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 \((r_x, r_y)\) with electric charge \(q\).
- Determine electric potential \(V\) at \((x, y)\) due to point charge.
- Convert to string.

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

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

\(k\) = electrostatic constant \(k = 8.99 \times 10^9 \text{ N} \cdot \text{m}^2/\text{C}^2\)
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
class Charge {
    double x0, y0, q0; // instance variables
    String toString();
    Charge(double x0, double y0, double q0) {
        this.x0 = x0;
        this.y0 = y0;
        this.q0 = q0;
    }
    double potentialAt(double x, double y) {
        return k * q0 / Math.sqrt(Math.pow(x - x0, 2) + Math.pow(y - y0, 2));
    }
}
```

**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(x + 0.5, y + 0.5, 21.3);
    Charge c2 = new Charge(x + 0.1, y + 0.9, 81.9);
    double v1 = c1.potentialAt(x, y);
    double v2 = c2.potentialAt(x, y);
    StdOut.println(c1 + 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 x0, y0; // instance variable declaration
    private final double q0;

    public Charge(double x0, double y0, double q0) {
        this.x0 = x0;
        this.y0 = y0;
        this.q0 = q0;
    }

    public String toString() {
        return new String();
    }
}
```

**Anatomy of a Constructor**

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

```java
public Charge(double x0, double y0, double q0) {
    this.x0 = x0;
    this.y0 = y0;
    this.q0 = q0;
}
```

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

```java
Charge c1 = new Charge(.5, .63, 21.3);
Charge c2 = new Charge(.13, .94, 81.9);
```
Anatomy of an Instance Method

**Instance method.** Define operations on instance variables.

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

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

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

---

Anatomy of a Class

```java
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 potential(double x, double y)
{
    double k = 8.99e9;
    double dx = x - rx;
    double dy = y - ry;
    return k * q / Math.sqrt(dx*dx + dy*dy);
}
```

---

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
Arrays of objects:
- Allocate memory for the array with `new`;
- Allocate memory for each individual object within the array with `new`.
```

---

Potential Visualization

Read N point charges from standard input; compute total potential at each point in unit square.
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 = 0.0;
        for (int k = 0; k < N; k++) {
            double x = 1.0 * i / SIZE;
            double y = 1.0 * j / SIZE;
            V += a[k].potentialAt(x, y);
        }
        Color color = getColor(V);
        pic.set(i, SIZE-1-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 {
    private double x, y; // turtle is at (x, y)
    private double angle; // facing this direction
    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 Graphics

public class Turtle {
    private double x, y; // turtle is at (x, y)
    private double angle; // facing this direction
    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
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 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
```

```
a = 3 + 4i, b = -2 + 3i
a + b = 1 + 7i
a \times b = -18 + 1i
|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.

```
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.plus(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 c = a * b since no operator overloading in Java.

Complex Number Data Type: Implementation

```
public class Complex {

private final double re;
private final double im;

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

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);
}
```

```
result of a.toString()  result of c.toString()  
```
Mandelbrot Set

Mandelbrot set. A 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.

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 \(z_0\) is not in Mandelbrot set.
- Pseudo-fact: if \(|z_{255}| \leq 2\) then \(z_0\) "likely" in Mandelbrot set.

Complex Number Data Type: Another Client

Mandelbrot function with complex numbers.
- Is \(z_0\) 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 StdDraw.WHITE;
        z = z.times(z);
        z = z.plus(z0);
    } return StdDraw.BLACK;
}
```

More dramatic picture: replace `StdDraw.WHITE` with grayscale or color.
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 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();
}
```

- Java Mandelbrot -.5 0 2
- Java Mandelbrot .1045 -.637 .01
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, ....

3.2 Extra Slides
Example: Bouncing Ball in Unit Square

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

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

    public Ball() {
        rx = ry = 0.5;
        vx = vy = Math.random() * 0.03;
        radius = 0.01 + Math.random() * 0.01;
    }

    public void move() {
        if ((rx + vx > 1.0) || (rx + vx < 0.0)) vx = -vx;
        if ((ry + vy > 1.0) || (ry + vy < 0.0)) vy = -vy;
        rx = rx + vx;
        ry = ry + vy;
    }

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

Object References

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

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

Ball b2 = new Ball();
b2.move();
b2 = b1;
b2.move();
```

```
<table>
<thead>
<tr>
<th>addr</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>0</td>
</tr>
<tr>
<td>C1</td>
<td>0</td>
</tr>
<tr>
<td>C2</td>
<td>0</td>
</tr>
<tr>
<td>C3</td>
<td>0</td>
</tr>
<tr>
<td>C4</td>
<td>0</td>
</tr>
<tr>
<td>C5</td>
<td>0</td>
</tr>
<tr>
<td>C6</td>
<td>0</td>
</tr>
<tr>
<td>C7</td>
<td>0</td>
</tr>
<tr>
<td>C8</td>
<td>0</td>
</tr>
<tr>
<td>C9</td>
<td>0</td>
</tr>
<tr>
<td>CA</td>
<td>0</td>
</tr>
<tr>
<td>CB</td>
<td>0</td>
</tr>
<tr>
<td>CC</td>
<td>0</td>
</tr>
</tbody>
</table>
```

main memory (64-bit machine)

```
<table>
<thead>
<tr>
<th>addr</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>0.50</td>
</tr>
<tr>
<td>C1</td>
<td>0.50</td>
</tr>
<tr>
<td>C2</td>
<td>0.05</td>
</tr>
<tr>
<td>C3</td>
<td>0.01</td>
</tr>
<tr>
<td>C4</td>
<td>0.01</td>
</tr>
<tr>
<td>C5</td>
<td>0</td>
</tr>
<tr>
<td>C6</td>
<td>0</td>
</tr>
<tr>
<td>C7</td>
<td>0</td>
</tr>
<tr>
<td>C8</td>
<td>0</td>
</tr>
<tr>
<td>C9</td>
<td>0</td>
</tr>
<tr>
<td>CA</td>
<td>0</td>
</tr>
<tr>
<td>CB</td>
<td>0</td>
</tr>
<tr>
<td>CC</td>
<td>0</td>
</tr>
</tbody>
</table>
```

registers

main memory (64-bit machine)
Object References

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

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

Ball b2 = new Ball();
b2.move();
b2 = b1;
b2.move();
```

```
main memory (64-bit machine)

C0 0.60
C1 0.52
C2 0.05
C3 0.01
C4 0.03
C5 0
C6 0
C7 0
C8 0
C9 0
CA 0
CB 0
CC 0
```

```
C0 0.57
C1 0.54
C2 0.07
C3 0.04
C4 0.04
C5 0
C6 0
C7 0.57
C8 0.54
C9 0.07
CA 0.04
CB 0.04
CC 0
```

```
C0 0.55
C1 0.51
C2 0.05
C3 0.01
C4 0.03
C5 0
C6 0
C7 0
C8 0
C9 0
CA 0
CB 0
CC 0
```

```
C0 0.60
C1 0.52
C2 0.05
C3 0.01
C4 0.03
C5 0
C6 0
C7 0
C8 0
C9 0
CA 0
CB 0
CC 0
```

```
C0 0.55
C1 0.51
C2 0.05
C3 0.01
C4 0.03
C5 0
C6 0
C7 0
C8 0
C9 0
CA 0
CB 0
CC 0
```
**Object References**

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

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

Ball b2 = new Ball();
b2.move();
b2 = b1;
b2.move();
```

Data stored in c7 - c8 for abstract bit recycler.

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>0.60</td>
</tr>
<tr>
<td>C1</td>
<td>0.52</td>
</tr>
<tr>
<td>C2</td>
<td>0.05</td>
</tr>
<tr>
<td>C3</td>
<td>0.01</td>
</tr>
<tr>
<td>C4</td>
<td>0.03</td>
</tr>
<tr>
<td>C5</td>
<td>0</td>
</tr>
<tr>
<td>C6</td>
<td>0</td>
</tr>
<tr>
<td>C7</td>
<td>0.57</td>
</tr>
<tr>
<td>C8</td>
<td>0.54</td>
</tr>
<tr>
<td>C9</td>
<td>0.07</td>
</tr>
<tr>
<td>CA</td>
<td>0.04</td>
</tr>
<tr>
<td>CB</td>
<td>0.04</td>
</tr>
<tr>
<td>CC</td>
<td>0</td>
</tr>
</tbody>
</table>

**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();
            StdDraw.show(20);
        }
    }
}
```

50 Bouncing Balls

**Color**
- Associate a color with each ball; paint background black.

**Scientific variations**
- Account for gravity, spin, collisions, drag, …