Programming Languages and Techniques (CIS120)

Lecture 27

March 22, 2013

Generics, Collections, and Iterators

Announcements

- HW08 is due *Monday* at 11:59:59pm

- *Midterm 2 is Friday, March 29th in class*
- *Covers material through last Wednesday's lecture*
  - Mutable state (in OCaml and Java)
  - Queues (in OCaml and Java)
  - Objects (in OCaml and Java)
  - ASM (in OCaml and Java)
  - Reactive programming (in OCaml)
  - Arrays in (Java)
  - Subtyping & Inheritance (in Java)

The Java Collections Library

A case study in subtyping and generics.

(Also very useful!)

Java Packages

- Java code can be organized into *packages* that provide namespace management.
  - Somewhat like OCaml's modules
  - Packages contain groups of related classes and interfaces.
  - Packages are organized hierarchically in a way that mimics the file system's directory structure.
- A .java file can *import* (parts of) packages that it needs access to:

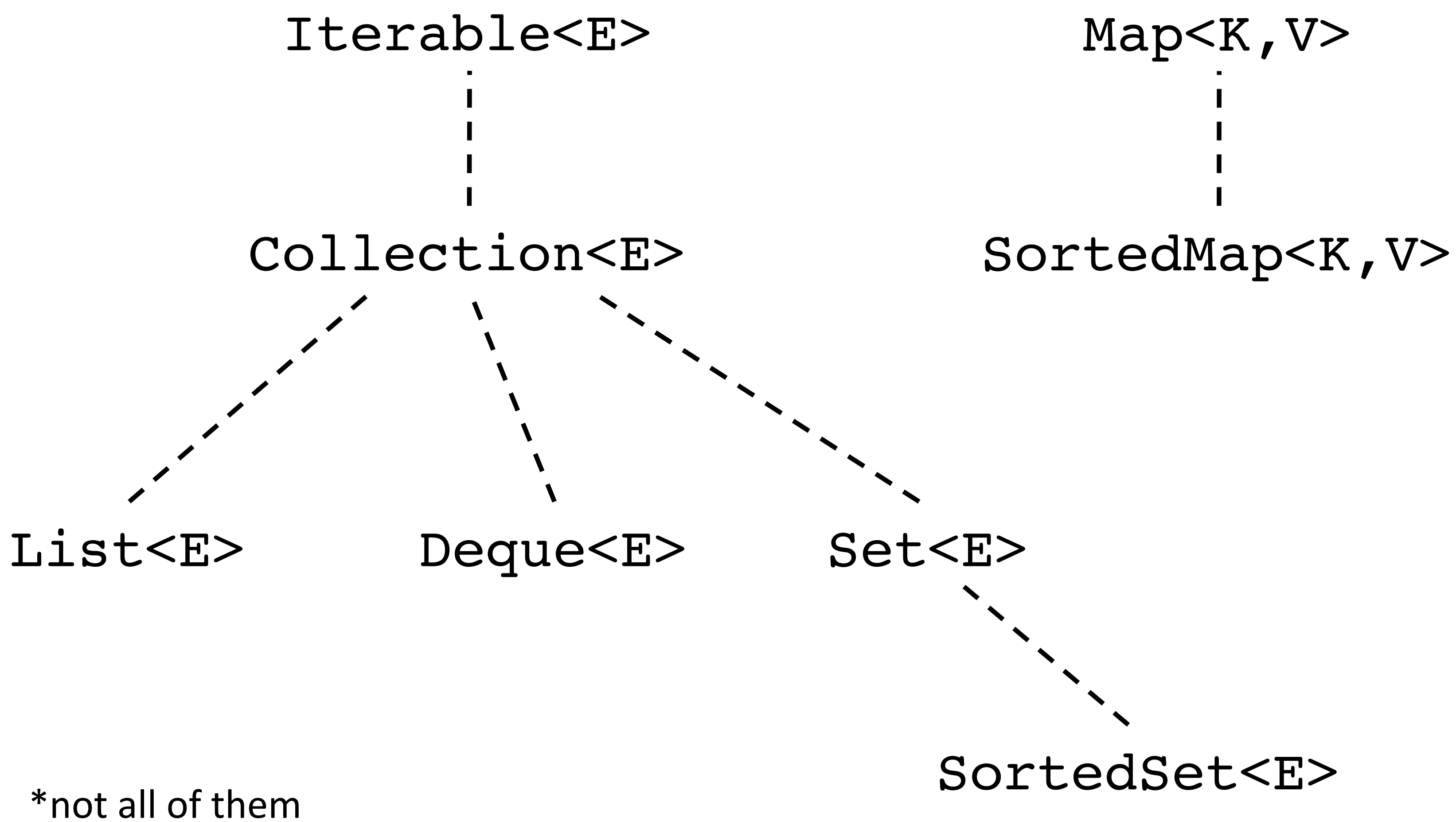
```
import org.junit.Test;     // just the JUnit Test class
import java.util.*;        // everything in java.util
```

- Important packages:
  - java.lang , java.io , java.util , java.math, org.junit
- See documentation at: http://download.oracle.com/javase/6/docs/api/index.html

Reading Java Docs

http://docs.oracle.com/javase/6/docs/api/java/util/package-summary.html

Interfaces* of the Collections Library



*not all of them

Collection<E> Interface (Excerpt)

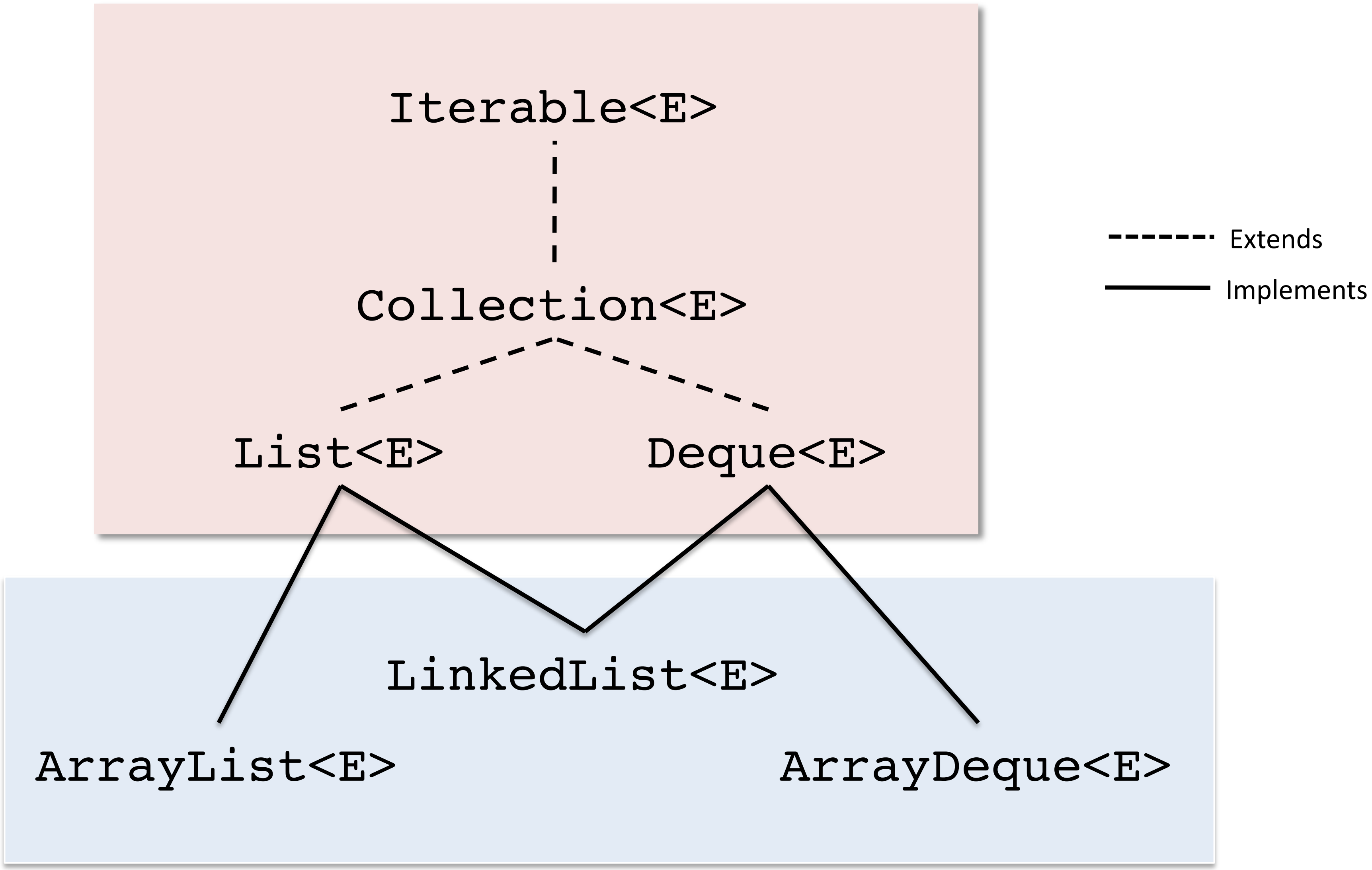
```
public interface Collection<E> extends Iterable<E> {
  // basic operations
  int size();
  boolean isEmpty();
  boolean add(E o);
  boolean remove(Object o);     // why not E?*
  boolean contains(Object o);

  // bulk operations
  …
}
```

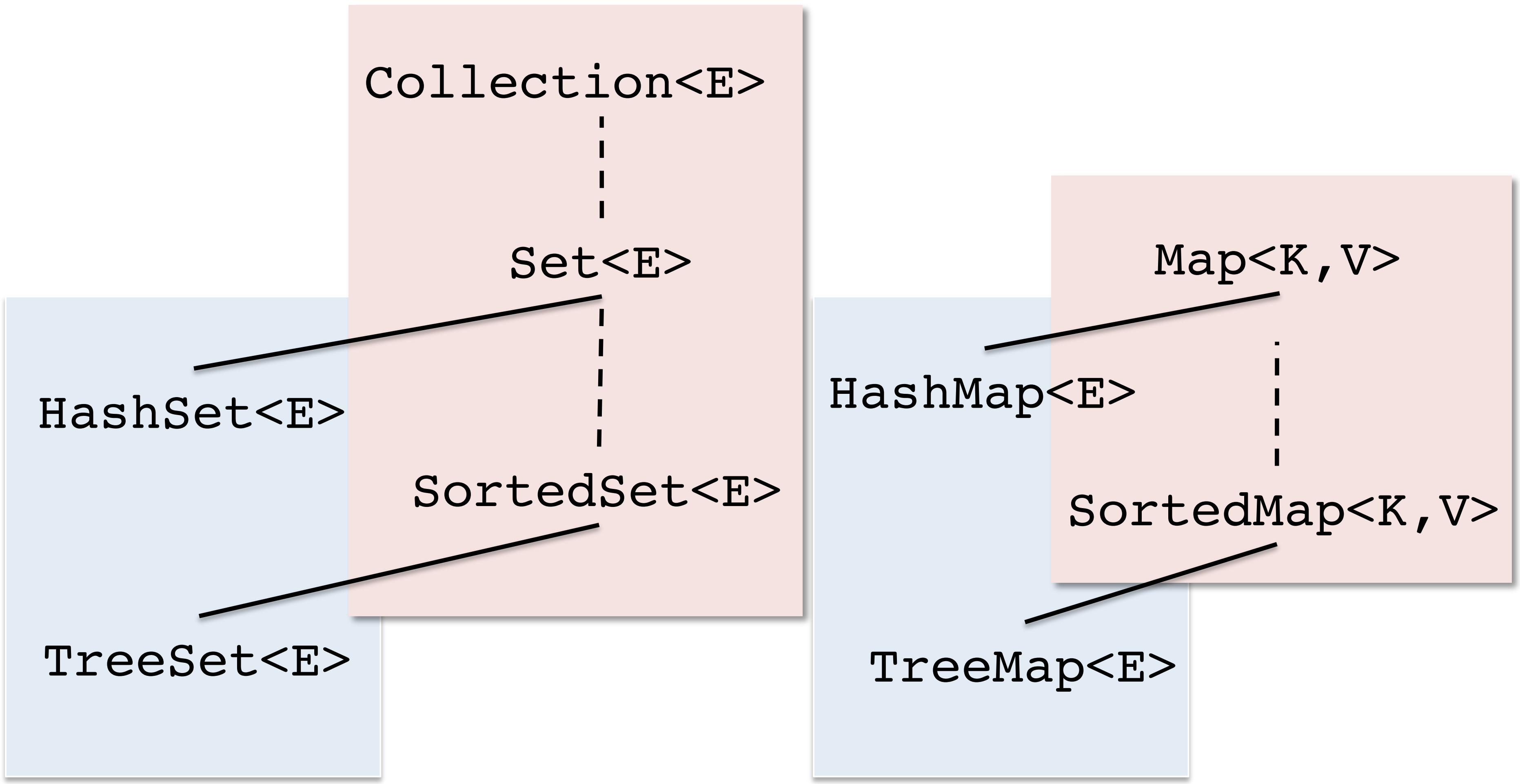
- We've already seen this interface in the OCaml part of the course.
- Most collections are designed to be *mutable* (like queues)

* Why not E? Internally, collections use the `equals` method to check for equality – membership is determined by o.equals, which does not have to be false for objects of different types. Most applications only store and remove one type of element in a collection, in which case this subtlety never becomes an issue.

Sequences


Iterable<E>
Collection<E>
List<E>
Deque<E>
LinkedList<E>
ArrayList<E>
ArrayDeque<E>
Extends
Implements

Sets and Maps*



*Read javadocs before instantiating these classes! There are some important details to be aware of to use them correctly.

Iterating over collections

iterators, while, for, for-each loops

Iterator and Iterable

```
interface Iterator<E> {
    public boolean hasNext();
    public E next();
    public void delete();   // optional
}
```

```
interface Iterable<E> {
    public Iterator<E> iterator();
}
```

Challenge: given a List<Book> how would you add each book's info to a catalogue using the iterator?

While Loops

syntax:

```
// repeat body until condition becomes false
while (condition) {
  body
}
```

statement

boolean *guard* expression

example:

```
List<Book> shelf = …  // create a list of Books

// iterate through the elements on the shelf
Iterator<Book> iter = shelf.iterator();
while (iter.hasNext()) {
    Book book = iter.next();
    catalogue.addInfo(book);
    numBooks = numbooks+1;
}
```

For Loops

syntax:

```
for (init-stmt; condition; next-stmt) {
  body
}
```

equivalent while loop:

```
init-stmt;
while (condition) {
  body
  next-stmt;
}
```

```
List<Book> shelf = …  // create a list of Books

// iterate through the elements on the shelf
for (Iterator<Book> iter = shelf.iterator();
     iter.hasNext();) {
   Book book = iter.next();
   catalogue.addInfo(book);
   numBooks = numbooks+1;
}
```

For-each Loops

syntax:

```
// repeat body for each element in collection
for (type var : coll) {
   body
}
```

element type

array or instance of Iterable<E>

example:

```
List<Book> shelf = …  // create a list of books

// iterate through the elements on a shelf
for (Book book : shelf) {
    catalogue.addInfo(book);
    numBooks = numbooks+1;
}
```

For-each Loops (Cont'd)

Another example:

```
int[] arr = …  // create an array of ints

// count the non-null elements of an array
for (int elt : arr) {
   if (elt != 0) cnt = cnt+1;
}
```

For-each can be used to iterate over arrays or any class that implements the `Iterable<E>` interface (notably `Collection<E>` and its subinterfaces).