CSE 143
Lecture 4

`ArrayList`

Reading: 10.1

slides created by Marty Stepp
http://www.cs.washington.edu/143/

Handling errors

- Currently our `ArrayIntList` class allows the user to do some bad things, like adding/getting elements beyond the end of the list (but within the capacity).

```
// Precondition: 0 <= index < size
public void get(int index) {
    return elementData[index];
}
```

- If we wanted to prevent such behavior, how could we do it?

Throwing exceptions (4.5)

```
throw new ExceptionType();
throw new ExceptionType("message");
```

- Causes the program to immediately crash with an exception.
  - Why might this be a good thing?
- Common types of exceptions:
  - ArithmeticException, ArrayIndexOutOfBoundsException, FileNotFoundException, IllegalArgumentException, IllegalStateException, IOException, NoSuchElementException, NullPointerException, RuntimeException, UnsupportedOperationException

```
public void get(int index) {
    if (index >= size) {
        throw new ArrayIndexOutOfBoundsException();
    }
    return elementData[index];
}
```

Exercise

- Write a program that reads a file and displays the words of that file as a list.
  - First display all words.
  - Then display them with all plural words (end in "s") capitalized.
  - Then display them in reverse order.
  - Then display them with all plural words removed.

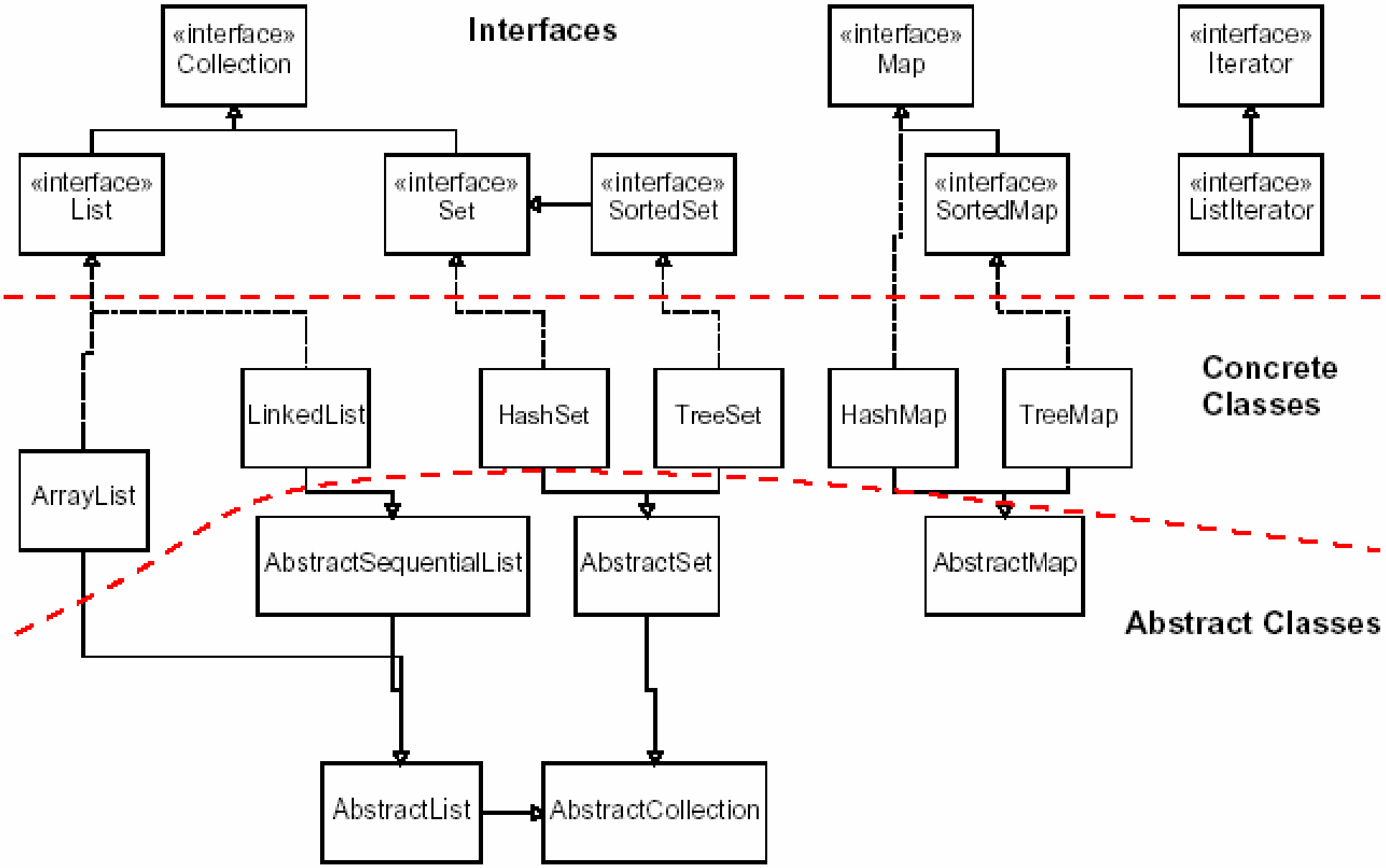
- These kinds of tasks are similar to those we performed with the `ArrayIntList`, but we are using `Strings`, not `ints`.
  - Should we write an `ArrayStringList` class?

Java Collection Framework

- Java includes a large set of powerful collection classes.
- We will learn to use several of these classes in CSE 143.
- The most basic, `ArrayList`, is essentially the same as our `ArrayIntList` but can store any type of value.
- All collections are in the `java.util` package.

```
import java.util.*;
```

Java collection framework



Type Parameters (Generics)

```
ArrayList<Type> name = new ArrayList<Type>();
```

- When constructing an `ArrayList`, you must specify the type of elements it will contain between < and >.
  - We say that the `ArrayList` class accepts a *type parameter*, or that it is a *generic* class.

```
ArrayList<String> names = new ArrayList<String>();
names.add("Marty Stepp");
names.add("Stuart Reges");
```

ArrayList methods (10.1)

`add(`**value**`)`	appends value at end of list
`add(`**index**`, `**value**`)`	inserts given value at given index, shifting subsequent values right
`clear()`	removes all elements of the list
`indexOf(`**value**`)`	returns first index where given value is found in list (-1 if not found)
`get(`**index**`)`	returns the value at given index
`remove(`**index**`)`	removes/returns value at given index, shifting subsequent values left
`set(`**index**`, `**value**`)`	replaces value at given index with given value
`size()`	returns the number of elements in list
`toString()`	returns a string representation of the list such as `"[3, 42, -7, 15]"`

ArrayList methods 2

`addAll(`**list**`)` `addAll(`**index**`, `**list**`)`	adds all elements from the given list to this list (at the end of the list, or inserts them at the given index)
`contains(`**value**`)`	returns true if given value is found somewhere in this list
`containsAll(`**list**`)`	returns true if this list contains every element from given list
`equals(`**list**`)`	returns true if given other list contains the same elements
`iterator()` `listIterator()`	returns an object used to examine the contents of the list (seen later)
`lastIndexOf(`**value**`)`	returns last index value is found in list (-1 if not found)
`remove(`**value**`)`	finds and removes the given value from this list
`removeAll(`**list**`)`	removes any elements found in the given list from this list
`retainAll(`**list**`)`	removes any elements *not* found in given list from this list
`subList(`**from**`, `**to**`)`	returns the sub-portion of the list between indexes **from** (exclusive) and **to** (inclusive)
`toArray()`	returns an array of the elements in this list

Learning about classes

- The Java API Specification is a huge web page containing documentation about every Java class and its methods.
  - The link to the API Specs is on the course web site.

Exercises

- Write a method `tokenize` that accepts a file `Scanner` and reads the words of the file into an `ArrayList` and returns it.

- Write a method `capitalizePlurals` that accepts an `ArrayList` of strings and replaces every word ending with an "s" with its uppercased version.

- Write a method `reverse` that reverses the order of the elements in an `ArrayList` of strings.

- Write a method `removePlurals` that accepts an `ArrayList` of strings and removes every word in the list ending with an "s", case-insensitively.

Modifying while looping

- Consider the following flawed pseudocode algorithm to remove plural elements from a list:

```
removePlurals(list) {
    for (int i = 0; i < list.size(); i++) {
        if element i is plural, remove it.
    }
}
```

  - What does the algorithm do wrong?

index	*0*	*1*	*2*	*3*	*4*	*5*
value	`"she"`	`"sells"`	`"seashells"`	`"by"`	`"the"`	`"seashore"`
size	6					

`ArrayList` of primitives?

- The type you specify when creating an `ArrayList` must be an object type; it cannot be a primitive type.
  - The following is illegal:

```
// illegal -- int cannot be a type parameter
ArrayList<int> list = new ArrayList<int>();
```

- But we can still use `ArrayList` with primitive types by using special classes called *wrapper* classes in their place.

```
ArrayList<Integer> list = new ArrayList<Integer>();
```

Wrapper classes

Primitive Type	Wrapper Type
int	Integer
double	Double
char	Character
boolean	Boolean

- A wrapper is an object whose sole purpose is to hold a primitive value.

- Once you construct the list, use it with primitives as normal:

```
ArrayList<Double> grades = new ArrayList<Double>();
grades.add(3.2);
grades.add(2.7);
...
```

Exercise

- Last week we wrote a program that used our `ArrayIntList` to read a file of numbers and print them in reverse order. Refactor that program to use `ArrayList`.

- Then modify the program to print a "stretched" version of the list of numbers, where every number is replaced with two elements, each of which is half as large as the original.
  - Write a `stretch` method to help with this.