ArrayLists

AP Computer Science

Arrays

- You can probably see some problems with using Arrays
  - One is where you do not know how many items you will be storing
  - Another is if you need to add or remove items often
- This leads us to a different option for storing information: ArrayLists

Collections

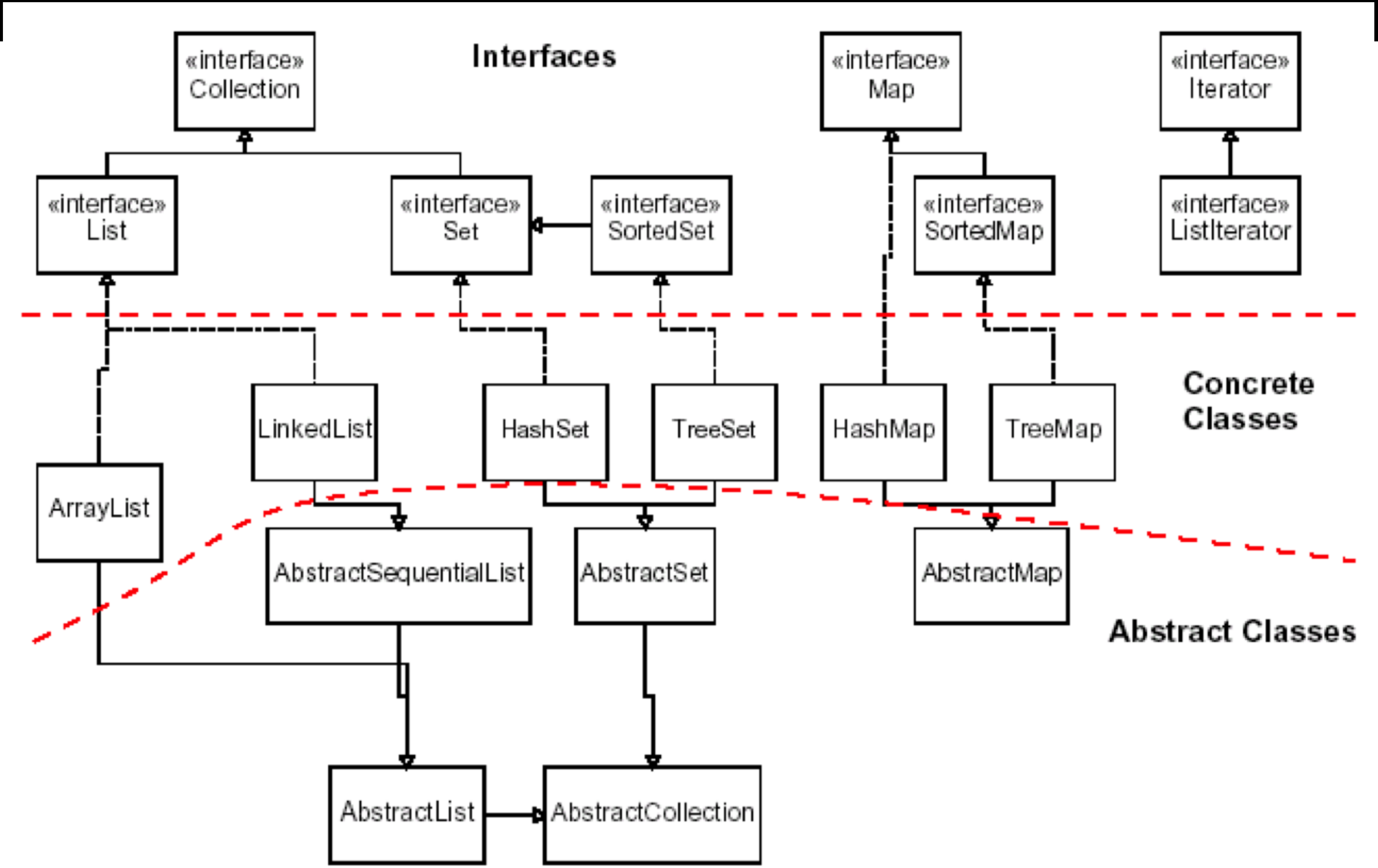
- collection: an object that stores data; a.k.a. "data structure"
    - the objects stored are called elements
    - some collections maintain an ordering; some allow duplicates
    - typical operations: add, remove, clear, contains (search), size

Collections

- examples found in the Java class libraries:
  - ArrayList, LinkedList, HashMap, TreeSet, PriorityQueue
- all collections are in the java.util package

import java.util.*;

Collections framework



ArrayLists

ArrayLists

- Rather than creating an array of boxes, create an object that represents a "list" of items. (initially an empty list.)

  []

- You can add items to the list.
  - The default behavior is to add to the end of the list.

  [hello, ABC, goodbye, okay]

- The list object keeps track of the element values that have been added to it, their order, indexes, and its total size.
  - Think of an "array list" as an automatically resizing array object.
  - Internally, the list is implemented using an array and a size field.

Declaration of ArrayLists

- To declare an array of a specified **type** with a given **name**:

```
ArrayList<type> name;
```

- Example with a list of type **Integer**:

```
ArrayList<Integer> list;
```

- When constructing an ArrayList, you must specify the type of elements it will contain between < and >.

- This is called a type parameter or a generic class.
- Allows the same ArrayList class to store lists of different types.

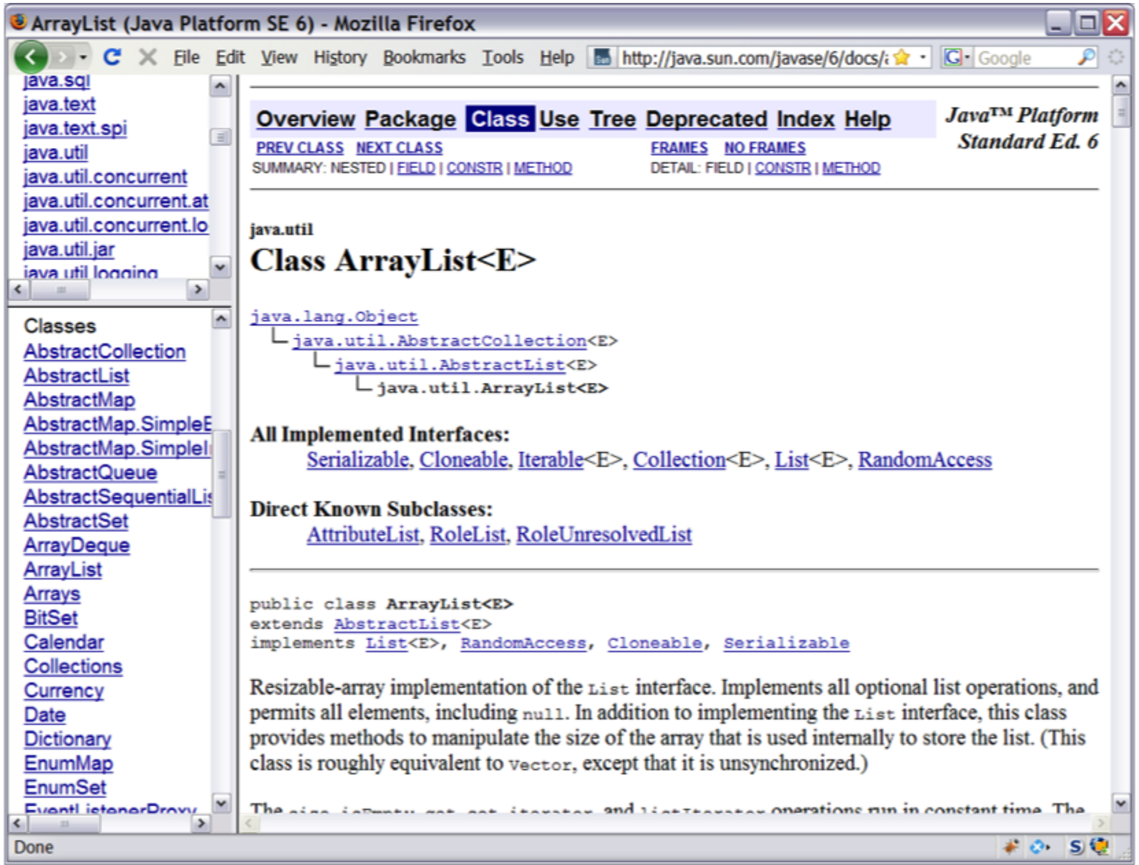
Instantiation of ArrayLists

- When you declare an ArrayList, you are creating a variable that can hold an ArrayList
- At first, it holds nothing, also know as **null**
- To use it, you have to instantiate an ArrayList:

```
// declaration
ArrayList<Integer> list;
// instantiation
list = new ArrayList<Integer>();
```

Learning about classes

- The Java API Specification is a huge web page containing documentation about every Java class and its methods.



java.util

Class ArrayList<E>

```
java.lang.Object
  └java.util.AbstractCollection<E>
      └java.util.AbstractList<E>
          └java.util.ArrayList<E>
```

All Implemented Interfaces:

Serializable, Cloneable, Iterable<E>, Collection<E>, List<E>, RandomAccess

Direct Known Subclasses:

AttributeList, RoleList, RoleUnresolvedList

```
public class ArrayList<E>
extends AbstractList<E>
implements List<E>, RandomAccess, Cloneable, Serializable
```

Resizable-array implementation of the `List` interface. Implements all optional list operations, and permits all elements, including `null`. In addition to implementing the `List` interface, this class provides methods to manipulate the size of the array that is used internally to store the list. (This class is roughly equivalent to `Vector`, except that it is unsynchronized.)

The `size`, `isEmpty`, `get`, `set`, `iterator`, and `listIterator` operations run in constant time. The

Arraylists Methods

ArrayList Methods

add(**value**)	appends value at end of list
add(**index**, **value**)	inserts given value just before the given index, shifting subsequent values to the 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 to the 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

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** (inclusive) and **to** (exclusive)
toArray()	returns the elements in this list as an array

Accessing ArrayList Elements

- You can access and modify an element of an ArrayList by using the add(), set, and get() methods

```
ArrayList<Integer> list;
list = new ArrayList<Integer>();
list.add(142);
list.set(0, 90);
System.out.println(list.get(0));
list.add(0, 87);
System.out.println(list.toString());
```

Output

```
90
[87, 90]
```

- Indexing starts at 0 and stops at 1 less than the size
- The index can be any number, variable, or expression that equates to an **integer**

Size of an ArrayList

- You can use the **size()** method to find out how many elements are in the ArrayList.
- Please note the difference from the **length()** method for Strings, the **length** member for Arrays, and **size()** for ArrayLists.

```
ArrayList<Integer> list;
list = new ArrayList<Integer>();
System.out.println("List has " +
list.size() + " elements");
```

Searching an ArrayList

- You can search an ArrayList by using the indexOf() or contains() methods

```
ArrayList<String> slist;
slist = new ArrayList<String>();
slist.add("Joe");
slist.add("Sue");
System.out.println(slist.indexOf("Sue"));
System.out.println(slist.indexOf("Nate"));
System.out.println(slist.contains("Joe"));
```

Output

```
1
-1
true
```

- indexOf() returns an integer value of the index, or -1
- contains() returns a boolean result

Removing from an ArrayList

- You can remove from an ArrayList by using the remove() methods

```
ArrayList<String> slist;
slist = new ArrayList<String>();
slist.add("Joe");
slist.add("Sue");
slist.add("Kirk");
slist.remove("Joe");
System.out.println(sslist.toString());
sslist.remove(0);
System.out.println(sslist.toString());
```

Output

```
[Sue, Kirk]
[Kirk]
```

Removing from an ArrayList

- How can you remove a specific integer value from an ArrayList?

```
ArrayList<Integer> ilist;
ilist = new ArrayList<Integer>();
ilist.add(2);
ilist.add(1);
ilist.add(0);
ilist.remove((Integer)0);
System.out.println(ilist.toString());
ilist.remove(0);
System.out.println(ilist.toString());
```

Output

```
[2, 1]
[1]
```

Clearing an ArrayList

- You can delete everything from an ArrayList using clear()?

```
ArrayList<Integer> ilist;
ilist = new ArrayList<Integer>();
ilist.add(2);
ilist.add(1);
ilist.add(0);
ilist.clear();
System.out.println(ilist.toString());
```

Output
[]

Loops and ArrayLists

for loop with an ArrayList

```
ArrayList<Integer> ilist;
ilist = new ArrayList<Integer>();
ilist.add(2);
ilist.add(1);
ilist.add(0);
for( int i = 0; i < list.size(); i++ )
{
   System.out.print(ilist.get(i) + " ");
}
```

Output

```
2 1 0
```

for each loop

- This loop goes through some list of items
- In this case the variable x stores the actual value of an array element

```
ArrayList<Double> ilist;
ilist = new ArrayList<Double>();
ilist.add(2.0);
ilist.add(1.0);
ilist.add(0.5);
for(double x : ilist)
{
    System.out.print(x + " ");
}
```

Output

```
2.0 1.0 0.5
```

Removing multiple items

- What is the output?

```
ArrayList<Integer> ilist;
ilist = new ArrayList<Integer>();
ilist.add(1);
ilist.add(1);
ilist.add(0);
for(int x = 0; x < ilist.size(); x++){
   if(ilist.get(x).equals(1)){
      ilist.remove(x);
   }
}
System.out.println(ilist.toString());
```

Output
[1, 0]

Removing multiple items

- What is the output?

```
ArrayList<Integer> ilist;
ilist = new ArrayList<Integer>();
ilist.add(1);
ilist.add(1);
ilist.add(0);
for(int x = ilist.size()-1; x >= 0; x--){
   if(ilist.get(x).equals(1)){
      ilist.remove(x);
   }
}
System.out.println(ilist.toString());
```

It is important to start at the end of the ArrayList and go towards the front when removing items.

Output
[0]

ArrayList as a parameter

```
public static void name(ArrayList<Type> name) {
```

- Example:

```
// Removes all plural words from the given list.
public static void removePlural(ArrayList<String> list){
      for (int i = list.size()-1; i >= 0; i--) {
      if (list.get(i).endsWith("s")) {
            list.remove(i);
      }
      }
}
```

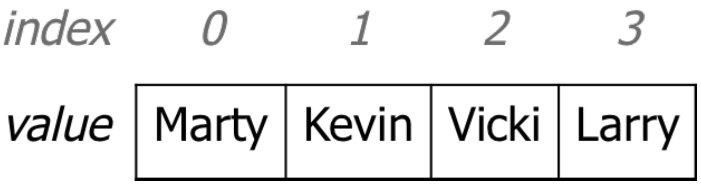
- You can also return a list:

```
public static ArrayList<Type> methodName(params)
```

ArrayList Index Out of bounds

- Legal indexes are between **0** and the **list's size() - 1**.
  - Reading or writing any index outside this range will cause an `IndexOutOfBoundsException`.

```
ArrayList<String> names = new ArrayList<String>();
names.add("Marty");   names.add("Kevin");
names.add("Vicki");   names.add("Larry");
System.out.println(names.get(0));        // okay
System.out.println(names.get(3));        // okay
System.out.println(names.get(-1));       // exception
names.add(9, "Aimee");                   // exception
```



index	*0*	*1*	*2*	*3*
value	Marty	Kevin	Vicki	Larry

Primitives and wrapper classes

ArrayList of primitives

- The type you specify when creating an ArrayList must be an object type; it cannot be a primitive type.

```
// 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.

```
// creates a list of ints
ArrayList<Integer> list = new ArrayList<Integer>();
```

Wrapper classes

- 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);
```

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

ArrayLists with no type

- There is the ability to have an ArrayList with no specified type

```
    ArrayList list = new ArrayList();
```

- To use elements from this ArrayList you need to type case them to the type you are working with

```
    ((String)list.get(0)).equals(“hello”)
```

ArrayLists of non-generic types

Student Class

```
public class Student{
   private String name;
   public Student(){
      name = "";
   }
   public Student(String n){
      name = n;
   }
   public void setName(String n){
      name = n;
   }
   public String getName(){
      return name;
   }
   public String toString(){
      return " " + name;
   }
}
```

ArrayLists of non-generic types

- ArrayLists can store any object type
- You specify the class name of the type you wish to store

```
ArrayList<Student> bus;
bus = new ArrayList<Student>(
bus.add(new Student("Joe"));
bus.add(new Student("Jane"));
bus.add(new Student("John"));
bus.get(0).setName("Ralph");
System.out.println(bus.get(0).getName());
bus.get(1).setName("Dale");
bus.get(2).setName("Sara");
System.out.println(bus.toString());
```

Output

```
Ralp
[Ralph, Dale, Sara]
```

Dog Class

```
public class Dog{
   private String name;
   private int age;
   public Dog(){
      name = "";
      age = 9;
   }
   public Dog(String n, int a){
      name = n;
      age = g;
   }
   // modifier and accessor methods not shown
   public String toString(){
      return name + " " + age;
   }
}
```

ArrayLists of non-generic types

- ArrayLists can store any object type
- You specify the class name of the type you wish to store

```java
ArrayList<Dog> kennel;
kennel = new ArrayList<Dog>();
kennel.add(new Dog("Rover", 3));
kennel.add(new Dog("Spot", 5));
kennel.add(new Dog("Duke", 2));
for( int i = 0; i < kennel.size(); i++ )
{
   System.out.println(kennel.get(i) + " ");
}
```

Output

```
Rover 3
Spot 5
Duke 2
```

What method is called by this print statement?

Collections

Collections Methods

`sort(`**list**`)`	sorts **list** in ascending order based on compareTo method
`binarySearch(`**find**`,` **list**`)`	searches for **find** in **list** (list must be sorted)
`rotate(`**list**, **num**`)`	shifts items in **list** left or right **num** locations
`reverse(`**list**`)`	reverses items in **list**

Collections.sort()

- We will cover sorting in detail later in the course
- Here is how you can sort an ArrayList
- You will need to add this import statement

```
import java.util.Collections;
```

```
ArrayList<Integer> list;
list = new ArrayList<Integer>();
list.add(7); list.add(2);
list.add(9); list.add(4);
Collections.sort(list);
System.out.println(list.toString());
```

Output
[2, 4, 7, 9]

- This only works with the generic (built in) types (for now)

Collections.binarySearch()

- We will cover searching in detail later in the course
- Here is how you can search an ArrayList
- You will need to add this import statement

```
import java.util.Collections;
```

```
ArrayList<Integer> list;
list = new ArrayList<Integer>();
list.add(7); list.add(2);
list.add(9); list.add(4);
Collections.sort(list); //this must happen first
System.out.println(Collections.binarySearch(list, 7));
System.out.println(Collections.binarySearch(list, 5));
```

Output

```
2
-3
```

- If the item is in the list it returns the index
- If it is not in the list, t returns -1 + - index of where the item would be

Collections.rotate()

- You will need to add this import statement

```
import java.util.Collections;
```

```
ArrayList<Integer> list;
list = new ArrayList<Integer>();
list.add(7); list.add(2);
list.add(9); list.add(4);
Collections.sort(list);
Collections.rotate(list, 2);
System.out.println(list.toString());
```

Output

```
[7, 9, 2, 4]
```

Collections.reverse()

- You will need to add this import statement

```
import java.util.Collections;
```

```
ArrayList<Integer> list;
list = new ArrayList<Integer>();
list.add(7); list.add(2);
list.add(9); list.add(4);
Collections.sort(list);
Collections.reverse(list);
System.out.println(list.toString());
```

Output

```
[9, 7, 4, 2]
```

Additional Collections methods

- To see other methods you can use the Java API for Collections