A programming problem

Consider the following task:

- Double values representing grades are read until the user enters a negative number.
- Then for all of the non-negative values entered, compute the following:
  - The average, minimum and maximum grade
  - The number of grades that are above average and below average
  - The median grade
A programming problem

- If we knew how many grades there were going to be, we could use an array
  - Recall what we know about arrays
    - A sequence of variables of the same data type (e.g. int, double, String, boolean, etc.)
    - Each variable in the array is called an element
    - Array elements are accessed through their index

![Diagram of an array with indices 0 to 9]
A programming problem

- But we don’t know how many grades there will be
  - We just want to keep reading until we find a negative score
  - An array requires us to know before we start using it how big it will be
    - We need to declare it with a specific size

```
double[] grades = new double[30];
```
- What we would like to have is an array that can change size as we use it
In Java, we can use an **ArrayList**

- Just like an array, except it can change size when we need it to
- Also known as a *dynamic array*
- No need to pre-determine the number of elements up front, just add to the array as we need it

```
0
```
A programming problem

- In Java, we can use an **ArrayList**
  - Just like an array, except it can change size when we need it to
    - Also known as a *dynamic array*
    - No need to pre-determine the number of elements up front, just add to the array as we need it
A programming problem

- In Java, we can use an *ArrayList*
  - Just like an array, except it can change size when we need it to
    - Also known as a *dynamic array*
    - No need to pre-determine the number of elements up front, just add to the array as we need it
So what does this have to do with ArrayLists?

- ArrayList is a Java class
  - Part of the Java Standard Library
  - It implements a *dynamic array*

- To use an ArrayList we need to know how to use it:
  - How do we declare it?
  - How can we put things into an ArrayList?
  - How can we read things from an ArrayList?
Let’s consider another programming problem

- We want code that reads in text one line at a time until the user enters the word “stop”
- Then the code should repeat back all of the lines in reverse order
- We do not know how many lines of text there will be before we start
- We can solve this with an ArrayList
import java.util.ArrayList;

public static void main(String [] args) {
    ArrayList<String> stringList = new ArrayList<String>();
    Scanner keyboard = new Scanner(System.in);
    String input = "";
    while (!input.equals("stop")) {
        input = keyboard.nextLine();
        stringList.add(input);
    }
    int i = stringList.size()-1;
    while (i>0) {
        System.out.println(stringList.get(i));
        i = i - 1;
    }
}
import java.util.ArrayList;
...

public static void main(String [] args) {
    ArrayList<String> stringList = new ArrayList<String>();
    Scanner keyboard = new Scanner(System.in);
    String input = "";
    while (!input.equals("stop")) {
        input = keyboard.nextLine();
        stringList.add(input);
    }

    int i = stringList.size()-1;
    while (i>0) {
        System.out.println(stringList.get(i));
        i = i - 1;
    }
}
To use an ArrayList we first need to import it

- ArrayList is a class that is part of the Java standard library
  - Just like Scanner
- We need to tell our program that we’re going to use it
ArrayList – A programming Problem

import java.util.ArrayList;

... 

public static void main(String[] args) {
    ArrayList<String> stringList = new ArrayList<String>();
    Scanner keyboard = new Scanner(System.in);
    String input = "";
    while (!input.equals("stop")) {
        input = keyboard.nextLine();
        stringList.add(input);
    }
    int i = stringList.size() - 1;
    while (i > 0) {
        System.out.println(stringList.get(i));
        i = i - 1;
    }
}

<String>?
What about this `<String>`?

- This syntax indicates that the class you are using is a *generic* class
  - `ArrayList<E> - <E>` is the placeholder for the class
  - Can be used with any class
    - Cannot be used with primitive types (int, boolean, char)
      - We’ll see what to do with these in a moment
  - You can think of it as replacing the array type declaration:
    ```java
    String[] stringArr = new String[10];
    ArrayList<String> stringL = new ArrayList<String>();
    ```
import java.util.ArrayList;
...

public static void main(String [] args) {
    ArrayList<String> stringList = new ArrayList<String>();
    Scanner keyboard = new Scanner(System.in);
    String input = "";
    while (!input.equals("stop")) {
        input = keyboard.nextLine();
        stringList.add(input);
    }
    int i = stringList.size()-1;
    while (i>0) {
        System.out.println(stringList.get(i));
        i = i - 1;
    }
}
ArrayList - Methods

- **add** is the main method used to add elements to an ArrayList:

  ```java
  boolean add(E obj)
  ```

  - Adds an element to the end of the ArrayList
    - E is whatever type we declared with the ArrayList
    - Returns true if it is able to add it, false if not
    - No real equivalent in arrays, since arrays aren’t dynamic
public static void main(String[] args) {
    ArrayList<String> stringList = new ArrayList<String>();
    Scanner keyboard = new Scanner(System.in);
    String input = "";
    while (!input.equals("stop")) {
        input = keyboard.nextLine();
        stringList.add(input);
    }
    int i = stringList.size()-1;
    while (i>0) {
        System.out.println(stringList.get(i));
        i = i - 1;
    }
}
ArrayList - Methods

- **get** is the method used to access elements:

  ```java
  E get(int index)
  ```

- Access an element at position index in the ArrayList:
  - Returns the element at the given index
  - E is the type you used when you declared the ArrayList
  - Equivalent to accessing an array via a subscript:

    ```java
    System.out.println(stringArr[index]);
    System.out.println(stringL.get(index));
    ```

  - Be careful! Just like arrays your code
ArrayList – A programming Problem

import java.util.ArrayList;
...

public static void main(String[] args) {
    ArrayList<String> stringList = new ArrayList<String>();
    Scanner keyboard = new Scanner(System.in);
    String input = "";
    while (!input.equals("stop")) {
        input = keyboard.nextLine();
        stringList.add(input);
    }
    int i = stringList.size()-1;
    while (i>0) {
        System.out.println(stringList.get(i));
        i = i - 1;
    }
}
ArrayList - Methods

- We also need to have a way to check how long our ArrayList is:
  
  ```java
  int size()
  ```

- Returns back the count of elements in the ArrayList
  - Equivalent to using the array length attribute:

  ```java
  int c = stringArr.length;
  int c = stringL.size();
  ```
ArrayList - Methods

- Since ArrayLists are dynamic, there’s a second method for adding elements:

  ```java
def add(int index, E obj)
```

- Adds an element at position index
  - Moves everything currently in the ArrayList from index on to the right one position to “insert” the new element into place
  - E is the type used when you declared the ArrayList
  - No real equivalent in arrays, since arrays aren’t dynamic
ArrayList - Methods

- We also want some way of changing elements at a position:
  
  ```java
  E set(int index, E obj)
  ```

- Changes the element at position index to the new value `obj`
  - `E` is the type used when you declared the ArrayList
  - Returns back what was previously stored at that index
  - Equivalent to using the array subscript:
    ```java
    stringArr[index] = obj;
    stringL.set(index, obj);
    ```
Because ArrayLists are dynamic, we can delete elements from them:

\[ E \text{ remove}(\text{int} \ \text{index}) \]

- Removes the element at index from the list
  - Everything to the right of index is shifted one position left to fill the gap
  - Returns back what we’ve removed
  - No real equivalent in arrays because arrays are not dynamic
Your Turn - ArrayLists

- Write a segment of code that takes an ArrayList of Strings named list and determines:
  - The length of the longest String in the list
  - The length of the shortest String in the list
  - The average length of Strings in the list
    - Compute the average as a double value
    - You can assume that the ArrayList has already been declared and is named list
Your Turn - ArrayLists
Your Turn – ArrayLists (for loop)
Your Turn Again - ArrayLists

- Write a segment of code that takes an ArrayList of Strings named list and removes any Strings that are longer than 5 characters.
```java
int i=0;
while (i<list.size()) {
    String current = list.get(i);
    if (current.length()>5) {
        list.remove(i);
    }
    i = i + 1;
}
```
int i=0;
while (i<list.size()) {
    String current = list.get(i);
    if (current.length()>5) {
        list.remove(i);
    }
    i = i + 1;
}
int i=0;
while (i<list.size()) {
    String current = list.get(i);
    if (current.length()>5) {
        list.remove(i);
    }
    i = i + 1;
}
Removing elements from a list

- We need to be careful removing elements from an ArrayList
  - Remember – when we remove elements, the entire ArrayList shifts into the position of the removed element:

```java
remove(6);
```

```plaintext
0 1 2 3 4 5 6 7 8 9
0 9 6
```
We need to be careful removing elements from an ArrayList

- Remember – when we remove elements, the entire ArrayList shifts into the position of the removed element:

\[ \text{remove}(6); \]
Removing elements from a list

- We need to be careful removing elements from an ArrayList
  - Remember – when we remove elements, the entire ArrayList shifts into the position of the removed element:

    ```java
    remove(6);
    ```

```
  0 1 2 3 4 5 7 8 9
  ↑  ↑  ↑  
  0 6 8
```
Removing elements from a list

- We need to be careful removing elements from an ArrayList

  \texttt{remove(6)};

  - The item at position 6 is removed.
    - Now the item that was at position 7 is at position 6
    - We will end up skipping this element in our loop
Fix this segment of code so that it doesn’t skip any values

```java
int i=0;
while (i<list.size()) {
    String current = list.get(i);
    if (current.length()>5) {
        list.remove(i);
    }
    i = i + 1;
}
```
A programming problem - revisited

Let’s reconsider the problem from the start of these slides:

- Double values representing grades are read until the user enters a negative number
- Then for all of the non-negative values entered, compute the following:
  - The average, minimum and maximum grade
  - The number of grades that are above average and below average
  - The median grade
- For this we need an ArrayList of primitive types
ArrayList Example

```java
import java.util.ArrayList;
...

public static void main(String [] args) {
    ArrayList<Double> gradeList = new ArrayList<Double>();
    Scanner keyboard = new Scanner(System.in);
    double input = Double.parseDouble(keyboard.nextLine());
    while (input>=0.0) {
        gradeList.add(input);
        input = Double.parseDouble(keyboard.nextLine());
    }
    int i = 0;
    int total = 0;
    while (i<gradeList.size()) {
        total = total + gradeList.get(i));
        i = i + 1;
    }
    double average = total/i;
}
ArrayList Example

```java
import java.util.ArrayList;
...

public static void main(String[] args) {
    ArrayList<Double> gradeList = new ArrayList<Double>();
    Scanner keyboard = new Scanner(System.in);
    double input = Double.parseDouble(keyboard.nextLine());
    while (input >= 0.0) {
        gradeList.add(input);
        input = Double.parseDouble(keyboard.nextLine());
    }
    int i = 0;
    int total = 0;
    while (i < gradeList.size()) {
        total = total + gradeList.get(i);
        i = i + 1;
    }
    double average = total / i;
}
```
ArrayLists are used to hold *objects*

- We **cannot** use them to hold *primitive types*

```java
ArrayList<double> gradeList = new ArrayList<double>();
```
Arrays Lists are used to hold objects

- We cannot use them to hold primitive types

```java
ArrayList<Double> gradeList = new ArrayList<Double>();
```

However, each primitive type has a wrapper class that we can use

- `int` → `class` Integer
- `char` → `class` Character
- `double` → `class` Double
Digression – Wrapper classes

- Used to transform a *primitive type* into a *class*
  - Useful for *generic classes* (like collections) that require a class and can’t use a primitive
Digression – Wrapper Classes

- Taking a primitive data element and wrapping it up in a wrapper class is known as *boxing*
  
  ```java
  Double myDouble = new Double(6.0);
  ```

- Taking the primitive out of the wrapper is known as *unboxing*
  
  ```java
  double myPrimitive = myDouble.doubleValue();
  ```

- Java will (usually) automatically do the right thing on its own (*autoboxing*)
  
  ```java
  Double myDouble = 6.0;
  double myPrimitive = myDouble;
  double sum = myDouble + 3.0;
  ```
ArrayLists

- ArrayLists are used to hold objects
  - We cannot use them to hold primitive types

ArrayList<int> intList = new ArrayList<int>();
Collections - ArrayList

- ArrayLists are used to hold objects
  - We cannot use them to hold primitive types

```java
ArrayList<Double> gradeList = new ArrayList<Double>();
```

- Instead we use the wrapper object:

```java
ArrayList<Double> gradeList = new ArrayList<Double>();
```
If we need an ArrayList of primitives, we use the corresponding wrapper class instead

- But we can use *autob boxing* to make our life easier adding and examining elements

```java
class Main {
    public static void main(String[] args) {
        ArrayList<Double> gradeList =
            new ArrayList<Double>();
        intList.add(12.0);
        double firstScore = gradeList.get(0);
    }
}
```
Your Turn - ArrayLists

- Write a piece of code that:
  - Declares a new ArrayList of integers named nums
  - Puts the even numbers from 2 through 20 into this list
Your Turn - ArrayLists

- Write a piece of code that:
  - Uses an ArrayList of Integers named list1 to build a new ArrayList of Integers named list2
  - The elements of list2 should be in reverse order from the elements that are in list1
ArrayLists and Methods

Like arrays, ArrayLists can be passed as parameters to methods and can be returned by methods (functions) e.g.

```java
private static void printList(ArrayList<Double> al)
```

```java
private static ArrayList<Integer> copyList(ArrayList<Integer> al)
```

```java
private static int indexOfMin(ArrayList<Integer> al)
```

```java
private static void sort(ArrayList<Character> al)
```
ArrayList Example

- On the following slide is some code that uses an array
  - Rewrite this code so that it uses an ArrayList instead
public class ArrayExample {
    public static int countOccurrences(char [] array, char ch) {
        int chCount=0;
        for (int loopCount=0; loopCount<array.length; loopCount++) {
            if (array[loopCount] == ch) {
                chCount=chCount+1;
            }
        }
        return chCount;
    }
    public static void main(String [] args) {
        char [] myChars = new char[4];
        char myCh = 'b';
        int count = countOccurrences(myChars, myCh);
        System.out.println("The number of occurrences is: "+count);
    }
}
ArrayList Example – One solution