Inheritance

• Extend class that is similar
• Define what is different

Many things fall

• Balls (as in BoxBall)
• Leaves
• Sleet
• Snow

Falling Objects

• Similarities
  — Active
  — Start at the top of the canvas
  — Fall at some speed
  — Disappear at a certain point
• Differences
  — Appearance
  — Speed

Define a class that captures the common features

```java
public class FallingObject {
    private double ySpeed;
    private double fallToPos;
    private Drawable2DInterface object;

    private static final int DELAY_TIME = 33;
}
```
Define a class that captures the common features

```java
public void run() {
    while (object.getY() < fallToPos) {
        pause(DELAY_TIME);
        object.move(0, ySpeed);
    }
    object.removeFromCanvas();
}
```

Define a class that captures the common features

```java
// constructor? What are the shared elements?
public void run() {
    while (object.getY() < fallToPos) {
        pause(DELAY_TIME);
        object.move(0, ySpeed);
    }
    object.removeFromCanvas();
}
```

Constructor

```java
public FallingObject(int thePos, double theSpeed) {
    ySpeed = theSpeed;
    fallToPos = thePos;
}
```

FallingLeaf?

```java
public class FallingLeaf extends FallingObject {
    // construct leaf and put at top of screen at x from left
    public FallingLeaf(DrawingCanvas canvas, Image leafpic, double x, double speed, int screenHeight) {
        super(screenHeight, speed);
        object = new VisibleImage(leafpic, 0, 0, canvas);
        object.move(x, -object.getHeight());
        this.start();
    }
}
```

super?

- superclass call
- If no superclass call in constructor, Java assumes “super()” and inserts/execute it for you.
  - This is why the lab worked!
Other FallingObjects

• Sleet (see constructor)
• FallingTomatoes?
  – They go “splat”

FallingTomato: run( )

```java
public void run() {
  while (object.getY() < fallToPos) {
    object.move(0, ySpeed);
  }
  // But then a little extra
  FilledOval splat = new FilledOval(object.getKX() - SIZE / 2, fallToPos, SIZE * 2, SIZE / 2, canvas);
  splat.setColor(Color.red);
  pause(SPLATTER_DELAY);
  splat.removeFromCanvas();
}
```

FallingTomato: run( )

```java
public void run() {
  while (object.getY() < fallToPos) {
    object.move(0, ySpeed);
  }
  // But then a little extra
  FilledOval splat = new FilledOval(object.getKX() - SIZE / 2, fallToPos, SIZE * 2, SIZE / 2, canvas);
  splat.setColor(Color.red);
  pause(SPLATTER_DELAY);
  splat.removeFromCanvas();
}
```

Other FallingObjects?

• Sleet (see constructor)
• FallingTomatoes?
  – They go “splat”
• FallingCows?
  – They go “moo”

Options?

• Option A: override run
  – Pros: easy; works
  – Cons: rewriting code (harder to change later); probably other cases where things happen at bottom

• Option B: change FallingObject to make it easier to change behavior of objects when they hit the bottom

Option B

```java
public void run() {
  while (object.getY() < fallToPos) {
    object.move(0, ySpeed);
  }
  this.hitBottom();
}
```

// Default “hit bottom” behavior - may be overridden
protected void hitBottom() {
  object.removeFromCanvas();
}

```java
public void run() {
  while (object.getY() < fallToPos) {
    object.move(0, ySpeed);
  }
  this.hitBottom();
}
```

// Default “hit bottom” behavior - may be overridden
protected void hitBottom() {
  object.removeFromCanvas();
}
Option B examples

- FallingTomato
  - `start()` invokes `run()` in the superclass
  - `hitBottom` in `run()` invokes the version in `FallingTomato`
  - Always invoke method most specific to the object
- FallingCow

Option C: super

- Use super call to invoke shared code in superclass
- Then execute own (local) code

Access Levels

- `private`: visible only in class
- `protected`: visible in class and subclasses
- `public`: visible everywhere

Basic Rules

- Instance variables:
  - `private` if no plan for future extensions
  - `protected` if class might be extended
- Methods:
  - `public` if to be used by other classes
  - `private` if only used inside class (helper methods)
  - `protected` if used in subclass

extends vs implements

- Interface: specify set of methods
- Inheritance:
  - `Reuse code`
  - Can treat subclass as superclass
    - `FallingObject o = new FallingSleet;`
    - `Images lab!`
    - `Subtype polymorphism: methods written to operate on the supertype also`
- One “extends” allowed.
- Multiple “implements” allowed.