3.2 Creating Data Types

Data Types

- Data type: Set of values and operations on those values.
- Basic types:
  - boolean
  - int
  - String

Today's Goal: 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 \((x, y)\) with electric charge \(q\).
- Determine electric potential \(V\) at \((x, y)\) due to point charge.
- Convert to string.

\[
V = \frac{kq}{r} \quad \text{where} \quad r = \sqrt{(rx - x)^2 + (ry - y)^2}
\]

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

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(0.51, 0.63, 21.3);
    Charge c2 = new Charge(0.13, 0.94, 81.9);
    double v1 = c1.potentialAt(x, y);
    double v2 = c2.potentialAt(x, y);
    StdOut.println(c1);
    StdOut.println(c2);
    StdOut.println(v1 + v2);
}
```

Output:
```
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.

```
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.
```
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.
```
Charge cl = new Charge(51.0, 63.21, 13.34);
Charge c2 = new Charge(13.3, 60.14, 81.93);
```

Anatomy of an Instance Method

Instance method. Define operations on instance variables.
```
public double potentialAt(double x, double y) {
    double k = 8.99e9; // argument variable name
    double dx = x - rx;
    double dy = y - ry;
    return k * q / Math.sqrt(dx*dx + dy*dy); // return
}
```

Invoking an instance method. Use dot operator to invoke a method.
```
double v1 = cl.potentialAt(x, y);
double v2 = cl.potentialAt(x, y);
```

Potential Visualization

Potential visualization. Read in N point charges from standard input; compute total potential at each point in unit square.
```
// read in the data
int N = Stdin.readInt();
Charge[] a = new Charge[N];
for (int i = 0; i < N; i++) {
    a[i] = Stdin.readDouble();
    double x0 = Stdin.readDouble();
    double y0 = Stdin.readDouble();
    double q0 = Stdin.readDouble();
    a[i] = new Charge(x0, y0, q0);
}
```

Potential Visualization

Arrays of objects. Allocate memory for the array with new; then allocate memory for each individual object with new.
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 = 0.6;
        for (int k = 0; k < N; k++) {
            double x = 1.0 * i / SIZE;
            double y = 1.0 * j / SIZE;
            V += a[k];
        }
        Color color = getColor(V);
        pic.set(i, 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**

```java
public class Turtle {
    // draw a square
    Turtle(turtle new Turtle(0.0, 0.0, 0.0);
    turtle goForward(double step);
    turtle turnLeft(double delta);
}
```

Turtle Graphics

```java
public class Spiral {
    // draw a square
    Spiral(turtle new Spiral(0.5, 0.5, 0.5);
    turtle goForward(double step);
    turtle turnLeft(double delta);
}
```

N-gon

```java
public class Pentagon {
    // draw a square
    Pentagon(turtle new Pentagon(0.5, 0.5, 0.5);
    turtle goForward(double step);
    turtle turnLeft(double delta);
}
```

Spira Mirabilis

```java
public class Spiralm {
    // draw a square
    Spiralm(turtle new Spiralm(0.5, 0.5, 0.5);
    turtle goForward(double step);
    turtle turnLeft(double delta);
}
```
Complex Number Data Type

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

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

**API.**

- `Complex(double real, double imag)`
  - constructor instance variables
  - creates a Complex object, and returns a reference to it
- `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

### Example

\[
\begin{align*}
\alpha &= 3 + 4i \\
\beta &= -2 + 3i \\
\alpha + \beta &= 1 + 7i \\
\alpha \times \beta &= -18 + 1i \\
|\alpha| &= 5
\end{align*}
\]

### 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.plus(b);
  StdOut.println("a = "+a);
  StdOut.println("b = "+b);
  StdOut.println("c = "+c);
}
```

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

Complex Number Data Type: Implementation

```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);
  }
}
```
Mandelbrot set. A set of complex numbers.
Plot set \((x, y)\) black if \(|z| < 2\) is in the set, and white otherwise.

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

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]);

    Picture pic = new Picture(N, N);

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            double x = xc - size*i/N;
            double y = yc - size*j/N;

            Complex z0 = new Complex(x, y);
            Color color = mand(z0);

            pic.set(i, j, color);
        }
    }

    pic.show();
}
```

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 (abs(z) > 2.0) return StdDraw.WHITE;
        z = z.times(z); z = z.plus(z0);
        if (abs(z) > 4.0) return StdDraw.BLACK;
    }
    return StdDraw.BLACK;
}
```

```java
Color mand(0.5, 1)
```
Mandelbrot Set

\[ \text{Java ColorMandelbrot - R R D C mandel.txt} \]

Mandelbrot Set Music Video

http://www.jonathancoulton.com/songdetails/Mandelbrot Set

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, vx, vy; // instance variables
    public Ball() {
        rx = ry = 0.5;
        vx = 0.015 = Math.random() * 0.03;
        vy = 0.015 = Math.random() * 0.03;
        radius = 0.01 + Math.random() * 0.03;
    }
    public void move() {
        // Ball bounce code...
    }
    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();
```

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

**Object References**

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

```
main memory (64-bit machine)
```

```java
public class Ball {
    private double rx, ry, vx, vy;
    private double radius;
    public Ball() {
        rx = ry = 0.5;
        vx = 0.015 = Math.random() * 0.03;
        vy = 0.015 = Math.random() * 0.03;
        radius = 0.01 + Math.random() * 0.03;
    }
    public void move() {
        // Ball bounce code...
    }
    public void draw() {
        StdDraw.filledCircle(rx, ry, radius);
    }
}
```
Allow client to manipulate an object as a single entity.
- Essentially a machine address (pointer).

Data stored in C7 → CS for abstract bit recycler.

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

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

Moving b2 also moves b1 since they are aliases that reference the same object.

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

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

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

```
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);
        }
    }
}
```

Creating Many Objects

Object References
- 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();
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```java
public class PassByValue {
    static void update(int a, int[] b, String c) {
        a = 7;
        b[1] = 7;
        c = "seven";
        System.out.println(a + c + b[1] + c + c);
    }

    public static void main(String[] args) {
        int a = 3;
        int b[] = {1, 2, 3, 4, 5};
        String c = "three";
        update(a, b, c);
    }
}
```

% java PassByValue
3 7 seven 3 3 seven

A Compound Data Type: Circles

Goal. Data type for circles in the plane.

```java
public class Circle {
    private Point center;
    private double radius;

    public Circle(Point center, double radius) {
        this.center = center;
        this.radius = radius;
    }

    public boolean contains(Point p) {
        return p.distanceTo(center) <= radius;
    }

    public double area() {
        return Math.PI * radius * radius;
    }

    public boolean intersects(Circle c) {
        return center.distanceTo(c.center) <= radius + c.radius;
    }
}
```

% java PointTest
a = (0.716810971264761, 0.0753539063358446)
b = (0.4052136795358151, 0.033848435224524076)
distance = 0.31434944941098036