ArrayList and other Collections

generics, iteration, & efficiency considerations
We’ve seen arrays ...

String [ ] a = new String [6];

ArrayLists are similar, but stretchy ...

ArrayList<String> a = new ArrayList<String>();
a.add(“Vogon”);
Stretchy

ArrayList<String> a = new ArrayList<String>();

a.add("Alpha");

a.add("Beta");
**Stretchy**

```java
ArrayList<String> a = new ArrayList<String>();

a.add("Alpha");

a.add("Beta");
```

- `a.isEmpty()` returns `true` and `a.size()` returns `0`.
- `a.isEmpty()` returns `false` and `a.size()` returns `1`.
- `a.isEmpty()` returns `false` and `a.size()` returns `2`.

---

**UNIVERSITY OF OREGON**

- **CIS 210**
```
ArrayList<String> a = new ArrayList<String>();

a.add("Alpha");
a.add("Beta");
```

- `a.size()` returns 0
  - `a.get(0)` is an error

- `a.size()` returns 1
  - `a.get(0)` returns “Alpha”
  - `a.get(1)` is an error

- `a.size()` returns 2
  - `a.get(0)` returns “Alpha”
  - `a.get(1)` returns “Beta”
ArrayList\<E\> implements Iterable

public Iterator\<E\> iterator();

interface Iterator\<E\> {
  boolean hasNext();
  E next();
  void remove();
}


**Iterating through an ArrayList**

```java
import java.util.ArrayList;
import java.util.Iterator;
...
public static void example() {
    ArrayList<String> myList =
        new ArrayList<String>();
    myList.add("abra");
    ...
    Iterator<String> elements =
        myList.iterator();
    while (elements.hasNext()) {
        String s = elements.next();
        System.out.println(s);
    }
}
```
Such a common pattern ...

```java
Iterator<String> elements = myList.iterator();

while (elements.hasNext()) {
    String s = elements.next();
    ...
}
```
... that Java provides a shorthand

```java
for (String s: myList) {
    ...
}
```

Asking for the iterator, testing with hasNext(), and getting an element with next() are all implicit in the shorthand “foreach” loop
So our code becomes ...

```java
public static void example() {
    ArrayList<String> myList =
        new ArrayList<String>();
    myList.add("abra");
    ...

    for (String s: myList) {
        System.out.println(s);
    }
}
```
How about an ArrayList of int?

ArrayList<int> ali = new ArrayList<int>();

NOPE.

The <E> in ArrayList<E> must be a class of objects.

int is not an object

Integer is a class that wraps ints

(see Reges & Stepp for details)
**Efficiency considerations**

```java
ArrayList<String> m = new ArrayList<String>();

for (int i=0; i < 100; ++i) {
    m.add(0, "boo");
}
```

*much more expensive than*

```java
for (int i=0; i < 100; ++i) {
    m.add("boo");
}
```
Efficiency considerations

ArrayList<String> m = new ArrayList<String>();

for (int i=0; i < 100; ++i) {
    m.add(0, “boo”);
}

each insertion at location 0 has to move all the other elements to make room ...

approximately $1 + 2 + 3 + \ldots + 100 \approx 10,000$ moves,
an $O(n^2)$ algorithm
(We say: “quadratic complexity”)
The Java Collections framework

A set of “Collection” classes

**Lists** (ArrayList, LinkedList)

**Sets** (TreeSet, HashSet)

**Maps** (HashMap, TreeMap)

*and more ...*

Organized for easy use, with common interfaces like “iterable” and similar named methods