CLASSES AND OBJECTS

We’re going to Objectville! We’re leaving this dusty ol' procedural town for good. I’ll send you a postcard.
Outline

• Primitive types
• Creating your own data types
  • Classes
  • Objects
  • Instance variables
  • Instance methods
  • Constructors
  • Arrays of objects
A Foundation for Programming

any program you might want to write

objects

methods

graphics, sound, and image I/O

arrays

conditionals and loops

Math

text I/O

primitive data types

assignment statements

build even bigger programs and reuse code

http://www.flickr.com/photos/vermegrigio/5923415248/
<table>
<thead>
<tr>
<th>Java type</th>
<th>what it stores</th>
<th>examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>tiny integer values -128 to 127</td>
<td>3  -87</td>
</tr>
<tr>
<td>short</td>
<td>small integer values -32768 to 32767</td>
<td>-3433 123</td>
</tr>
<tr>
<td>int</td>
<td>integer values -2,147,483,648 to 2,147,483,647</td>
<td>42  1234</td>
</tr>
<tr>
<td>long</td>
<td>big integer values -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807</td>
<td>5454 -43984938</td>
</tr>
<tr>
<td>double</td>
<td>floating-point values</td>
<td>9.95 3.0e8</td>
</tr>
<tr>
<td>float</td>
<td>less precise floating-point values</td>
<td>9.95f 3.0e8f</td>
</tr>
<tr>
<td>boolean</td>
<td>truth values</td>
<td>true false</td>
</tr>
<tr>
<td>char</td>
<td>characters</td>
<td>'a', 'b', '!'</td>
</tr>
</tbody>
</table>
Primitive Types: Limitations

- **Primitive types**
  - Limited set of operations
    - Example: `int` data type operations: add, subtract, multiply, divide, modulo
  - Can't easily combine related information
    - e.g. MarsLander:
      - two `double`'s to represent your Mars lander's position
      - another two for velocity, etc.
    - e.g. LoggingLease:
      - 2D array to track all the values for trees and bears
      - OR 4 1D arrays…
Create Your Own Data Types

- Class
  - *Blueprint* for a custom data type
- Object
  - *Instance* of a class
  - May be multiple objects for a particular class blueprint

- Objects have a set of things they know ("state")
  - Color of different body panels, location, fuel remaining
- Objects have a set of things they can do ("behavior")
  - Honk horn
  - Turn on lights
  - Drive forward
Let's Build a Simple Class

- **Goal:** represent a circle in 2D
  - What does a circle need to know?
    - x-coordinate
    - y-coordinate
    - radius
  - What can a circle do?
    - Draw itself
    - Print out its position and radius
Setting up the Circle Class

- Create `Circle.java` containing Circle class
- Add instance variables for what a `Circle` knows

```java
public class Circle {
    private double posX = 0.0;
    private double posY = 0.0;
    private double radius = 0.0;
}
```

**access modifier:**

`private` = only methods in this class can see and change these instance variables

We almost always declare our instance variables as private.
Adding an Instance Method

- Add instance methods for what a Circle can do

```java
public class Circle {
    private double posX = 0.0;
    private double posY = 0.0;
    private double radius = 0.0;

    public void draw() {
        StdDraw.filledCircle(posX, posY, radius);
    }

    public String toString() {
        return "(" + posX + ", " + posY + ") r = " + radius;
    }
}
```

instance variables: available (in scope) in any instance method of Circle
Adding an Instance Method

- Add **instance methods** for what a **Circle** can do

```java
public class Circle {
    private double posX = 0.0;
    private double posY = 0.0;
    private double radius = 0.0;

    public void draw() {
        StdDraw.filledCircle(posX, posY, radius);
    }

    public String toString() {
        return "(" + posX + ", " + posY + ") r = " + radius;
    }
}
```

**instance methods:** declared **without** the static keyword

**toString()**
Special method, called whenever object printed with `System.out.println`
Let's try out our new class!

- **Instantiating objects**
  - Like arrays, we must declare and create using `new`

```java
public class CircleClient {
    public static void main(String[] args) {
        Circle big = new Circle();
        Circle small = new Circle();

        big.draw();
        small.draw();

        System.out.println("big: " + big);
        System.out.println("small: " + small);
    }
}
```

"Build me a Circle object, I'm not sending you any input about how to do it."
Let's try out our New Class!

• Instantiating objects

• Like arrays, we must declare and create using new

```java
public class CircleClient {
    public static void main(String[] args) {
        Circle big = new Circle();
        Circle small = new Circle();

        big.draw();
        small.draw();

        System.out.println("big: " + big);
        System.out.println("small: " + small);
    }
}
```

% java CircleClient
big: (0.0, 0.0) r = 0.0
small: (0.0, 0.0) r = 0.0
public class Circle {
    private double posX = 0.0;
    private double posY = 0.0;
    private double radius = 0.0;

    public Circle(double x, double y, double r) {
        posX = x;
        posY = y;
        radius = r;
    }

    public void draw() {
        StdDraw.filledCircle(posX, posY, radius);
    }

    public String toString() {
        return "(" + posX + ", " + posY + ") r = " + radius;
    }
}
CircleClient Take Two

- **Constructor called** when we new objects

```java
public class CircleClient {
    public static void main(String [] args) {
        Circle big = new Circle(0.7, 0.7, 0.2);
        Circle small = new Circle(0.1, 0.5, 0.1);

        big.draw();
        small.draw();

        System.out.println("big: " + big);
        System.out.println("small: " + small);
    }
}
```

% java CircleClient
big: (0.7, 0.7) r = 0.2
small: (0.1, 0.5) r = 0.1
Colored Circles

- **Goal:** make each Circle object have a color specified by an red-green-blue (RGB) value
- **Call** `StdDraw.setPenColor()` in `draw()`
  - Create a new `Color` object for a given RGB value
  - `Color` is a class in the Java API
  - Default color for our Circle objects: **mauve**
import java.awt.*;

public class Circle {
    private double posX = 0.0;
    private double posY = 0.0;
    private double radius = 0.0;
    private Color color = new Color(0.88f, 0.68f, 1.0f);

    public Circle(double x, double y, double r) {
        posX = x;
        posY = y;
        radius = r;
    }

    public void draw() {
        StdDraw.setPenColor(color);
        StdDraw.filledCircle(posX, posY, radius);
    }
    ...
}
Allowing Clients to Change Color

```java
import java.awt.*;

public class Circle
{
    private double posX = 0.0;
    private double posY = 0.0;
    private double radius = 0.0;
    private Color color = new Color(0.88f, 0.68f, 1.0f);

    public Circle(double x, double y, double r)
    {
        posX = x;
        posY = y;
        radius = r;
    }

    public void setColor(double r, double g, double b)
    {
        color = new Color((float) r, (float) g, (float) b);
    }

    ...
}
```
public class CircleClient
{
    public static void main(String [] args)
    {
        Circle big = new Circle(0.7, 0.7, 0.2);
        Circle small = new Circle(0.1, 0.5, 0.1);
        big.setColor(Math.random(), Math.random(), Math.random());
        small.setColor(Math.random(), Math.random(), Math.random());
        big.draw();
        small.draw();
        System.out.println("big: "+ big);
        System.out.println("small: "+ small);
    }
}
Creating Lots of Circles

- We can have an array of objects
- **Step 1:** create an array to hold Circle objects

```java
Circle [] circles = new Circle[7];
```
The Value `null`

- What is in each location of the array?
  - Special value `null`
  - Default value for reference types (non-primitives)
  - Like an unprogrammed remote control

```java
Circle[] circles = new Circle[7];
```

```
null null null null null null null
```

```
0 1 2 3 4 5 6 circles
```
Creating all the Circle Objects

- Each array location needs a new object

```java
Circle[] circles = new Circle[7];
for (int i = 0; i < circles.length; i++)
{
    circles[i] = new Circle(Math.random(), Math.random(),
                            Math.random() * 0.2);
    circles[i].setColor(Math.random(), Math.random(), Math.random());
}
```
public class CircleClientDeluxe
{
    public static void main(String[] args)
    {
        Circle[] circles = new Circle[Integer.parseInt(args[0])];
        for (int i = 0; i < circles.length; i++)
        {
            circles[i] = new Circle(Math.random(),
                                        Math.random(),
                                        Math.random() * 0.2);
            circles[i].setColor(Math.random(),
                                 Math.random(),
                                 Math.random());
            circles[i].draw();
        }
    }
}

% java CircleClientDeluxe 100
Overlap Detection

- **Goal:** draw many Circle objects without overlap
  - When do two circles overlap?

Euclidean distance between centers:

\[ d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \]

Circles overlap if:

\[ d < (r_1 + r_2) \]
Implementing Overlap Detection

• Overlap detection is **something a Circle can do**
  • We can add a method to Circle class for this!

**Euclidean distance between centers:**
\[ d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \]

Circles overlap if:
\[ d < (r_1 + r_2) \]

```java
public boolean overlap(Circle other) {
    double deltaX = posX - other.posX;
    double deltaY = posY - other.posY;
    double d = Math.sqrt(deltaX * deltaX + deltaY * deltaY);
    if (d < (radius + other.radius))
        return true;
    return false;
}
```
public class CircleClientSuperDeluxe
{
    public static void main(String[] args)
    {
        Circle[] circles = new Circle[Integer.parseInt(args[0])];
        for (int i = 0; i < circles.length; i++)
        {
            boolean overlap = false;
            do
            {
                circles[i] = new Circle(Math.random(),
                                         Math.random(),
                                         Math.random() * 0.2);

                int j = 0;
                overlap = false;
                while ((j < i) && (!overlap))
                {
                    overlap = circles[i].overlap(circles[j]);
                    j++;
                }
            } while (overlap);
            circles[i].setColor(Math.random(),
                                 Math.random(),
                                 Math.random());
            circles[i].draw();
        }
    }
}
Recap

• **Creating your own data types**
  - Object-oriented programming (OOP)
  - Design classes encapsulating:
    • What objects know (“state”)
    • What objects can do (“behavior”)
  - Prevalent concept in most modern programming languages
Summary

- Primitive types
- Creating your own data types
  - Classes
  - Objects
  - Instance variables
  - Instance methods
  - Constructors
  - Arrays of objects