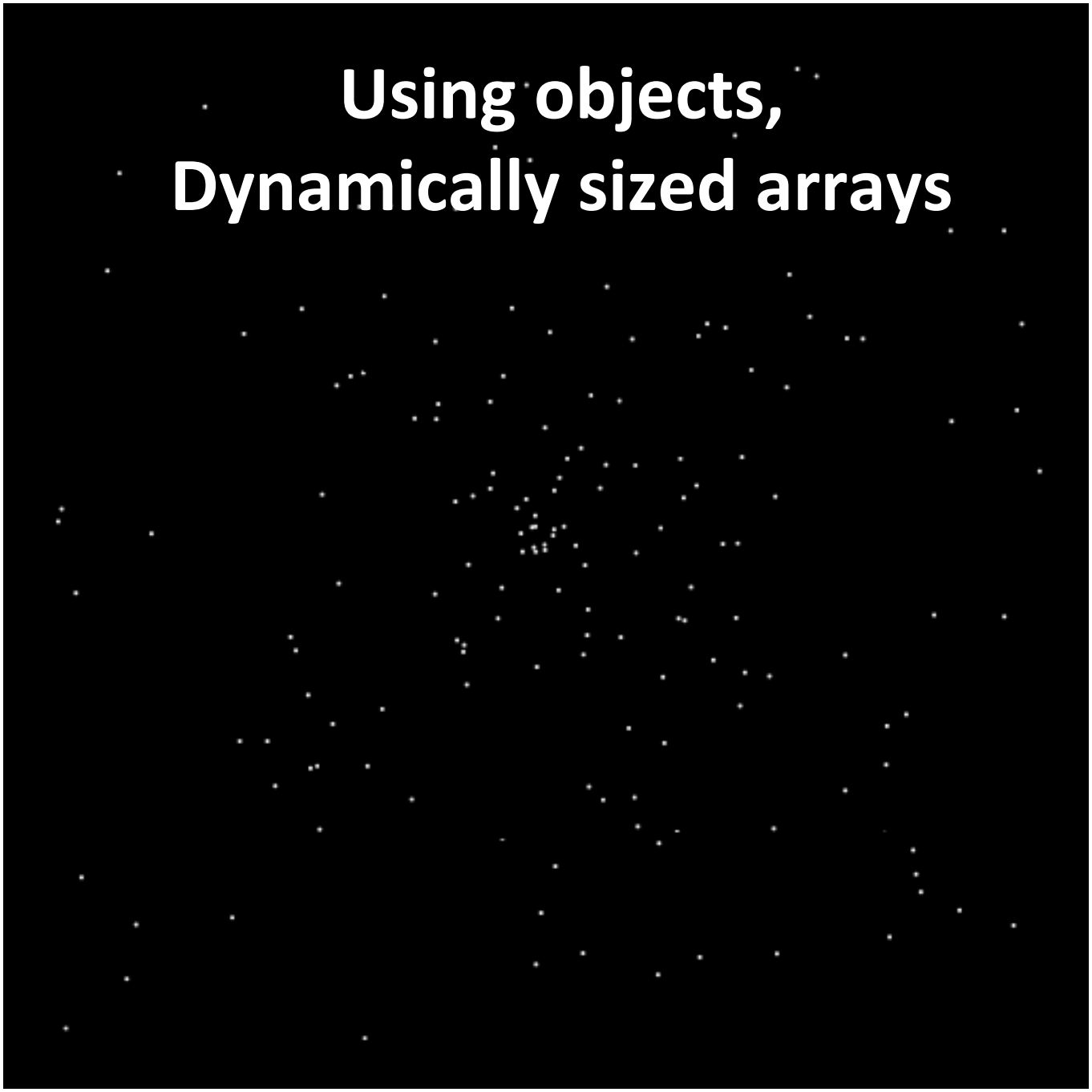
# Using objects, Dynamically sized arrays

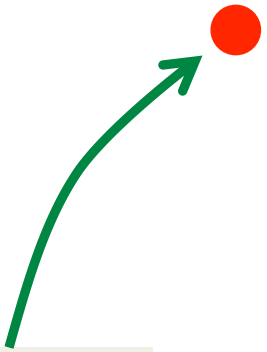


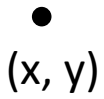
# Overview

- Using an object example
  - Charge & Picture object
  - Client to visualize charge potential
  - Example of a normal fixed sized array
- Dynamically sized arrays
  - Java ArrayList
  - Java packages and the import statement
  - Wrapper classes for primitive types

# Using objects

- Assume we are given an object data type that represents a charged particle.






**A charged particle.**

**What does the object know?**

Three floating-point numbers:
- x-position
- y-position
- electrical charge

**What can the object do?**

Calculate the electrical potential at a point (x, y) given the particle's x-position, y-position, and charge.

Print itself out to the console.

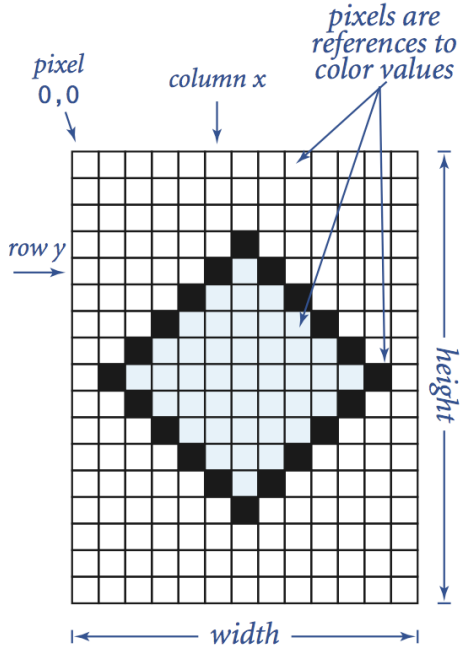
# Charged particle API

- API (Application Programming Interface)
  - Public specification for what a class does
  - All a client program needs to know
  - Signature, return type, and comments for all public methods
- API for charged particle class:

```
public class Charge
-----------------------------------------
       Charge(double x0, double y0, double q0) // location and charge
double potentialAt(double x, double y)         // potential @ (x,y) due to charge
String toString()                              // string representation
```

# API for object representing an image

```
public class Picture
---------------------------------------------------------------------
       Picture(String filename)    // create a picture from a file
       Picture(int w, int h)       // create a blank w-by-h picture
   int width()                     // return the width of the picture
   int height()                    // return the height of the picture
 Color get(int i, int j)           // return the color of pixel (i,j)
  void set(int i, int j, Color c)  // set the color of pixel (i,j) to c
  void show()                      // display the image in a window
  void save(String filename)       // save the image to a file
```



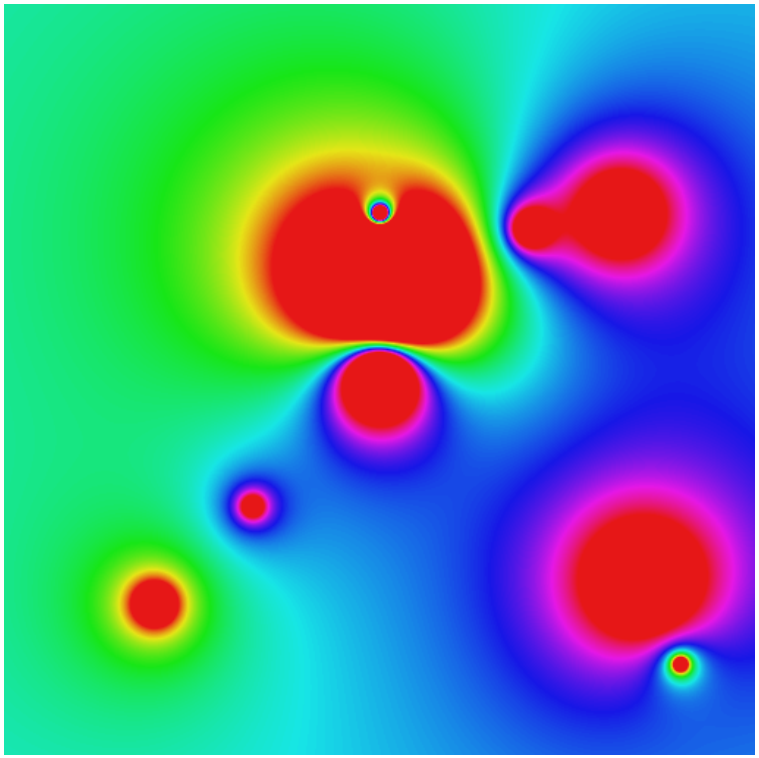
# Using Charge and Picture

- Goal: read in point charges from a file, compute total potential in unit square

```
9
.51  .63 -100
.50  .50   40
.50  .72   10
.33  .33    5
.20  .20  -10
.70  .70   10
.82  .72   20
.85  .23   30
.90  .12   -5
```

charges.txt

```
% java Potential < charges.txt
```



# Charge + Picture client, part 1

```
public static void main(String[] args)
{
    // Read in the particles from the file
    int n =
    Charge [] a =
    for (int i = 0; i < n; i++)
    {
        double x =
        double y =
        double q =
        a[i] =
    }
```

```
9
.51 .63 -100
.50 .50   40
.50 .72   10
.33 .33    5
.20 .20  -10
.70 .70   10
.82 .72   20
.85 .23   30
.90 .12  -5
```

```
public class Charge
-----------------------------------------
       Charge(double x0, double y0, double q0) // location and charge
double potentialAt(double x, double y)         // potential @ (x,y) due to charge
String toString()                              // string representation
```

# Charge + Picture client, part 1

```
public static void main(String[] args)
{
    // Read in the particles from the file
    int n = StdIn.readInt();
    Charge [] a = new Charge[n];
    for (int i = 0; i < n; i++)
    {
        double x = StdIn.readDouble();
        double y = StdIn.readDouble();
        double q = StdIn.readDouble();
        a[i] = new Charge(x, y, q);
    }
```

```
9
.51 .63 -100
.50 .50   40
.50 .72   10
.33 .33    5
.20 .20  -10
.70 .70   10
.82 .72   20
.85 .23   30
.90 .12  -5
```

```
public class Charge
-----------------------------------------
       Charge(double x0, double y0, double q0) // location and charge
double potentialAt(double x, double y)         // potential @ (x,y) due to charge
String toString()                              // string representation
```

# Charge + Picture client, part 2

```
    // Prepare an empty picture to store the visualization of the potential
    final int SIZE = 512;
    Picture pic = new Picture(          );

    // Loop over all rows in the image
    for (int row                        )
    {
        // Loop over all columns in the image
        for (int col                        )
        {
```

```
public class Picture
-----------------------------------------------------------------
       Picture(String filename)   // create a picture from a file
       Picture(int w, int h)      // create a blank w-by-h picture
   int width()                    // return the width of the picture
   int height()                   // return the height of the picture
 Color get(int i, int j)          // return the color of pixel (i,j)
  void set(int i, int j, Color c) // set the color of pixel (i,j) to c
  void show()                     // display the image in a window
  void save(String filename)      // save the image to a file
```

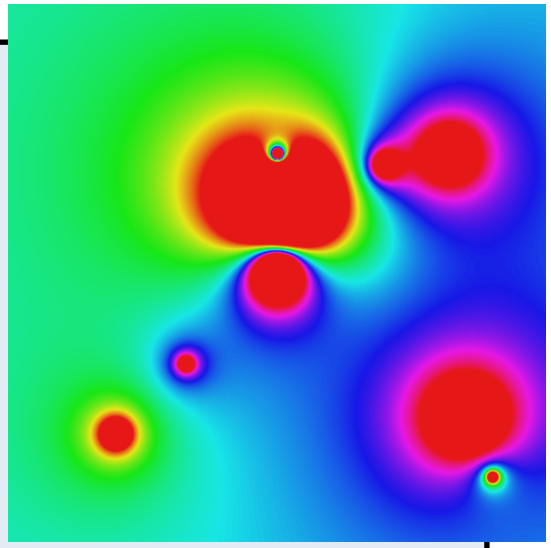
# Charge + Picture client, part 2

```
    // Prepare an empty picture to store the visualization of the potential
    final int SIZE = 512;
    Picture pic = new Picture(SIZE, SIZE);

    // Loop over all rows in the image
    for (int row = 0; row < SIZE; row++)
    {
        // Loop over all columns in the image
        for (int col = 0; col < SIZE; col++)
        {
```

```
public class Picture
-----------------------------------------------------------------
       Picture(String filename)   // create a picture from a file
       Picture(int w, int h)      // create a blank w-by-h picture
   int width()                    // return the width of the picture
   int height()                   // return the height of the picture
 Color get(int i, int j)          // return the color of pixel (i,j)
  void set(int i, int j, Color c) // set the color of pixel (i,j) to c
  void show()                     // display the image in a window
  void save(String filename)      // save the image to a file
```

# Charge + Picture client, part 3

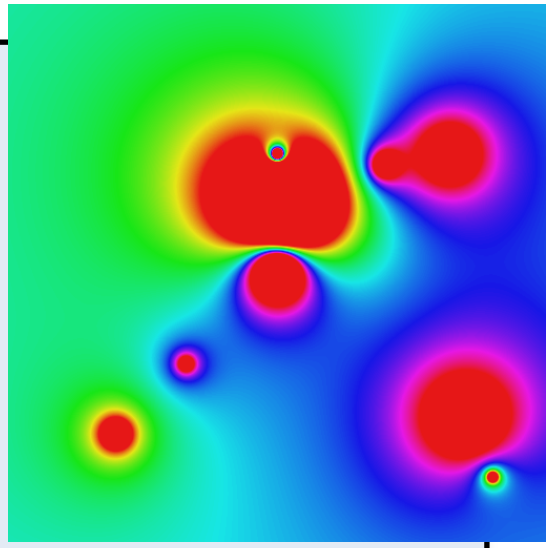
```
// Loop over all rows in the image
for (int row = 0; row < SIZE; row++)
{
    // Loop over all columns in the image
    for (int col = 0; col < SIZE; col++)
    {
        // Loop over all particles, calculating the
        // sum total of charge from all particles.
        double v = 0.0;
        for (int i = 0; i < n; i++)
        {
            double x = (      ) row / SIZE;
            double y = (      ) col / SIZE;
            v += a[i].
        }
```



```
public class Charge
-----------------------------------------
       Charge(double x0, double y0, double q0) // location and charge
double potentialAt(double x, double y)         // potential @ (x,y) due to charge
String toString()                              // string representation
```

# Charge + Picture client, part 3

```
// Loop over all rows in the image
for (int row = 0; row < SIZE; row++)
{
    // Loop over all columns in the image
    for (int col = 0; col < SIZE; col++)
    {
        // Loop over all particles, calculating the
        // sum total of charge from all particles.
        double v = 0.0;
        for (int i = 0; i < n; i++)
        {
            double x = (double) row / SIZE;
            double y = (double) col / SIZE;
            v += a[i].potentialAt(x, y);
        }
```



```
public class Charge
-----------------------------------------
       Charge(double x0, double y0, double q0) // location and charge
double potentialAt(double x, double y)         // potential @ (x,y) due to charge
String toString()                              // string representation
```

# The problem with arrays

- Normal Java arrays:
  - Can hold primitive types
  - Can hold reference types
  - Must declare size when we create

```
int n = StdIn.readInt();
Charge [] a = new Charge[n];
```

```
int n = StdIn.readInt();
double [] x = new double[n];
double [] y = new double[n];
```

- What if we need to add another element?
- What if we want to remove an element?
- What if we don't know how big to create?

# Java library

- Java library
  - Tons of useful classes you can use
  - Only the most important are automatically available without excessive typing:
    - Things like `String`, `System.out`, etc.
- Today:
  - Look at one particular class: **`ArrayList`**
  - Provides dynamically sized arrays

# Java packages

- Packages
  - A collection of classes under one *namespace*
    - Avoids problems if multiple classes have same name
  - Common stuff in `java.lang` package

```
// Two ways to declare a String
String s = "hello world!";
java.lang.String s2 = "hello world!";
```

The `String` class lives in a package called `java.lang`, qualifying is optional for this package

  - `ArrayList` is in a the `java.util` package
    - Add line outside of class: `import java.util.ArrayList;`

# Package java.util

Contains the collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes (a string tokenizer, a random-number generator, and a bit array).

**See:**

Description

| Interface Summary | |
|---|---|
| Collection | The root interface in the *collection hierarchy*. |
| Comparator | A comparison function, which imposes a *total ordering* on some collection of objects. |
| Enumeration | An object that implements the Enumeration interface generates a series of elements, one at a time. |
| EventListener | A tagging interface that all event listener interfaces must extend. |
| Iterator | An iterator over a collection. |
| List | An ordered collection (also known as a *sequence*). |
| ListIterator | An iterator for lists that allows the programmer to traverse the list in either direction, modify the list during iteration, and obtain the iterator's current position in the list. |
| Map | An object that maps keys to values. |
| Map.Entry | A map entry (key-value pair). |
| Observer | A class can implement the `Observer` interface when it wants to be informed of changes in observable objects. |
| RandomAccess | Marker interface used by `List` implementations to indicate that they support fast (generally constant time) random access. |
| Set | A collection that contains no duplicate elements. |
| SortedMap | A map that further guarantees that it will be in ascending key order, sorted according to the *natural ordering* of its keys (see the `Comparable` interface), or by a comparator provided at sorted map creation time. |
| SortedSet | A set that further guarantees that its iterator will traverse the set in ascending element order, sorted according to the *natural ordering* of its elements (see Comparable), or by a Comparator provided at sorted set creation time. |

| Class Summary | |
|---|---|
| AbstractCollection | This class provides a skeletal implementation of the `Collection` interface, to minimize the effort required to implement this interface. |
| AbstractList | This class provides a skeletal implementation of the `List` interface to minimize the effort required to implement this interface backed by a "random access" data store (such as an array). |
| AbstractMap | This class provides a skeletal implementation of the `Map` interface, to minimize the effort required to implement this interface. |
| AbstractSequentialList | This class provides a skeletal implementation of the `List` interface to minimize the effort required to implement this interface backed by a "sequential access" data store (such as a linked list). |
| AbstractSet | This class provides a skeletal implementation of the `Set` interface to minimize the effort required to implement this interface. |
| ArrayList | Resizable-array implementation of the `List` interface. |
| Arrays | This class contains various methods for manipulating arrays (such as sorting and searching). |

# Reversing lines in a file

- Goal: Print lines from StdIn in reverse order
- Problem: We don't know how many to expect


states.txt

```
java ReverseLines < states.txt
Wyoming
Wisconsin
West Virginia
Washington
Virginia
Vermont
Utah
Texas
...
```

# Reversing lines in a file

"I want to type `ArrayList` instead of `java.util.ArrayList` everywhere."

"I want an empty `ArrayList` and I promise to only put `String` objects in it."

"Please add this `String` to my `ArrayList`."

```
import java.util.ArrayList;

public class ReverseLines
{
   public static void main(String[] args)
   {
      ArrayList<String> lines = new ArrayList<String>();

      while (!StdIn.isEmpty())
         lines.add(StdIn.readLine());

      for (int i = lines.size() - 1; i >= 0; i--)
         System.out.println(lines.get(i));
   }
}
```

"How many things are in my list?"

"Please return the $i^{th}$ element of the array."

# Reversing lines in a file

```
import java.util.ArrayList;

public class ReverseLines
{
   public static void main(String[] args)
   {
      ArrayList<String> lines = new ArrayList<String>();

      while (!StdIn.isEmpty())
         lines.add(StdIn.readLine());

      for (int i = lines.size(); i > 0; i--)
         System.out.println(lines.get(i));
   }
}
```

# Reversing lines in a file

```
java ReverseLines < states.txt
Exception in thread "main" java.lang.IndexOutOfBoundsException:
Index: 50, Size:
 50
        at java.util.ArrayList.RangeCheck(ArrayList.java:547)
        at java.util.ArrayList.get(ArrayList.java:322)
        at ReverseLines.main(ReverseLines.java:14)
```

```
   public static void main(String[] args)
   {
      ArrayList<String> lines = new ArrayList<String>();

      while (!StdIn.isEmpty())
         lines.add(StdIn.readLine());

      for (int i = lines.size(); i > 0; i--)
         System.out.println(lines.get(i));
   }
}
```

Just like normal arrays, `ArrayList` objects use 0-based indexing.
The index to the `get()` instance method must be in [0, size() – 1].

# Reversing numbers in a file

- Goal: Reverse doubles read from StdIn
- Problem: We don't know how many numbers

```
1.0 1.5
2.1 2.7 3.0
3.9
5.5
6.7
8.8
9.99
10.553
22.5
33.74
```

nums.txt

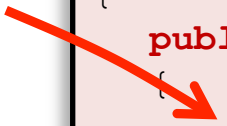
```
java ReverseNums < nums.txt
33.74
22.5
10.553
9.99
8.8
6.7
5.5
3.9
...
```

# Reversing numbers in a file

```
import java.util.ArrayList;

public class ReverseNums
{
   public static void main(String[] args)
   {
      ArrayList<double> nums = new ArrayList<double>();

      while (!StdIn.isEmpty())
         nums.add(StdIn.readDouble());

      for (int i = nums.size() - 1; i >= 0; i--)
         System.out.println(nums.get(i));
   }
}
```

# Reversing numbers in a file: failure

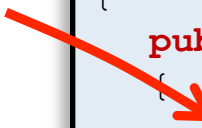
This will not work! Java generics like `ArrayList` only take reference data types, not primitive types like `double`.



```
import java.util.ArrayList;

public class ReverseNums
{
   public static void main(String[] args)
   {
      ArrayList<double> nums = new ArrayList<double>();

      while (!StdIn.isEmpty())
         nums.add(StdIn.readDouble());

      for (int i = nums.size() - 1; i >= 0; i--)
         System.out.println(nums.get(i));
   }
}
```

# Using primitive wrapper classes: success

`Double` class wraps a primitive `double` data type into an object so we can put it into the `ArrayList`.



```
import java.util.ArrayList;

public class ReverseNums
{
   public static void main(String[] args)
   {
      ArrayList<Double> nums = new ArrayList<Double>();

      while (!StdIn.isEmpty())
         nums.add(StdIn.readDouble());

      for (int i = nums.size() - 1; i >= 0; i--)
         System.out.println(nums.get(i));
   }
}
```

# Java primitive wrapper classes

- Wrapper classes
  - Provide a way to use primitives with generics like `ArrayList`
  - Usually primitive type capitalized
  - Stick to primitives unless you actually need a wrapper
    - Less overhead

| Primitive type | Wrapper class |
|---|---|
| byte | Byte |
| short | Short |
| int | Integer |
| long | Long |
| float | Float |
| double | Double |
| char | Character |
| boolean | Boolean |

# Autoboxing

- Autoboxing
  - Java 5.0 converts to/from wrapper classes as needed

This works even though `StdIn.readDouble()` returns a primitive `double` but the `ArrayList` requires a `Double` object.

```
ArrayList<Double> nums = new ArrayList<Double>();

while (!StdIn.isEmpty())
   nums.add(StdIn.readDouble());

for (int i = nums.size() - 1; i >= 0; i--)
   System.out.println(nums.get(i));
```

# Adding and removing

- Adding an element
  - Method: `add(Object o)`
    - Appends the specified object to the end of the list
    - Size of list will increase by one after calling
- Removing an element by index
  - Method: `remove(int index)`
    - Removes element at the specified position in the list
    - Shifts subsequent elements to the left (subtracts one from their indices)
    - Size of list will decrease by one after calling

# Removing (cont'd)

- Removing a specific element
  - Method: `remove(Object o)`
    - Removes the first occurrence of the specified element from the list if present
    - Returns true if the list contained the element
    - Shifts subsequent elements to the left (subtracts one from their indices)
    - Size of list will decrease by one if element found
- Removing all elements
  - Method: `clear()`

# ArrayListExample

```
import java.util.ArrayList;
public class ArrayListExample
{
   public static void main(String[] args)
   {
      ArrayList<String> names = new ArrayList<String>();
      names.add("alice");
      names.add("bob");
      names.add("bob");
      names.add("carol");
      System.out.println(names);

      names.remove(2);
      System.out.println(names);

      names.remove("bob");
      System.out.println(names);
      names.remove("bob");
      System.out.println(names);

      names.clear();
      System.out.println(names);
   }
}
```

```
[alice, bob, bob, carol]
[alice, bob, carol]
[alice, carol]
[alice, carol]
[]
```

# Removing in a loop: failure

```
import java.util.ArrayList;
public class ArrayListRemoveLoop
{
   public static void main(String[] args)
   {
      ArrayList<String> names = new ArrayList<String>();
      names.add("alice");
      names.add("bob");
      names.add("bob");
      names.add("carol");
      System.out.println(names);

      for (int i = 0; i < names.size(); i++)
      {
         if (names.get(i).equals("bob"))
            names.remove(i);
      }
      System.out.println(names);
   }
}
```

```
[alice, bob, bob, carol]

[alice, bob, carol]
```

This doesn't work since when we remove the first "bob", the list is shortened by one inside the loop. We end up skipping over the second "bob".

# Removing in a loop: success

```
import java.util.ArrayList;
public class ArrayListRemoveLoop
{
   public static void main(String[] args)
   {
      ArrayList<String> names = new ArrayList<String>();
      names.add("alice");
      names.add("bob");
      names.add("bob");
      names.add("carol");
      System.out.println(names);

      for (int i = names.size() - 1; i >= 0; i--)
      {
         if (names.get(i).equals("bob"))
            names.remove(i);
      }
      System.out.println(names);
   }
}
```

```
[alice, bob, bob, carol]
[alice, carol]
```

Going backwards through the list fixes the bug. Removing something in the loop doesn't affect what elements we'll see as we move left through the list.

# Fun with ArrayLists


- Goal: Starfield simulation
  - Stars start in center of screen
  - Move in random direction
  - Stars disappear once off screen
  - Periodically add new stars
- Use an ArrayList!
  - Allows us to dynamically add new `Star` objects
  - Allows removal of objects off the screen
    - Otherwise we'd be wasting memory and CPU time

# Summary

- ArrayLists
  - Like an array but extra-powerful
  - Has no fixed sized
  - Add/remove elements dynamically as needed
  - Contains objects of a specified reference type
  - Cannot hold primitive types (e.g. `double, int`)
    - Wrapper objects used instead (e.g. `Double, Integer`)
  - Be careful when you add/remove in a loop!