# CS159

Nathan Sprague

March 27, 2015

# Generics - Motivation

- Let's look back at the Abstract Classes Lab ↗
  - `LengthDatabase` - stores a collection of named lengths.
  - `MeasurementDatabase` - can store ANY TwoPartMeasure.
  - Which is better?

# Generics - Motivation

- Let's look back at the Abstract Classes Lab ↗
  - `LengthDatabase` - stores a collection of named lengths.
  - `MeasurementDatabase` - can store ANY TwoPartMeasure.
  - Which is better?
- Why not create an ObjectDatabase?

# Generics - Motivation

- Let's look back at the Abstract Classes Lab ↗
  - `LengthDatabase` - stores a collection of named lengths.
  - `MeasurementDatabase` - can store ANY TwoPartMeasure.
  - Which is better?
- Why not create an ObjectDatabase?
  - Error prone: we may accidentally store a mix of Lengths and Weights.
  - Inconvenient: we need to perform a cast to access any of the methods of the objects that are stored.

# ArrayList

- Syntax for ArrayList declaration:
  - `ArrayList<ReferenceType> myList;`
- Example of instantiation:
  - `myList = new ArrayList<ReferenceType>();`
- Notice:
  - ArrayList is a "generic type", may be declared to hold any reference type.
  - No size is provided.
  - ArrayList API↗
- Let's try it out...

# Looking Ahead - Collections

- ArrayList is one of many collections provided by the "Java Collections Framework".
- Collection - a class that stores multiple elements.

# Looking Ahead - Collections

- ArrayList is one of many collections provided by the "Java Collections Framework".
- Collection - a class that stores multiple elements.
- We will distinguish between:
  - The interface to a collection - how we interact with the collection.
  - The implementation of the collection - how the data is stored "behind the scenes".

- Java Collections Overview ↗
- Java Collections Interfaces Overview ↗

# ArrayList

Naming convention for Java Collection types: ArrayList

- Array - Coded using arrays "under the hood".
- List - Implements the List interface↗.

- ArrayList API↗

# Question

Do you see anything odd in this code?

```
1 ArrayList<Integer> nums = new ArrayList<Integer>();
2 nums.add(150);
3 nums.add(200);
4 System.out.println(nums.get(1).toString());
```

# Autoboxing

- What is going on here?
  - Java automatically converts primitive types to reference types when necessary.
  - `nums.add(150);`
  - silently becomes:
  - `nums.add(new Integer(150));`

# Copying ArrayLists

- What will happen when this code executes?

```
1          ArrayList<String> strings = new ArrayList<String>();
2
3          strings.add("Hello");
4          strings.add("There");
5
6          ArrayList<String> moreStrings;
7          moreStrings = strings;
8
9          moreStrings.remove(0);
10
11         System.out.println(strings.size());
```

# Copying ArrayLists

- How about now?

```
1          ArrayList<String> strings = new ArrayList<String>();
2
3          strings.add("Hello");
4          strings.add("There");
5
6          ArrayList<String> moreStrings;
7          moreStrings = new ArrayList<String>(strings);
8
9          moreStrings.remove(0);
10
11         System.out.println(strings.size());
```