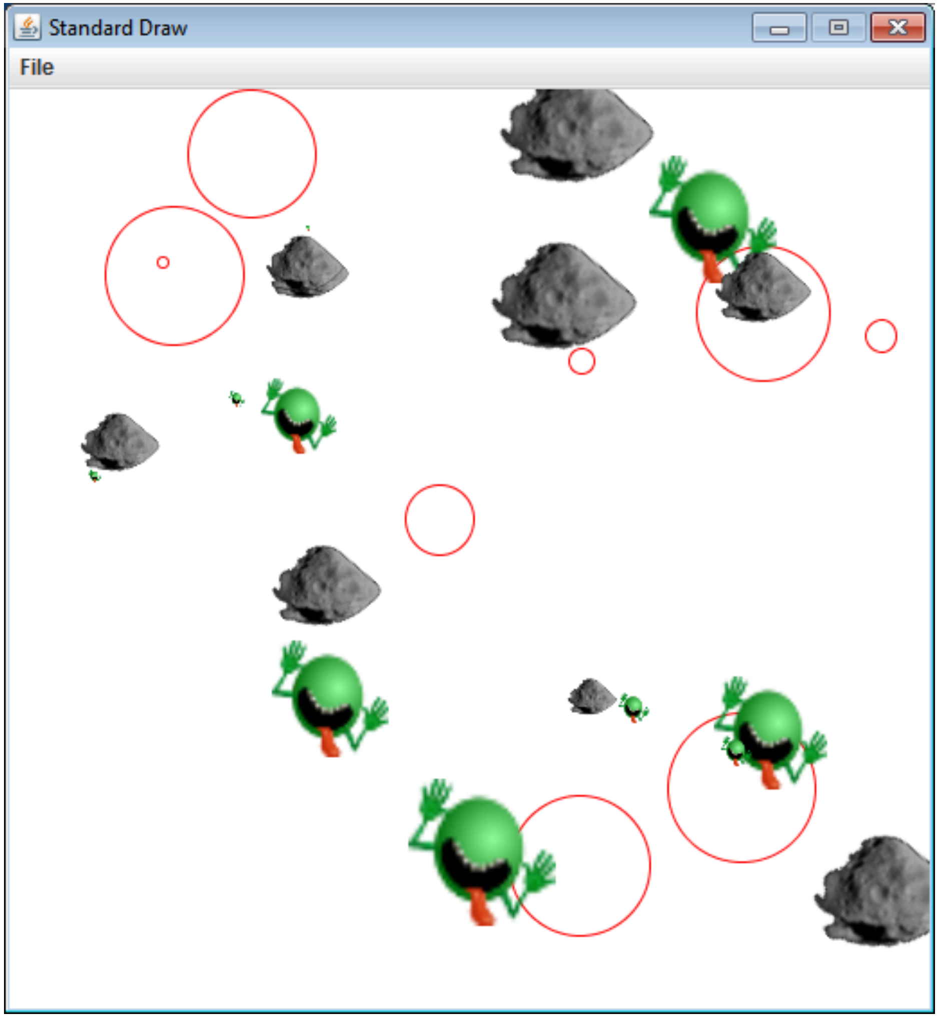
Inheritance and objects, Regular expressions



Overview

- Inheritance
  - Sharing code between related classes
  - Putting similar objects in the same bucket
- Common design pattern
  - Class that holds a collection of other objects
  - Let's you simplify your main program
  - Hides details of how you store things
- Regular expressions
  - Matching strings again patterns

Inheritance

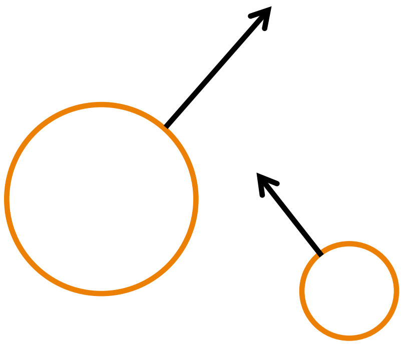
- One class can "extend" another
  - Parent class: shared vars/methods
  - Child class: more specific vars/methods

- Lets you share code
  - Repeated code is evil

- 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

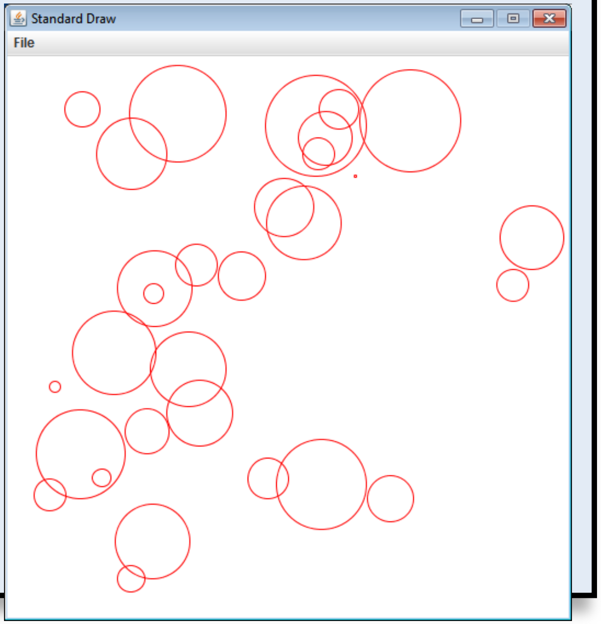


Bouncing circle class

```java
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;
   }
}
```

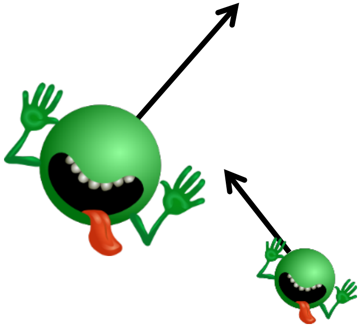
Bouncing circle client

```
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



Bouncing circular image class

```
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()
   { ... }
}
```

All this code appeared in the Circle class!

Bouncing circular image class

```
public class CircleImage extends Circle
{
    protected String image;

    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(x, y, image, r * 2, r * 2);
    }
}
```

This class is a child of the `Circle` class

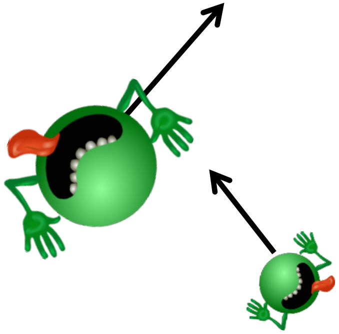
Calls the `Circle` constructor which sets all the other instance variables.

We need `protected` access modifier so children can see our instance variables.

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

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



Rotating bouncing circular image class

```java
public class CircleImageRotate extends CircleImage
{
   protected int angle;

   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(x, y, image, r * 2, r * 2, angle);
   }

   public void updatePos()
   {
      angle = (angle + 1) % 360;
      super.updatePos();
   }
}
```

Calls the constructor of our parent class `CircleImage`.

Calls the `updatePos()` in our parent's parent class `Circle`.

Client with three object types

- Goal: Random collection of bouncing circles, images and rotating images
- Without inheritance:
  - Create three different arrays:

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

  - Loop through them separately:

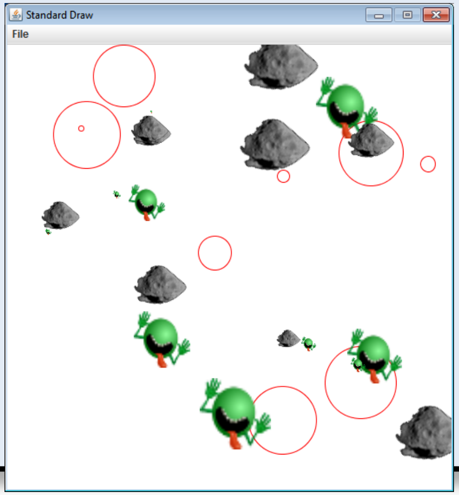
```
for (int i = 0; i < circles1.length; i++)
   circles1[i].updatePos();

for (int i = 0; i < circles2.length; i++)
   circles2[i].updatePos();

for (int i = 0; i < circles3.length; i++)
   circles3[i].updatePos();
```

Client with three object types

```
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);
}
```

With inheritance:

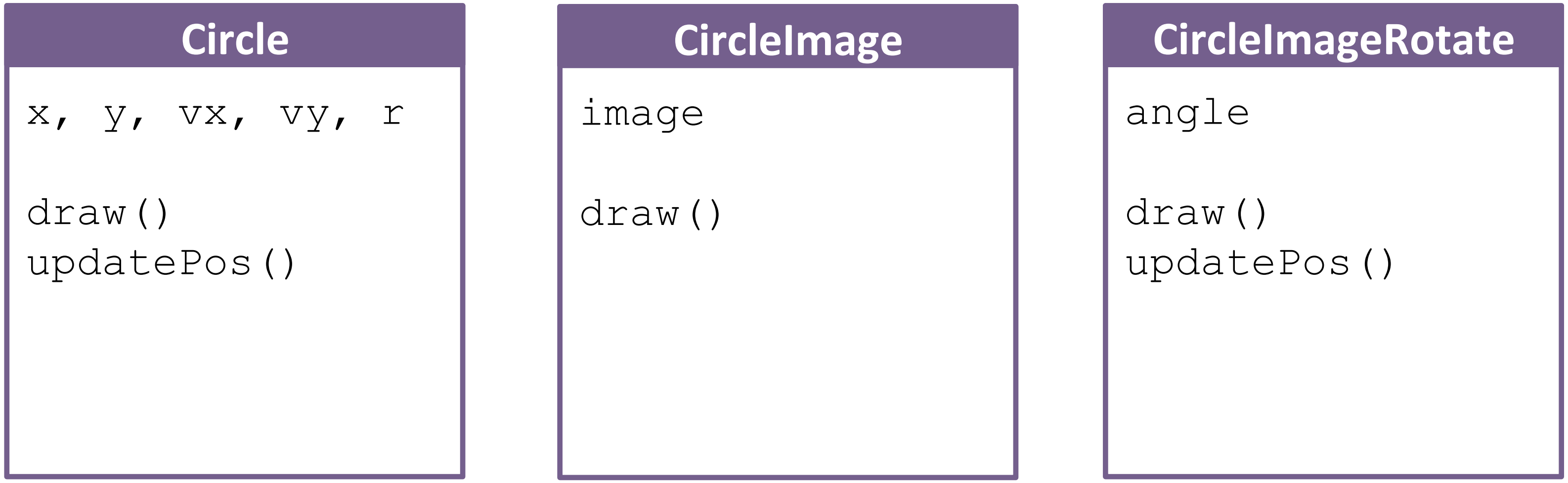
Put them all together in one array!



What method gets run?

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

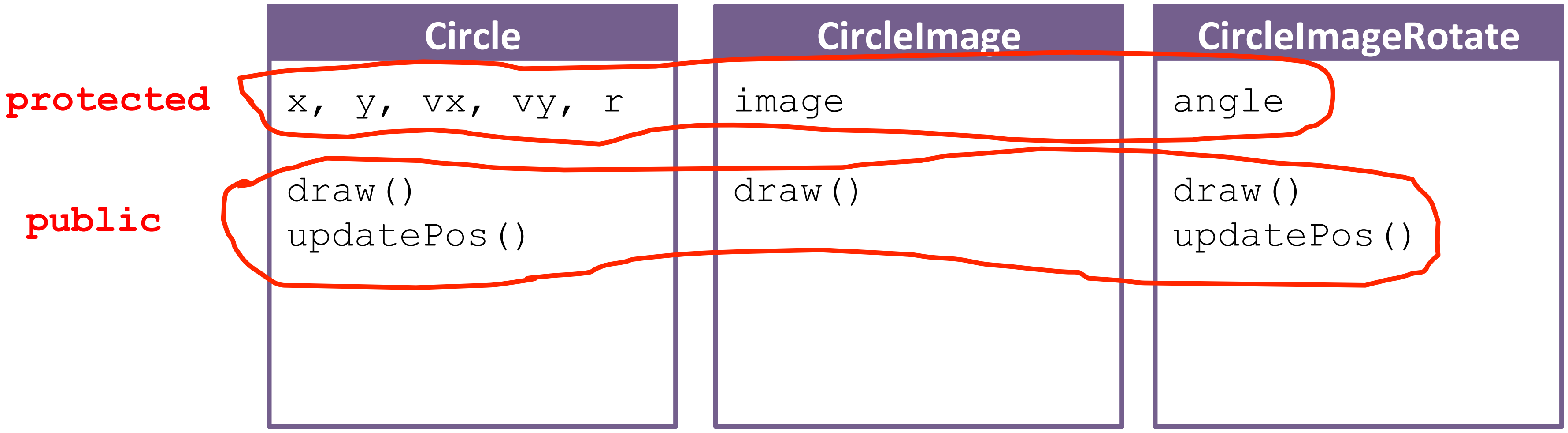
`circles[i]` may be a `Circle`, `CircleImage` or `CircleImageRotate` object



Most specific method will run. 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
    - protected = class itself and any class that extends it
    - public = everybody can see it

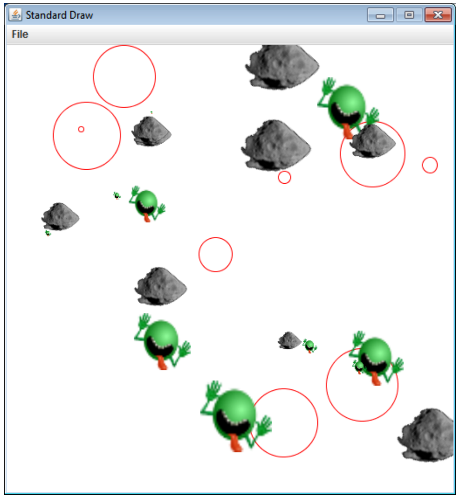


	Circle	CircleImage	CircleImageRotate
protected	`x, y, vx, vy, r`	`image`	`angle`

public	`draw()`
	`updatePos()`
		`draw()`	`draw()`
			`updatePos()`

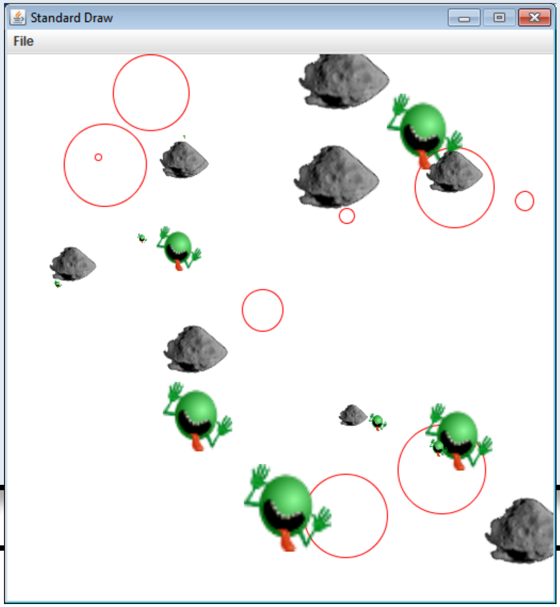
Object storing a collection

- Goal: Simplify main program, offload work into object that manages a bunch of objects
  - Also 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

```
Bouncers bouncers = new Bouncers();

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

while (true)
{
   StdDraw.clear();
   bouncers.updateAll();
   bouncers.drawAll();
   StdDraw.show(10);
}
```



```
public class Bouncers
-----------------------------------------------------------------------
public void add()          // add a random type of bouncing object with a
                           // random location, velocity, and radius
public void updateAll()    // update the position of all bouncing objects
public void drawAll()      // draw all the objects to the screen
```

Application Programming Interface (API) for the Bouncers class.

Bouncer implementation, 1/2

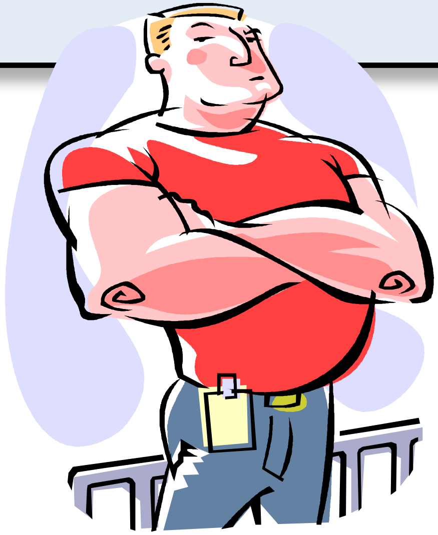
```
public class Bouncers
{
   private ArrayList<Circle> objs = new ArrayList<Circle>();

   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"))
   }
```

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 of `objs`



Parse column data

- Goal: Compute average of a line of numbers
- Problem: Numbers per line is unknown

```
10 20 30
40.0

50 60.12
70 80 90 100 110 120 130 140
1.2 2.3 3.4
```

avgnums.txt

```
% java AvgPerLine < avgnums.txt
20.0
40.0
55.06
105.0
2.3000000000000003
```

AvgPerLine implementation

```
public class AvgPerLine
{
   public static void main(String [] args)
   {
      while (!StdIn.isEmpty())
      {
          String line = StdIn.readLine();
          String [] cols = line.split("\\s+");
          if ((cols.length > 0) && (cols[0].length() > 0))
          {
              double total = 0.0;
              for (String col : cols)
                 total += Double.parseDouble(col);
              System.out.println(total / cols.length);
         }
     }
   }
}
```

Read in entire line of text

Split on whitespace

Regular expressions

- Helps match and split up strings
  - Built-in to Java's String class methods:
    - `String [] split(String regex)`
    - `boolean matches(String regex)`
    - `String replaceAll(String regex, String replacement)`
  - Escape any \ in regular expression as \\

```
String [] cols = line.split("\\s+");
```

Regular expression that matches 1 or more whitespace characters

Regular expression quick reference

Construct	Matches
.	Any character
\d	A digit: 0-9
\s	A whitespace character
\w	A word character: a-z A-Z 0-9 _
\D	A non-digit (anything except 0-9)
\S	A non-whitespace character
\W	A non-word character

Expression	Example matches
…	cat, sat, mat, …
c..	cat, cow, cut, …
[abc]at	aat, bat, cat
[abc]+z	az, bz, cz, aaz, abz, bcz, bbacz, …
[0-9]{5}	12345, 59701, 01234, …
\d\d\d\d	1980, 2005, 9999, …

Classes	Matches
[abc]	Character a, b or c
[^abc]	Any character except a, b, or c
[a-z]	Characters a, b, c, …, z
[A-Z]	Characters A, B, C, …, Z
[a-zA-Z]	Characters a, A, b, B, …, z, Z

Quantifier	Matches
*	Zero or more occurrences
+	One or more occurrences
?	Zero or one occurrences
{n}	Exactly n occurrences
{n,}	At least n occurrences
{n,m}	Between n and m occurrences inclusive

Regular expression example

- Goal: Display all words in a file ending -ing

```
% java GerundFinder < mobydick.txt

having nothing driving regulating growing pausing bringing stepping knocking not
hing surprising leaning looking striving pacing Nothing loitering falling enchan
ting reaching overlapping receiving meaning going something something taking goi
ng being broiling thing putting lording making anything knowing paying paying be
ing paying being considering having whaling going whaling something "Whaling wha
ling being performing cajoling resulting discriminating overwhelming attending e
verlasting ignoring whaling Quitting learning reaching following whaling somethi
ng everything monopolizing having following shouldering comparing halting pausin
g tinkling stopping moving proceeding thing flying hearing sitting beating weepi
ng wailing teeth-gnashing backing Moving creaking looking swinging painting repr
esenting swinging leaning howling toasting chattering shaking everlasting making
 holding being blubbering going Entering straggling reminding painting understan
ding throwing something hovering floating painting something weltering purposing
 spring impaling glittering resembling sweeping death-harvesting horrifying whal
ing sojourning Crossing howling Projecting dark-looking goggling cheating enteri
ng examining telling tapping sharing ruminating adorning stooping working trying
 adjoining Nothing winding scalding looking nothing knowing evening rioting Star
ting offing tramping capering making sleeping making dazzling seeming sleeping s
leeping being getting going feeling saying dusting planing grinning spraining pl
aning gathering throwing yoking leaving standing looking seeing spending cherish
```

GerundFinder

```java
public class GerundFinder
{
   public static void main(String [] args)
   {
      while (!StdIn.isEmpty())
      {
          String word = StdIn.readString();
          if (word.matches(".+ing"))
             System.out.print(word + " ");
      }
      System.out.println();
   }
}
```

1 or more characters followed by "ing"

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
- Regular expressions
  - Built in string pattern matching in Java
  - Helped us split line of text into array of strings