2 = new Charge(13.98, 81.93);
```

Anatomy of an Instance Method

Instance method. Define operations on instance variables.

```java
public double potentialAt(double x, double y) {
    double k = 8.99e9; // constant variable
    double dx = x - rx;
    double dy = y - ry;
    return k * q / Math.sqrt(dx * dx + dy * dy); // call on a static method
}
```

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

```java
double v1 = c1.potentialAt(x0, y0);
double v2 = c2.potentialAt(x0, y0);
```

Potential Visualization

Potential visualization. Read in N point charges from standard input; 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 with `new`, then allocate memory for each individual object with `new`.

```java
// read in charges.txt
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

```java
// 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 = 0.5;
        for (int k = 0; k < SIZE; k++) {
            double x = i * 1.0 / SIZE;
            double y = j * 1.0 / SIZE;
            V += x[k];
            color = getColor(V);
            pic.set(i, SIZE-j, color);
        }
    }
}
pic.show();
```

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 {
   public Turtle(double x0, double y0, double dx, double dy) {
   // create a new turtle at (x0, y0) facing dx
degrees counterclockwise from the x-axis
   }
   public void turnLeft(double delta) {
   // turn the turtle delta degrees counterclockwise
   }
   public void goForward(double step) {
   // advance the turtle step distance, drawing a line
   }
   public String toString() {
   // return a string representation of the turtle
   }
```

- `public class Spiral {
   public static void main(String[] args) {
   double angle = 360.0 / N;
   double step = Math.sin(Math.toRadians(angle/2));
   Turtle turtle = new Turtle(0.0, 0.0, angle/2);
   for (int i = 0; i < N; i++) {
   turtle.goForward(step);
   turtle.turnLeft(angle);
   }
   }
   ```

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));
   Turtle turtle = new Turtle(0.0, 0.0, angle/2);
   for (int i = 0; i < N; i++) {
   turtle.goForward(step);
   turtle.turnLeft(angle);
   }
   }
   ```

Spira Mirabilis

```java
public class SpiraMirabilis {
   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));
   Turtle turtle = new Turtle(0.0, 0.0, angle/2);
   for (int i = 0; i < N; i++) {
   step *= decay;
   turtle.goForward(step);
   turtle.turnLeft(angle);
   }
   }
   ```
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 Complex(double real, double imag)`
- `Complex plus(Complex b)`
- `Complex times(Complex b)`
- `double abs()`
- `String toString()`

- `a = 3 + 4i`
- `b = -2 + 3i`
- `a + b = 1 + 7i`
- `a * 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.

```java
public class Complex {
    private final double re;
    private final 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);
    }
}
```

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

Complex Number Data Type: Implementation
Mandelbrot Set

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

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

Plot 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 $|z| > 2$ for any $t$, then $z$ not in Mandelbrot set.
- Pseudo-fact: if $|z| \leq 2$ then "likely" in Mandelbrot set.

Mandelbrot function with complex numbers.
- Is $z_0$ in the Mandelbrot set?
- Returns white (definitely no) or black (probably yes).

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 = (int) size;
    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();
}
```
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**

```
java ColorMandelbrot -X X X X mandel.txt
```

**Mandelbrot Set Music Video**

[Video Link](http://www.jonathancoulton.com/songdetails/Mandelbrot Set)

**3.2 Extra Slides**
Object reference.  
- Allow client to manipulate an object as a single entity.  
- Essentially a machine address (pointer).

```java
public class Ball {
    private double vx, vy;  
    private double radius;  

    public Ball() {  
        vx = 0.015 * Math.random() * 0.03;  
        vy = 0.015 * Math.random() * 0.03;  
        radius = 0.01 + Math.random() * 0.005;  
    }

    public void move() {  
        if (vx > 0) { vx = -vx; }  
        if (vy > 0) { vy = -vy; }  
        ry = ry + vy;  
        rx = rx + vx;  
    }

    public void draw() {  
        StdDraw.filledCircle(rx, ry, radius);  
    }
}
```

**Example: Bouncing Ball in Unit Square**

Bouncing ball.  Model a bouncing ball moving in the unit square with constant velocity.

```java
Ball b1 = new Ball();  
b1.move();

Ball b2 = new Ball();  
b2.move();

b2 = b1;  
b2.move();
```
Creating Many Objects

Each object is a data type value.
• Use new to invoke constructor and create each one.
  • Ex: create N bouncing balls and animate them.

```java
public class BouncingBalls {
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        Ball[] balls = new Ball[N];
        for (int i = 0; i < N; i++) {
            balls[i] = new Ball();
        }

        while (true) {
            StdDraw.clear();
            for (int i = 0; i < N; i++) {
                balls[i].move();
                balls[i].draw();
            }
            StdDraw.show(50);
    }
}
```
OOP Context

Reference. Variable that stores the name of a thing.

<table>
<thead>
<tr>
<th>Thing</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web page</td>
<td><a href="http://www.princeton.edu">www.princeton.edu</a></td>
</tr>
<tr>
<td>Bit size</td>
<td>45-205-220314774</td>
</tr>
<tr>
<td>World of TOY memory</td>
<td>xC</td>
</tr>
<tr>
<td>Bytes of computer memory</td>
<td>851262</td>
</tr>
<tr>
<td>Home</td>
<td>35 Olden Street</td>
</tr>
</tbody>
</table>

Some consequences.
- Assignment statements copy references (not objects).
- The == operator tests if two references refer to same object.
- Pass copies of references (not objects) to functions.
- Efficient since no copying of data
- Function can change the object

Pass-by-Value

Arguments to methods are always passed by value.
- Primitive types: passes copy of value of actual parameter.
- Objects: passes copy of reference to actual parameter.