Containers In Java

Prof. Chris Jermaine
cmj4@cs.rice.edu

One of the Cool Things About Java...

- Is all of the classes you get bundled with the JVM
- Provides functionality that you as a programmer can rely upon
  - — You know that if someone can run your code...
  - — Then they have that class available...
  - — And it will work on their platform”
  - — Of key importance when you are doing system-specific stuff, such as I/O!

Most Commonly Used Standard Classes

- Have to do with I/O
  - — Writing to the screen and to files, reading from the keyboard, etc.
- -OR- Have to do with organizing data
  - — “Container” classes that allow you to store/retrieve data

The Java “Collections” Framework

- Provides a set of generic classes for storing/retrieving data
  - “Generic” means they can store objects of almost any type
- All Collections implement one of a few key interfaces
  - List
  - Map
  - Stack
  - Queue

The Java "Collections" Framework

- Provides a set of generic classes for storing/retrieving data
  - "Generic" means they can store objects of almost any type
- What does "generic" mean in Java?
  - Will do this in much more detail later in the class
  - But will give you enough to survive on now
- In Java, a "generic" is a parameterized class
  - Takes a class name (or several) as an arg
  - In Java collections, this/these are typically the class(es) you are storing
- Ex: ArrayList:
  - ArrayList <Integer> foo = new ArrayList <Integer> ();
  - ArrayList <Double> foo = new ArrayList <Double> ();

What Types of Collections Are There?

- All Java Collections implement one of a few key interfaces
  - — List

    Allows you to store things in order, iterate through them (Ex: ArrayList)
  - — Map

    Allows you to store (key, value) pairs, and access a specific value for a given key
  - — Stack

    Allows you to add stuff on top of the stack (push), take it off of the top (pop)
  - — Queue

    Allows you to add stuff to the queue, take it out of the front
  - — Few others...

What Types of Collections Are There?

- But be aware, List, Map, Stack, Queue are *interfaces*
- To actually use, need to choose a Collections class that *implements* the interface
  - — Means that the class has all of the ops defined by the interface
  - — Will cover interfaces in detail in a few lectures...
- What are the standard Collections classes you'll use?
  - — ArrayList (wrapper for a simple array)
  - — HashMap (map implemented using hash table: very fast lookups, OK inserts)
  - — TreeMap (map implemented using a binary tree: fast lookups, fast inserts)
  - — PriorityQueue (queue that always has smallest/biggest item at front)

Want to Know More About Collections?

- Java documentation on the web is the way to go!
  - — Google search: “Java 6 Oracle XXX” is your first option
  - — Only go to resources such as StackOverflow once you’ve read the official docs

Two More Things to Cover

- Say you define a class Foo
- Want to store it in a priority queue
  - — PriorityQueue <Foo> myQueue = new PriorityQueue <Foo> ();
- Won’t necessarily work the way you want. Why?

Two More Things to Cover

- Say you define a class Foo
- Want to store it in a priority queue
  - PriorityQueue <Foo> myQueue = new PriorityQueue <Foo> ();
- Won't necessarily work the way you want. Why?
  - A priority queue poll () call always removes "smallest" item
  - But what does "smallest" mean for objects of type Foo?
  - Need to define this!
  - How?

Way 1: Make Sure Foo Extends Comparable

```
class Foo implements Comparable <Foo> {

  private int x;
  ...
  public int compareTo (Foo me) {
    return (x - me.x);
  }
  ...
}
```

Way 2: Supply a Comparator

- "Comparator <T>" is a standard Java interface
  - — This means it can compare objects of type T
- You create a class that implements it

```
class FooComparator implements Comparator <Foo> {
  public int compareTo (Foo me, Foo withMe) {
    return (me.getVal () - withMe.getVal());
  }
}
  ...

PriorityQueue <Foo> myQ =
  new PriorityQueue <Foo> (10, new FooComparator ());
```

- Question: which to use... Comparable or Comparator?

What Happen w/o Comparable/Comparator?

- Without this, behavior is arbitrary
  - — How does Java know when one Foo is larger/smaller than another?
  - — I am not even sure how Java will do it (pretty random!)
  - — In an ideal world, would not even compile without this

Last Thing to Cover: Iteration

- Can write succinct code to loop through all items in anything that implements the Iterable interface
- Ex: ArrayList

```
ArrayList myList <Foo> = new ArrayList <Foo> ();
...
// this code prints the contents of myList!
for (Foo f : myList) {
  f.Print ();
}
```

- Not always applicable, but nice when you can apply it