Advanced Programming Techniques in Java

ArrayList

Lecture 16

Class Objectives

- `ArrayList` (section 10.1)

Review: Summary of Fea
Classes,Abstract Classe

Property	Actu
Instances (objects) of this can be created.	
This can define instance variables and methods.	
This can define constants.	
The number of these a class can extend.	0
The number of these a class can implement.	
This can extend another class.	
This can declare abstract methods.	
Variables of this type can be declared.	

Review: The `ArrayList` class

- An `ArrayList` object uses an array to store its va

- Think of it as an auto-resizing array that can hold ar
  convenient methods
- It maintains most of the benefits of arrays, such as
- It frees us from some tedious operations on arrays,
- To use `ArrayList` remember to import `java.ut`
- We can declare arrays of different types e.g., `int[`
  class has similar flexibility

Review: Java Generics

- Used to make an object usable for any types, while Java allows
- Normally we must be specific about the type we're p us to make this variable
- Useful for making data structures, which we want to to insert into them

Review: Java Generics

- We can make this code "generic"

```
public class PointBox{
    private Point p; public
    void put(Point p){ this.p
    = p;
    }
    public Point get( ){
        return this.p;
    }
}
```

```
publ
    T
    p
    =
    }
    p

    }
}
```

- Now we can put an object of any type “T” into the b

Review: How to use this “Generic

- In the `main` method, you can initialize a `Box` doing the following:

```
Box<TYPE> name = new Box<TYPE>( );
```

- e.g: `Box<String> stringBox = new Box<Str`
- or: `Box<Point> pointBox = new Box<Point`

our code can be used for any type!

Example Code

```
public class Main{
  public static void main(String[ ]
    Point p2 = new Point(0,5);
    System.out.println("Making a box
    Box<Point> b1 = new Box<Point>(
    b1.put(p2);
    System.out.println(b1.get().getY

  }
}
```

Makes

Java doesn't complain that we do `.getY()` on
the box, since we told it that the object was goin

In summary …

- **Generic class** is a type in Java that is written to
- Generic (or "parameterized") classes were added to J safety of Java's collections
- A parameterized type has one or more other types' na

WhyUse Generic Colle

- Better type-checking: catch m

```
// without Generics
List list = new ArrayLis
list.add("hello");

// With Generics
List<Integer> list = new
list.add("hello"); // wi
```

  them earlier
- Documents intent
- Avoidstheneed to downcast fr

```
List list = new ArrayList();
list.add("hello");
String s = (String) list.get(0);
```

When re-written to use generics, the code doe

```
List<String> list = new ArrayList<Stri
list.add("hello");
String s = list.get(0); // no cast
```

Review: The `ArrayList` class

- An `ArrayList` object uses an array to store its va
- Think of it as an auto-resizing array that can hold ar methods

- It maintains most of the benefits of arrays, such as
- It frees us from some tedious operations on arrays,
- To use `ArrayList` remember to import `java.ut`
- We can declare arrays of different types e.g., `int[`
  class has **similar flexibility**

Wrapper Classes for Primit

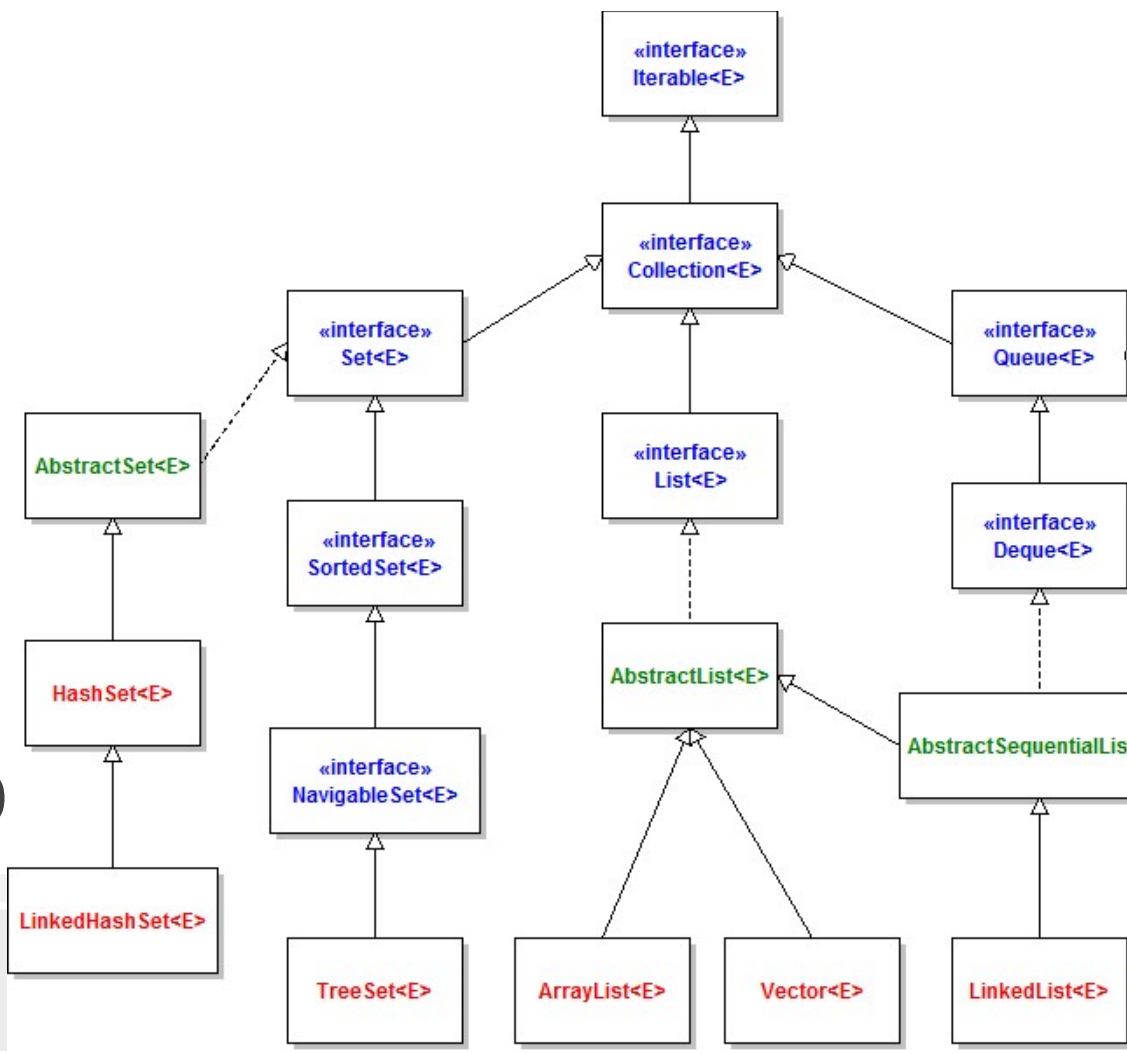
- Primitive numeric types are not objects, but so
  processed like objects
  - When?
- Java provides *wrapper classes* whose objects
  - `Float, Double, Integer, Boolean,`
    - They provide constructor methods to create ne
    - Also provide methods to “unwrap”

Wrapper classes

Primitive Type	Wrapp
`int`	`Integ`
`double`	`Doubl`
`char`	`Chara`
`float`	`Float`
`boolean`	`Boole`

- A wrapper is an object whose sole purpose is to hold

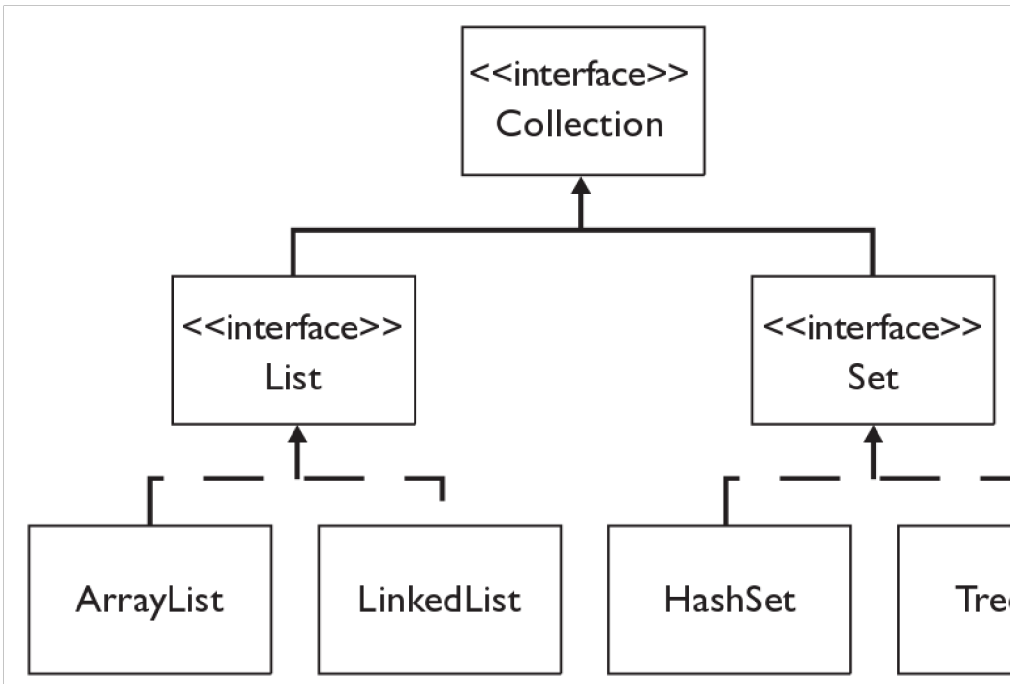
- Once you construct the list, use it with primitives as n


«interface»
Iterable<E>
«interface»
Collection<E>
«interface»
Set<E>
«interface»
Queue<E>
AbstractSet<E>
«interface»
List<E>
«interface»
SortedSet<E>
«interface»
Deque<E>
HashSet<E>
AbstractList<E>
«interface»
NavigableSet<E>
AbstractSequentialLis
LinkedHashSet<E>
TreeSet<E>
ArrayList<E>
Vector<E>
LinkedList<E>

O

```
public class ArrayList<E> extends Abstract
```

Java collections framework



`ArrayList` of any type of obje

- When constructing an `ArrayList`, you must speci between `< >`
- By making the `ArrayList` class a Generic class, the sa different types

- **Syntax**: `ArrayList<Type> name = new ArrayLi`

```
ArrayList<String> names = new ArrayList<St
```

- Java 7's shorter "diamond operator" syntax

```
ArrayList<String> names = new ArrayList
```

`ArrayList` of any type of ob

- You can store any type of object in an `ArrayList`
- `ArrayList<Point> points = new ArrayList<Po`
- The `points` list will manipulate and return `Points`
- `ArrayList<Color> points = new ArrayList<Co`
- The `points` list will manipulate and return `Colors`

Adding elements

- Elements are added dynamically to the end of the

```
ArrayList<String> list = new
ArrayList<String>(); list.add("Brandeis")
list.add("Department"); list.add("Compute
Science");
```

- What we store after each addition:

```
[]
[Brandeis]
[Brandeis, Department]
[Brandeis, Department, Computer Science]
```

values
the list

Passing correct object types

- Java makes sure you add values of appropriate ob
  exception

```
ArrayList<String> list = new Arra
Point p = new Point();
list.add(p);
```

- This does not compile because a `String` object i

Printing `ArrayLists`

- Unlike arrays, printing an `ArrayList` is easier sinc
  class overrides the `toString` method

```
ArrayList<String> list = new ArrayList
System.out.println("list = " + list);
list.add("Brandeis");
System.out.println("list = " + list);
list.add("Department");
System.out.println("list = " + list);
list.add("Computer Science");
System.out.println("list = " + list);
```

- Output:

you can print it even when it

```
list = [] list = [Brandeis] list = [Brandeis,
Department] list = [Brandeis, Department,
Computer Science]
```

More on adding elements

- You can add a value at particular **index** in the list by **`add(int index, E element)`**
- It inserts the specified element at the specified position in the right
- Example: `list.add(1, "cs12");`

before: `list = [Brandeis, Department, Compu`

after: `list = [Brandeis, cs12, Department,`

`Science]`

Removing elements

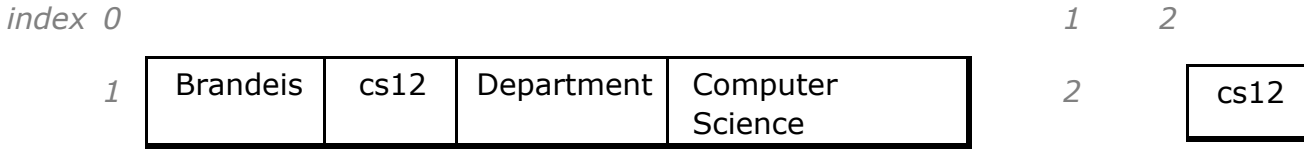
- Elements can also be removed by index:

```
System.out.println("before remove list = "
list); list.remove(0); list.remove(1);
System.out.println("after remove list = "
```

before : `list = [Brandeis, cs12, Department, Co`

after: `list = [cs12,Computer Science]`

- Notice that as each element is removed, the others hole
- Therefore, the second remove gets rid of `Department`, n



- You can call the **`size()`** method to get the numbe

Issues with dynamic addition

- Assume you have an `ArrayListwords words =` `and, seven, years, ago]`
  - You want to add '~'

  Solution 1:

```
for (int i=0; i < words.size(); i++) {
      words.add(i,'~');
}
```

- Does this work?

Issues with dynamic addition

- Assume you have an `ArrayListwords words = years, ago]`
  - You want to add '~' before each

```
for (int i=0; i < words.size(); i++) {
      words.add(i,'~');
}
```

- Does this work?
- Infinite loop: it will never stop (out of memory error)

```
words = [~, four, score, and, seven, yea
words = [~,~, four, score, and, seven, y
words = [~, ~, ~, four, score, and, seve
ago] ....
```

Solution 1

- The problem was that we ignored the shifting o
- Since we add '~' we want to move 2 positions to the

solution:

Solution 1

- The problem was that we ignored the shifting o
- Since we add '~' we want to move 2 positions to the

solution:

```
for (int i=0; i < words.size(); i+=2) {
      words.add(i,'~');
}
```

```
words = [~, four, score, and, seven, years, ag
words = [~, four, ~,score, and, seven, years,
words = [~, four, ~,score, ~, and, seven, year
....
```

```
words = [~, four, ~, score, ~, and, ~,seven, ~
```

“Backwards” solution (solution 2)

- You can visit the elements from right to left
- Ensures that any changes you make occur on element

```
for (int i= words.size()-1; i>=0; i--) {
      words.add(i,'~');
}
```

```
words = [four, score, and, seven, years, ~, ag
words = [four, score, and, seven, ~, years, ~,
....
```

```
words = [~, four, ~, score, ~, and, ~,seven, ~
```

Issues with dynamic removal

- We now want to redo this operation (remove '~')
- Write code that removes every other element starti

```
four, ~, score, ~, and, ~,seven, ~, years,
```

- Does this work? Why?

```
for (int i=0; i < words.size(); i+=2) {
      words.remove(i);
}
```

Issues with dynamic removal

- We now want to redo this operation (remove '~')
- Write code that removes every other element startin

```
four, ~, score, ~, and, ~,seven, ~, years,
```

- Does this work? Why?

```
for (int i=0; i < words.size(); i+=2) {
      words.remove(i);
}
```

- Output `words = [four, ~, score, ~, and, ~,s`
  `words = [four, ~, ~, and, ~,seven, ~, year`
  `~, ~, and, seven, ~, years, ~, ago] ...`

Solution 1

- Again, dynamic shifting causes the problem
- Once you remove an element, all the rest are shifted to th

Correct solution:

```
for (int i=0; i < words.size(); i++) {
      words.remove(i);
}
```

- Output `words = [four, score, and, seven, ye`

  `ago]`