Outline

• **Inheritance**
  - Sharing code between related classes
  - Putting similar objects in the same bucket
  - Extremely common in modern OOP languages

• **Managing many objects**
  - Create class holding a collection of other objects
  - Let's you simplify your main program
  - Hides details of how you store things
Inheritance

• One class *extends* another
  • Parent class: shared vars/methods
  • Child class: more specific vars/methods
    • Class declared *extends* the parent class

• Why? Lets you share code
  • Repeated code is evil

• Why? Store similar objects in same bucket
  • Can lead to simpler implementations
Inheritance Example

- **Goal:** Animate circles that bounce off the walls
  - What does an object know?
    - x-position, y-position
    - x-velocity, y-velocity
    - radius
  - What can an object do?
    - Draw itself
    - Update its position, check for bouncing off walls
public class Circle {
    private double x, y, vx, vy, r;

    public Circle(double x, double y, double vx, double vy, double r) {
        this.x = x;
        this.y = y;
        this.vx = vx;
        this.vy = vy;
        this.r = r;
    }

    public void draw() {
        StdDraw.setPenColor(StdDraw.RED);
        StdDraw.circle(x, y, r);
    }

    public void updatePos() {
        x += vx;
        y += vy;
        if ((x < 0.0) || (x > 1.0))
            vx *= -1;
        if ((y < 0.0) || (y > 1.0))
            vy *= -1;
    }

    public double getX() {
        return x;
    }

    public double getY() {
        return y;
    }

    public double getRadius() {
        return r;
    }
}
public class CircleClient
{
    public static void main(String[] args)
    {
        Circle[] circles = new Circle[30];

        for (int i = 0; i < circles.length; i++)
            circles[i] = new Circle(Math.random(),
                                      Math.random(),
                                      0.002 - Math.random() * 0.004,
                                      0.002 - Math.random() * 0.004,
                                      Math.random() * 0.1);

        while (true)
        {
            StdDraw.clear();
            for (int i = 0; i < circles.length; i++)
            {
                circles[i].updatePos();
                circles[i].draw();
            }
            StdDraw.show(10);
        }
    }
}
Inheritance Example

• **Goal:** Add images that bounce around
  • What does an object know?
    • x-position, y-position
    • x-velocity, y-velocity
    • radius
    • image filename
  • What can an object do?
    • Draw itself
    • Update its position, check for bouncing off walls
public class CircleImage {
    private double x, y, vx, vy, r;
    private String image;

    public CircleImage(double x, double y, double vx, double vy, double r, String image) {
        this.x = x;
        this.y = y;
        this.vx = vx;
        this.vy = vy;
        this.r = r;
        this.image = image;
    }

    public void draw() {
        StdDraw.picture(x, y, image, r * 2, r * 2);
    }

    public void updatePos() {
        x += vx;
        y += vy;
        if ((x < 0.0) || (x > 1.0))
            vx *= -1;
        if ((y < 0.0) || (y > 1.0))
            vy *= -1;
    }
}
Inheritance: Bouncing Circular Images!

This class is a child of the Circle class

We only need our additional instance variable, others inherited from Circle

```java
public class CircleImage extends Circle {
    private String image; // image representing object

    public CircleImage(double x, double y, double vx, double vy, double r, String image)
    {
        super(x, y, vx, vy, r);
        this.image = image;
    }

    public void draw()
    {
        StdDraw.picture(getX(), getY(), image, getRadius() * 2, getRadius() * 2);
    }
}
```

Override = method with same method signature as parent's method

Overridden version of draw() method, this one draws a picture scaled according to the radius.

NOTE: Need getter methods to get at private instance variables declared in parent.

Overload = multiple methods in same class with different signatures
Inheritance Example

• **Goal:** Add images that bounce and rotate
  • What does an object know?
    • x-position, y-position
    • x-velocity, y-velocity
    • radius
    • image filename
    • rotation angle
  • What can an object do?
    • Draw itself
    • Update its position, check for bouncing off walls
      rotate image by one degree
public class CircleImageRotate extends CircleImage {
    private int angle; // current rotation angle of image

    public CircleImageRotate(double x, double y, double vx, double vy, double r, String image) {
        super(x, y, vx, vy, r, image);
    }

    public void draw() {
        StdDraw.picture(getX(), getY(), getImage(),
                        getRadius() * 2, getRadius() * 2, angle);
    }

    public void updatePos() {
        angle = (angle + 1) % 360;
        super.updatePos();
    }
}
Unified Modeling Language (UML) Class Diagram

Circle
- x: double
- y: double
- vx: double
- vy: double
- r: double

Circle(double, double, double, double, double, double)
- draw(): void
- updatePos(): void
- getX(): double
- getY(): double
- getRadius(): double

CircleImage
- image: String

CircleImage(double, double, double, double, double, double, String)
- draw(): void
- getImage(): String

CircleImageRotate
- angle: int

CircleImageRotate(double, double, double, double, double, double, String)
- draw(): void
- updatePos(): void
Client, 3 Object Types, Without Inheritance/Polymorphism

- **Goal:** Bouncing circles, images, and rotating images

  - Create three different arrays (tedious!)

    ```java
    Circle[] circles1 = new Circle[10];
    CircleImage[] circles2 = new CircleImage[10];
    CircleImageRotate[] circles3 = new CircleImageRotate[10];
    ```

  - Fill in all three arrays (tedious!)

    ```java
    for (int i = 0; i < circles1.length; i++)
        circles1[i] = new Circle(x, y, vx, vy, r);
    for (int i = 0; i < circles2.length; i++)
        circles2[i] = new CircleImage(x, y, vx, vy, r, "dont_panic_40.png");
    for (int i = 0; i < circles3.length; i++)
        circles3[i] = new CircleImageRotate(x, y, vx, vy, r, "asteroid_big.png");
    ```

  - Loop through them separately (tedious!)

    ```java
    for (int i = 0; i < circles1.length; i++)
        circles1[i].updatePos();
    for (int i = 0; i < circles2.length; i++)
        circles2[i].updatePos();
    for (int i = 0; i < circles3.length; i++)
        circles3[i].updatePos();
    ```
Circle [] circles = new Circle[30];
for (int i = 0; i < circles.length; i++)
{
    int rand = (int) (Math.random() * 3.0);
    double x = Math.random();
    double y = Math.random();
    double vx = 0.002 - Math.random() * 0.004;
    double vy = 0.002 - Math.random() * 0.004;
    double r = Math.random() * 0.1;

    if (rand == 0)
        circles[i] = new Circle(x, y, vx, vy, r);
    else if (rand == 1)
        circles[i] = new CircleImage(x, y, vx, vy, r, "dont_panic_40.png");
    else
        circles[i] = new CircleImageRotate(x, y, vx, vy, r, "asteroid_big.png");
}

while (true)
{
    StdDraw.clear();
    for (int i = 0; i < circles.length; i++)
    {
        circles[i].updatePos();
        circles[i].draw();
    }
    StdDraw.show(10);
}
What Method gets Run?

```java
while (true)
{
    StdDraw.clear();
    for (int i = 0; i < circles.length; i++)
    {
        circles[i].updatePos();
        circles[i].draw();
    }
    StdDraw.show(10);
}
```

circles[i] could be: Circle, CircleImage or CircleImageRotate object

### Circle
- `x, y, vx, vy, r`
- `draw()`
- `updatePos()`

### CircleImage
- `image`
- `draw()`

### CircleImageRotate
- `angle`
- `draw()`
- `updatePos()`

**Rule:** Most specific method executes. If the subclass has the desired method, use that. Otherwise try your parent. If not, then your parent's parent, etc.
Access Modifiers

- **Access modifiers**
  - Controls if subclasses see instance vars/methods
    - *private* = only the class itself
    - *public* = everybody can see
    - no modifier (default) = everybody in package
    - *protected* = everybody in package, any class that extends it (even outside of package)
Object Collections

- **Goal:** Simplify main, offload work to object that manages a collection of objects
  - Helps hide implementation details
    - You can change how you store things later
- **Let's fix up the bouncing main()**
  - Introduce new class **Bouncers**
  - Holds all the Circle type objects
  - Update and draw them all at once
Simplified main Program

```java
Bouncers bouncers = new Bouncers();

for (int i = 0; i < 30; i++)
    bouncers.add();

while (true)
{
    StdDraw.clear();

    bouncers.updateAll();
    bouncers.drawAll();

    StdDraw.show(10);
}
```

```java
public class Bouncers
{
    // Create an empty collection of bouncing objects
    Bouncers() {

    // Add a random type of bouncing object with a
    // random location, velocity, and radius
    void add() {

    // Update the position of all bouncing objects
    void updateAll() {

    // Draw all the objects to the screen
    void drawAll() {

    Application Programming Interface (API) for the Bouncers class.
```
Bouncer Implementation, 1/2

```java
public class Bouncers {
    private ArrayList<Circle> objs = new ArrayList<Circle>();

    public void add() {
        int rand = (int) (Math.random() * 3.0);
        double x = Math.random();
        double y = Math.random();
        double vx = 0.002 - Math.random() * 0.004;
        double vy = 0.002 - Math.random() * 0.004;
        double r = Math.random() * 0.1;

        if (rand == 0)
            objs.add(new Circle(x, y, vx, vy, r));
        else if (rand == 1)
            objs.add(new CircleImage(x, y, vx, vy, r, "dont_panic_40.png"));
        else
            objs.add(new CircleImageRotate(x, y, vx, vy, r, "asteroid_big.png"));
    }
}
```

I decided to use an ArrayList as my underlying data structure.

Note: clients of Bouncers don't know this and don't really have to care.
Bouncer Implementation, 2/2

public void updateAll()
{
    for (Circle obj : objs)
    {
        obj.updatePos();
    }
}

public void drawAll()
{
    for (Circle obj : objs)
    {
        obj.draw();
    }
}

Perfect time to bust out the enhanced for loop. Much more succinct than looping over all the integer indexes.
Summary

• **Object inheritance**
  • Share code between similar objects
  • Can put objects related by inheritance into a single collection (array, ArrayList, etc.)

• **Class holding collection of objects**
  • Helps simplify and contain logic