10. Creating Data Types

Object-oriented programming (OOP)

Object-oriented programming (OOP).
• Create your own data types.
• Use them in your programs (manipulate objects).

Examples (stay tuned for details)

<table>
<thead>
<tr>
<th>data type</th>
<th>set of values</th>
<th>examples of operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color</td>
<td>three 8-bit integers</td>
<td>get red component, brighten</td>
</tr>
<tr>
<td>Picture</td>
<td>2D array of colors</td>
<td>get/set color of pixel (i, j)</td>
</tr>
<tr>
<td>String</td>
<td>sequence of characters</td>
<td>length, substring, compare</td>
</tr>
</tbody>
</table>

An abstract data type is a data type whose representation is hidden from the client.

Impact: We can use ADTs without knowing implementation details.
• Previous lecture: how to write client programs for several useful ADTs
• This lecture: how to implement your own ADTs

Implementing a data type

To create a data type, you need provide code that
• Defines the set of values (instance variables).
• Implements operations on those values (methods).
• Creates and initialize new objects (constructors).

In Java, a data-type implementation is known as a class.

A Java class

| instance variables |
| constructors |
| methods |
| test client |

Instance variables
• Declarations associate variable names with types.
• Set of type values is "set of values".

Methods
• Like static methods.
• Can refer to instance variables.

Constructors
• Methods with the same name as the type.
• No return type declaration.
• Invoked by new, returns object of the type.
Anatomy of a Class

```java
public class Charge {
    private double x, y; // position
    private double q; // charge

    public Charge(double x0, double y0, double q0) {
        x = x0;
        y = y0;
        q = q0;
    }

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

    public String toString() {
        String H = qHVHKHatHKHVHKRKHVHrxHVHKWHKHVHryHVHKSKkHH;
        return H;
    }

    public static void main(String[] args) {
        Charge c = new Charge(.72, .33, .21.3);
        StdOut.printf("%d %e\n", c.potentialAt(.42, .73);
        StdOut.printf("%d %e\n", c.potentialAt(.42, .73);
    }
}
```

ADT for point charges

A **point charge** is an idealized model of a particle that has an electric charge. An ADT allows us to write Java programs that manipulate point charges.

<table>
<thead>
<tr>
<th>Values</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>position (x, y)</td>
<td>(.53, .63)</td>
</tr>
<tr>
<td>electrical charge</td>
<td>21.3</td>
</tr>
<tr>
<td></td>
<td>.81.9</td>
</tr>
</tbody>
</table>

### API (operations)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Charge(double x0, y0, q0)</td>
<td></td>
</tr>
<tr>
<td>doublePotentialAt(double x, y)</td>
<td>electric potential at (x, y) due to charge</td>
</tr>
<tr>
<td>toString()</td>
<td>string representation of this charge</td>
</tr>
</tbody>
</table>

Crash course on electric potential

Electric potential is a measure of the effect of a point charge on its surroundings.

- **It increases** in proportion to the charge value.
- **It decreases** in proportion to the inverse of the distance from the charge.

Mathematically,

- Suppose a point charge \( c \) is located at \((r_x, r_y)\) and has charge \( q \).
- Let \( r \) be the distance between \((x, y)\) and \((r_x, r_y)\).
- Let \( V_c(x, y) \) be the potential at \((x, y)\) due to \( c \).
- Then \( V_c(x, y) = \frac{q \cdot x}{r^2} \) where \( k = 8.99 \times 10^9 \) is a normalizing factor.

Q. What happens when multiple charges are present?

A. The potential at a point is the **sum** of the potentials due to the individual charges.

Note: Similar laws hold in many other situations.
Point charge implementation: Test client

Best practice. Begin by implementing a simple test client.

```java
public static void main(String[] args)
{
    Charge c = new Charge(.72, .31, 20.1);
    StdOut.println(c);
    StdOut.printf("%6.2eln", c.potentialAt(.42, .71));
}
```

\[ V_c(x, y) = \frac{kQ}{r} \]

\[ r = \sqrt{(r_x - x)^2 + (r_y - y)^2} \]
\[ = \sqrt{3^2 + 4^2} = 5 \]
\[ V_c(.42, .71) = 8.99 \times 10^2 \times \frac{20.1}{5} \]
\[ = 3.6 \times 10^1 \]

% java Charge 21.3 at (0.72, 0.31) 3.01e+11

What we expect, once the implementation is done.

Point charge implementation: Constructor

Constructors create and initialize new objects.

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

Clients use `new` to invoke constructors.

- Pass arguments as in a method call.
- Return value is reference to new object.

Possible memory representation of

Charge c = new Charge(.72, .31, 20.1);

\[ x = \text{memory address} \]

Point charge implementation: Instance variables

Instance variables define data-type values.

<table>
<thead>
<tr>
<th>Values</th>
<th>examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>position (x, y)</td>
<td>(-33, 63)</td>
</tr>
<tr>
<td>electrical charge</td>
<td>21.3</td>
</tr>
</tbody>
</table>

Public class Charge

- private final double rx, ry;
- private final double q;

...}

Modifiers control access.

- `private` denies clients access and therefore makes data type abstract.
- `final` disallows any change in value and therefore makes variable immutable.

Key to OOP. Each object has instance-variable values.

Point charge implementation: Methods

Methods define data-type operations (implement APIs).

```java
public class Charge
{
    public double potentialAt(double x, double y)
    { return k * q / Math.sqrt(dx*dx + dy*dy); }
    public String toString()
    { return "Charge at " + x + " , " + y + " with Q = " + q; }
}
```

API

- `Charge(double x0, double y0, double q0)`
- `potentialAt(double x, double y)` electric potential at (x, y) due to charge
- `toString()` string representation of this charge

Public class Charge

```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); }
    public String toString()
    { return "Charge at " + x + " , " + y + " with Q = " + q; }
    ...
}
```

Key to OOP. An instance variable reference in a class method refers to the value for the object that was used to invoke the method.
Point charge implementation

```java
public class Charge
{
    private double rx, ry; // position
    private double q; // charge

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

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

    public String toString()
    {
        return String.format("%s %d at (%.2f, %.3f)", q, potential(rx, ry).
    }
}
```

Point charge client: Potential visualization

```java
import java.awt.Color;
public class Potential
{
    public static Charge[] readCharges()
    {
        int N = StdIn.readInt();
        Charge[] a = new Charge[N];
        for (int i = 0; i < N; i++)
        { double x0 = StdIn.readInt();
            double y0 = StdIn.readInt();
            double q0 = StdIn.readInt();
            a[i] = new Charge(x0, y0, q0);
        }
        return a;
    }
}
```

Potential visualization

```java
import java.awt.Color;
public class Potential
{
    public static Charge[] readCharges()
    {
        int N = StdIn.readInt();
        Charge[] a = new Charge[N];
        for (int i = 0; i < N; i++)
        { double x0 = StdIn.readInt();
            double y0 = StdIn.readInt();
            double q0 = StdIn.readInt();
            a[i] = new Charge(x0, y0, q0);
        }
        return a;
    }
```
Potential visualization II: A moving charge

```
N more charges.txt
0.51 .53 .56 .57 .58 .59 .60 .61 .62 .63 .64 .65 .66 .67 .68 .69 .70 .71 .72 .73 .74 .75 .76 .77 .78 .79 .80 .81 .82 .83 .84 .85 .86 .87 .88 .89 .90 .91 .92 .93 .94 .95 .96 .97 .98 .99 .00 .01 .02 .03 .04 .05 .06 .07 .08 .09 .10 .11 .12 .13 .14 .15 .16 .17 .18 .19 .20 .21 .22 .23 .24 .25 .26 .27 .28 .29 .30 .31 .32 .33 .34 .35 .36 .37 .38 .39 .40 .41 .42 .43 .44 .45 .46 .47 .48 .49 .50 .51 .52 .53 .54 .55 .56 .57 .58 .59 .60 .61 .62 .63 .64 .65 .66 .67 .68 .69 .70 .71 .72 .73 .74 .75 .76 .77 .78 .79 .80 .81 .82 .83 .84 .85 .86 .87 .88 .89 .90 .91 .92 .93 .94 .95 .96 .97 .98 .99

N java PotentialWithMovingCharge < charges0.txt
```

Potential visualization III: Discontinuous color map

```
public static Color toColor(double V) {
    V = 128 + V / 0.0101;
    int t = 0;
    if (V > 255) t = 255;
    else if (V >= 0) t = (int) V;
    return new Color(t, t, t);
}
```

Potential visualization IV: Arbitrary discontinuous color map (a bug!)

If you are an artist
- Choose 255 beautiful colors.
- Put them in an array.
- Index with t to pick a color.

If you are a computer scientist
- Play with colors.
- Maybe you'll hit on something...
Pop quiz 1 on OOP

Q. Fix the serious bug in this code:

```java
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;
    }
}
```

ADT for turtle graphics

A turtle is an idealized model of a plotting device.

An ADT allows us to write Java programs that manipulate turtles.

<table>
<thead>
<tr>
<th>Values</th>
<th>position (x, y)</th>
<th>orientation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(.5, .5)</td>
<td>90°</td>
</tr>
<tr>
<td></td>
<td>(.25, .75)</td>
<td>135°</td>
</tr>
<tr>
<td></td>
<td>(.22, .12)</td>
<td>10°</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>API (operations)</th>
<th>Turtle(double x0, double y0, double q0)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>void turnLeft(double delta)</td>
</tr>
<tr>
<td></td>
<td>rotate delta degrees counterclockwise</td>
</tr>
<tr>
<td></td>
<td>void goForward(double step)</td>
</tr>
<tr>
<td></td>
<td>move distance step, drawing a line</td>
</tr>
</tbody>
</table>

Turtle graphics implementation: Test client

Best practice. Begin by implementing a simple test client.

```java
public static void main(String[] args) {
    Turtle turtle = new Turtle(0.0, 0.0, 0.0);
    turtle.goForward(1.0);
    turtle.turnLeft(120.0);
    turtle.goForward(1.0);
    turtle.turnLeft(120.0);
    turtle.goForward(1.0);
    turtle.turnLeft(120.0);
}
```
Turtle implementation: Instance variables and constructor

**Instance variables** define data-type values.

**Constructors** create and initialize new objects.

```java
public class Turtle {
    private double x, y;  // instance variables are not final
    public Turtle(double x0, double y0, double angle) {
        x = x0;
        y = y0;
        angle = angle;
    }
    ...
}
```

Turtle implementation: Methods

**Methods** define data-type operations (implement APIs).

```java
public class Turtle {
    public void turnLeft(double delta) {
        angle += delta;
    }
    public void forward(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: 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(0.0, 0.0, 0.0);
        for (int i = 0; i < N; i++) {
            turtle.turnLeft(angle);
            turtle.forward(step);
        }
    }
    ...
}
```
Turtle client: Spira Mirabilis

```java
public class Spiral
{
    public static void main(String[] args)
    {
        int N = Integer.parseInt(args[0]);
        double decay = Integer.parseInt(args[1]);
        double angle = 180.0 / 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++)
        {
            step /= decay;
            turtle.goForward(step);
            turtle.turnLeft(angle);
        }
    }
}
```

Spira Mirabilis in the wild

Crash course in complex numbers

A complex number is a number of the form $a + bi$ where $a$ and $b$ are real and $i \equiv \sqrt{-1}$.

Complex numbers are a quintessential mathematical abstraction that have been used for centuries to give insight into real-world problems not easily addressed otherwise.

To perform algebraic operations on complex numbers, use real algebra, replace $i^2$ by $-1$ and collect terms.

- Addition example: $(3 + 4i) + (-2 + 3i) = 1 + 7i$.
- Multiplication example: $(3 + 4i) \times (-2 + 3i) = -18 + i$.

The magnitude or absolute value of a complex number $a + bi$ is $|a + bi| = \sqrt{a^2 + b^2}$.

Applications: Signal processing, control theory, quantum mechanics, analysis of algorithms...
ADT for complex numbers

A complex number is a number of the form $a + bi$ where $a$ and $b$ are real and $i = \sqrt{-1}$.

An ADT allows us to write Java programs that manipulate complex numbers.

<table>
<thead>
<tr>
<th>Values</th>
<th>complex number</th>
<th>3 + 4i</th>
<th>-2 + 2i</th>
</tr>
</thead>
<tbody>
<tr>
<td>real part</td>
<td>3.0</td>
<td>-2.0</td>
<td></td>
</tr>
<tr>
<td>imaginary part</td>
<td>4.0</td>
<td>2.0</td>
<td></td>
</tr>
</tbody>
</table>

Complex number data type implementation: Test client

Best practice. Begin by implementing a simple test client.

```java
public static void main(String[] args) {
    Complex a = new Complex(3.0, 4.0);
    Complex b = new Complex(-2.0, 3.0);
    StdOut.println("a = " + a);
    StdOut.println("b = " + b);
    StdOut.println("a * b = " + a.times(b));
}
```

What we expect, once the implementation is done.

```
% java Complex
a = 3.0 + 4.0i
b = -2.0 + 3.0i
a * b = -10.0 + 18.0i
```

Complex number data type implementation: Methods

Methods define data-type operations (implement APIs).

```java
public class Complex {
    private final double re;
    private final double im;
    public Complex(double real, double imag) {
        re = real;
        im = imag;
    }
    ...
    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 = a.re * b.re - a.im * b.im;
        double imag = a.re * b.im + a.im * b.re;
        return new Complex(real, imag);
    }
    public double abs() {
        return Math.sqrt(re*re + im*im); }
    public String toString() {
        return re + " + " + im + "i";
    }
    ...
}
```

Complex number data type implementation: Instance variables and constructor

Instance variables define data-type values.

Constructors create and initialize new objects.

```java
public class Complex {
    private final double re;
    private final double im;
    public Complex(double real, double imag) {
        re = real;
        im = imag;
    }
    ...
}
```

Values

<table>
<thead>
<tr>
<th>complex number</th>
<th>3 + 4i</th>
<th>-2 + 2i</th>
</tr>
</thead>
<tbody>
<tr>
<td>real part</td>
<td>3.0</td>
<td>-2.0</td>
</tr>
<tr>
<td>imaginary part</td>
<td>4.0</td>
<td>2.0</td>
</tr>
</tbody>
</table>
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 Complex(Complex b) {
        double real = b.re + b.im;
        double imag = b.im - b.re;
        return new Complex(real, imag);
    }

    public Complex times(Complex b) {
        double real = a.re * b.re - a.im * b.im;
        double imag = a.re * b.im + a.im * b.re;
        return new Complex(real, imag);
    }

    public double abs() {
        return -Math.sqrt(re * re + im * im);
    }

    public String toString() {
        return re + " + " + im + "i";
    }

    public static void main(String[] args) {
        Complex a = new Complex(1.0, 0.0);
        Complex b = new Complex(1.0, 5.0);
        StdOut.println(a + " + " + a.times(b));
    }
}
```

The Mandelbrot set

The **Mandelbrot set** is a set of complex numbers.

- Represent each complex number \( x + yi \) by a point \((x, y)\) in the plane.
- If a point is in the set, we color it **BLACK**.
- If a point is not in the set, we color it **WHITE**.

**Examples**

- In the set: \(-0.5 + 0i\)
- Not in the set: \(1 + i\)

**Challenge**

- No simple formula exists for testing whether a number is in the set.
- Instead, the set is defined by an **algorithm**.

Determining whether a point is in the Mandelbrot set

Is a complex number \(z_0\) in the set?

- Iterate \(z_{n+1} = (z_n)^2 + z_0\).
- If \(|z_n| \text{ diverges to infinity, } z_0 \text{ is not in the set.}
- If not, \(z_0\) is in the 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>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(z_{n+1})</th>
<th>(z_n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1 + 0i)</td>
<td>(0 + 1i)</td>
</tr>
<tr>
<td>(1 + 3i)</td>
<td>(1 + i)</td>
</tr>
<tr>
<td>(-7 + 7i)</td>
<td>(2 + 0i)</td>
</tr>
<tr>
<td>(-97i)</td>
<td>(3 + 0i)</td>
</tr>
<tr>
<td>(-1933i)</td>
<td>(4 + 0i)</td>
</tr>
</tbody>
</table>

- \(z = -1/2 \pm 0i\) is in the set
- \(z = 1 + i\) is not in the set
- \(|z| > 2\) for any \(t\), then \(z\) is not in the set.
- \(|z| \leq 2\) then \(z\) is “likely” in the set.

Plotting the Mandelbrot set

**Practical issues**

- Cannot plot infinitely many points.
- Cannot iterate infinitely many times.

**Approximate solution for first issue**

- Sample from an \(N\)-by-\(N\) grid of points in the plane.
- Zoom in to see more detail (stay tuned!).

**Approximate solution for second issue**

- Fact: if \(|z| > 2\) for any \(t\), then \(z\) is not in the set.
- Pseudo-fact: if \(|z| \leq 2\) then \(z\) is “likely” in the set.

**Important note**: Solutions imply significant computation.
Mandelbrot function of a complex number.
- Returns WHITE if the number is not in the set.
- Returns BLACK if the number is (probably) in the set.

```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);
    }
    return Color.BLACK;
}
```

For a more dramatic picture, return new Color(255-t, 255-t, 255-t) or colors picked from a color table.

```java
public class Mandelbrot
{
    public static Color mand(Complex z0)
    {
        // See previous slide...
    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 plc = new Picture(N, N);
        for (int col = 0; col < N; col++)
        {
            for (int row = 0; row < N; row++)
            {
                double x0 = xc - size/2 + size*col/N;
                double y0 = yc - size/2 + size*row/N;
                Complex z0 = new Complex(x0, y0);
                Color color = mand(z0);
                plc.set(col, N-1-row, color);
            }
        }
    plc.show();
}
```
OOP summary

Object-oriented programming (OOP).

- Create your own data types (sets of values and ops on them).
- Use them in your programs (manipulate objects).

OOP helps us simulate the physical world

- Java objects model real-world objects.
- Not always easy to make model reflect reality.
- Examples: charged particle, color, sound, genome....

OOP helps us extend the Java language

- Java doesn't have a data type for every possible application.
- Data types enable us to add our own abstractions.
- Examples: complex, vector, polynomial, matrix, picture....

10. Creating Data Types

Section 3.2
http://introcs.cs.princeton.edu