# Programming Languages and Techniques (CIS120)

Lecture 27

Mar 23, 2012

## Generics, Collections and Iteration

# Announcements

- CIS Course Faire TODAY at 3PM, here
- HW08 is due on Monday, at 11:59:59pm
- Midterm 2 is Friday, Mar 30th
  - Location : FAGN AUD
- Midterm focus: 2/10 through today
- Review session, Wed 8-10PM, Levine 101

# Exam 2 Topics

- Mutable data structures
  - Linked data structures (queues, deques and Rooms)
  - Resizable arrays
  - Abstract Stack Machine
  - Reference equality (==) vs. structural equality (=)  (and == vs. .equals)
- Objects
  - Object encoding in OCaml
  - Objects in Java (dynamic dispatch, constructors, inheritance)
  - Encapsulation, aliasing
- Reactive & higher-order programming
  - Using first-class functions for event handlers
  - Transformer interface from HW07
- Array programming in Java
  - 1D and 2D arrays
  - Static methods
- Types
  - Options, mutable records in OCaml
  - Interfaces, Subtyping & Classes in Java

# Polymorphism

# Subtype Polymorphism

```
public interface ObjQueue {
  public void enq(Object o);
  public Object deq();
  public boolean isEmpty();
  public boolean contains(Object o);
  …
}
```

```
ObjQueue q = …;

q.enq("CIS 120");
___A___  x = q.deq();                    // What type for A?
System.out.println(x.toLowerCase()); // Does this work?
q.enq(new Point(0.0,0.0));
___B___  y = q.deq();
```

# Generics (Parametric Polymorphism)

```
public interface Queue<E> {
  public void enq(E o);
  public E deq();
  public boolean isEmpty();
  public boolean contains(E o);
  …
}
```

```
Queue<String> q = …;

q.enq("CIS 120");
___A___  x = q.deq();                    // What type for A?
System.out.println(x.toLowerCase()); // Does this work?
q.enq(new Point(0.0,0.0));               // What about this?
___B___  y = q.deq();
```

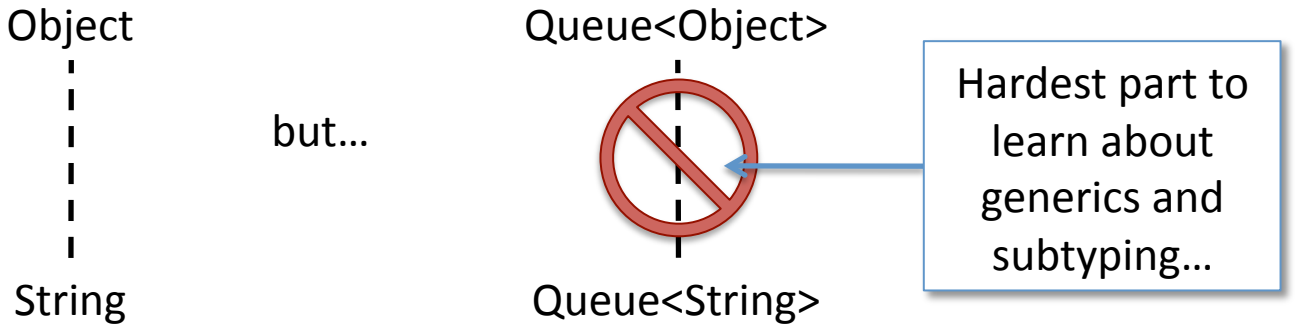
# Subtyping and Generics

# Subtyping and Generics

```
Queue<String> qs = new QueueImpl<String>();     // OK
Queue<Object> qo = qs;                          // OK?

qo.enq(new Object());
String s = qs.deq();                        // oops!
```

- Java generics are *invariant*:
  - Subtyping of *arguments* to generic types does not imply subtyping between the instantiations:



# The Java Collections Framework

A case study in subtyping and generics.

(Also very useful!)

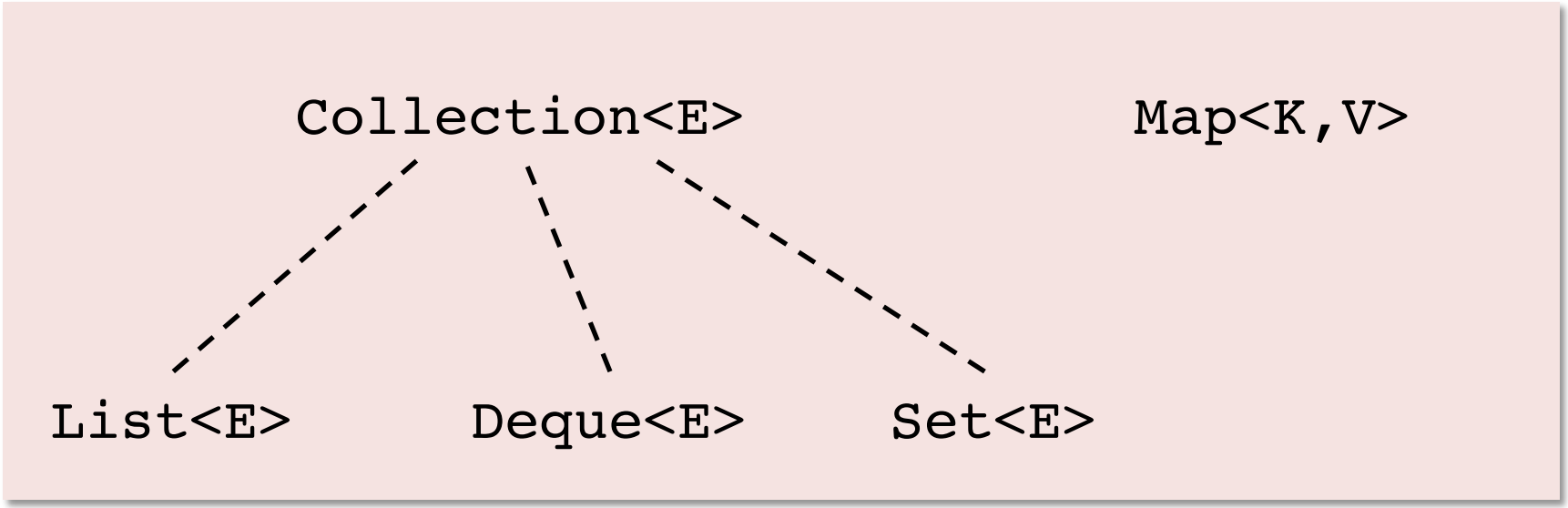
# Java Packages

- 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

# Interfaces* of the Collections Libary




```
import java.util.Collection;
import java.util.List;
```

*not all of them

# Collection<E> Interface (Excerpt)

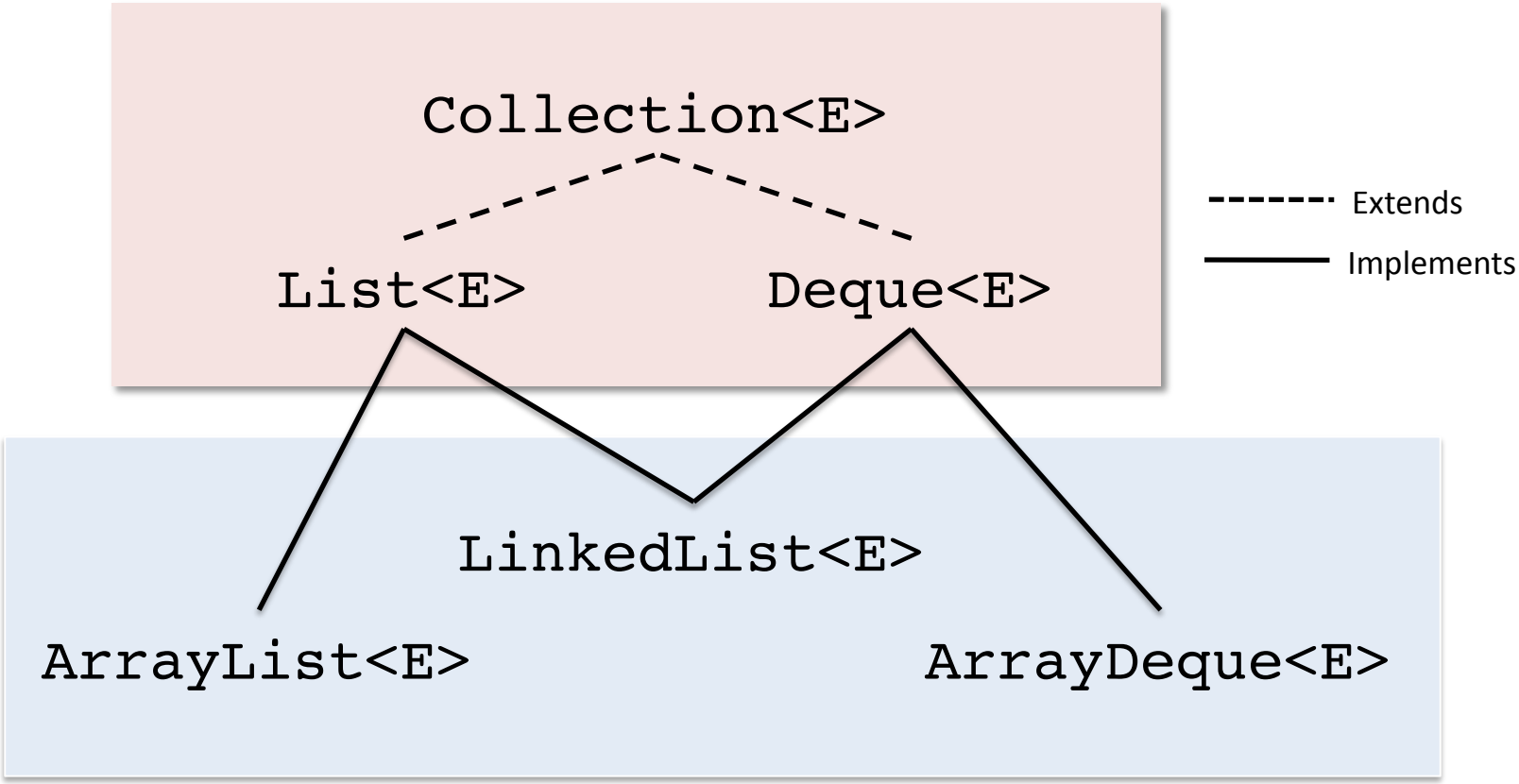
```
public interface Collection<E> extends Iterable<E> {
  // basic operations
  int size();
  boolean isEmpty();
  boolean add(E o);
  boolean remove(Object o);      // why not E?*
  boolean contains(Object o);

  // bulk operations
  …
}
```

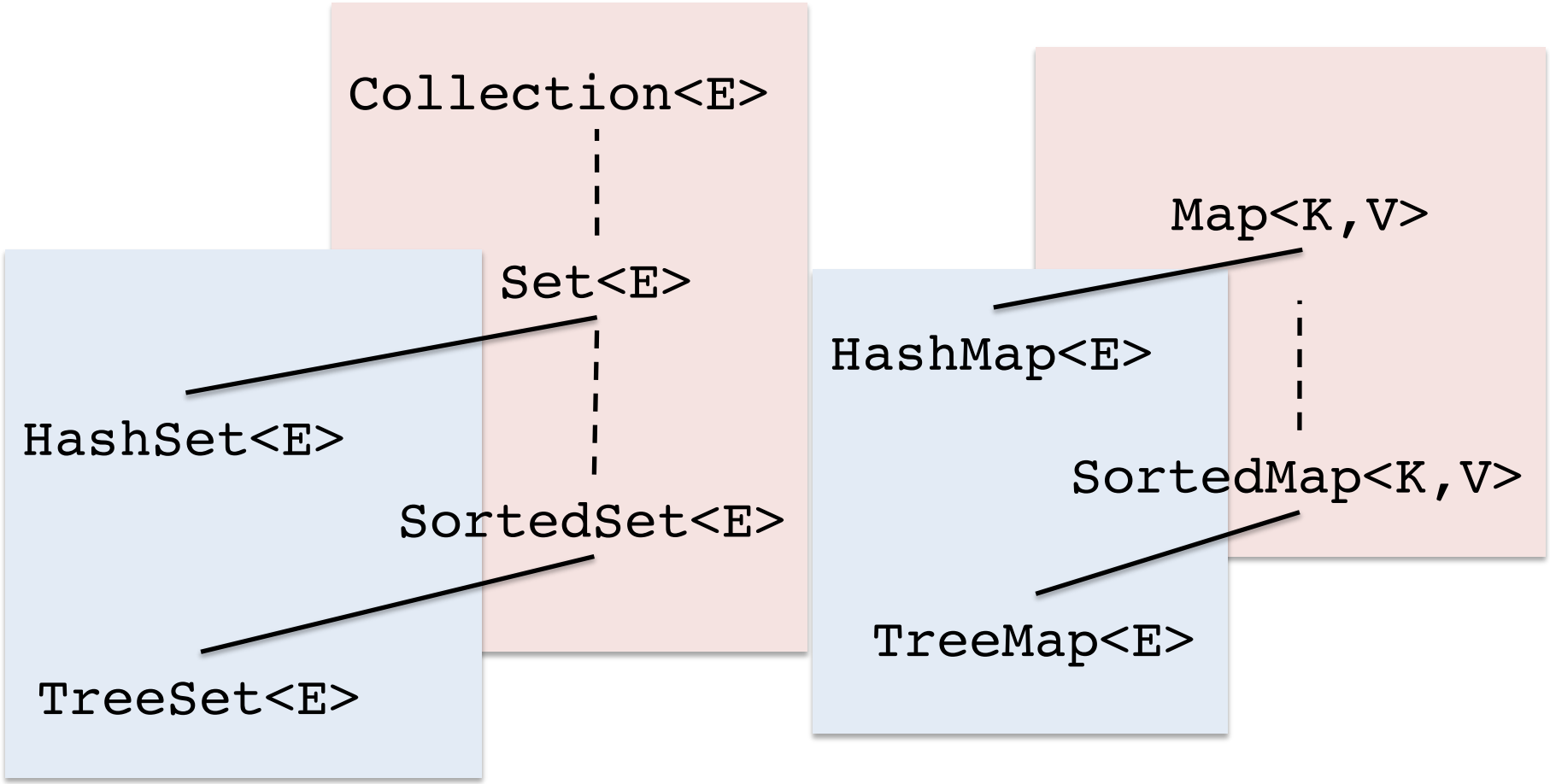
- We've already seen this interface in the OCaml part of the course.
- Most collections are designed to be *mutable* (like queues)

* 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.

# Sequence Implementations



# Sets and Map Implementations



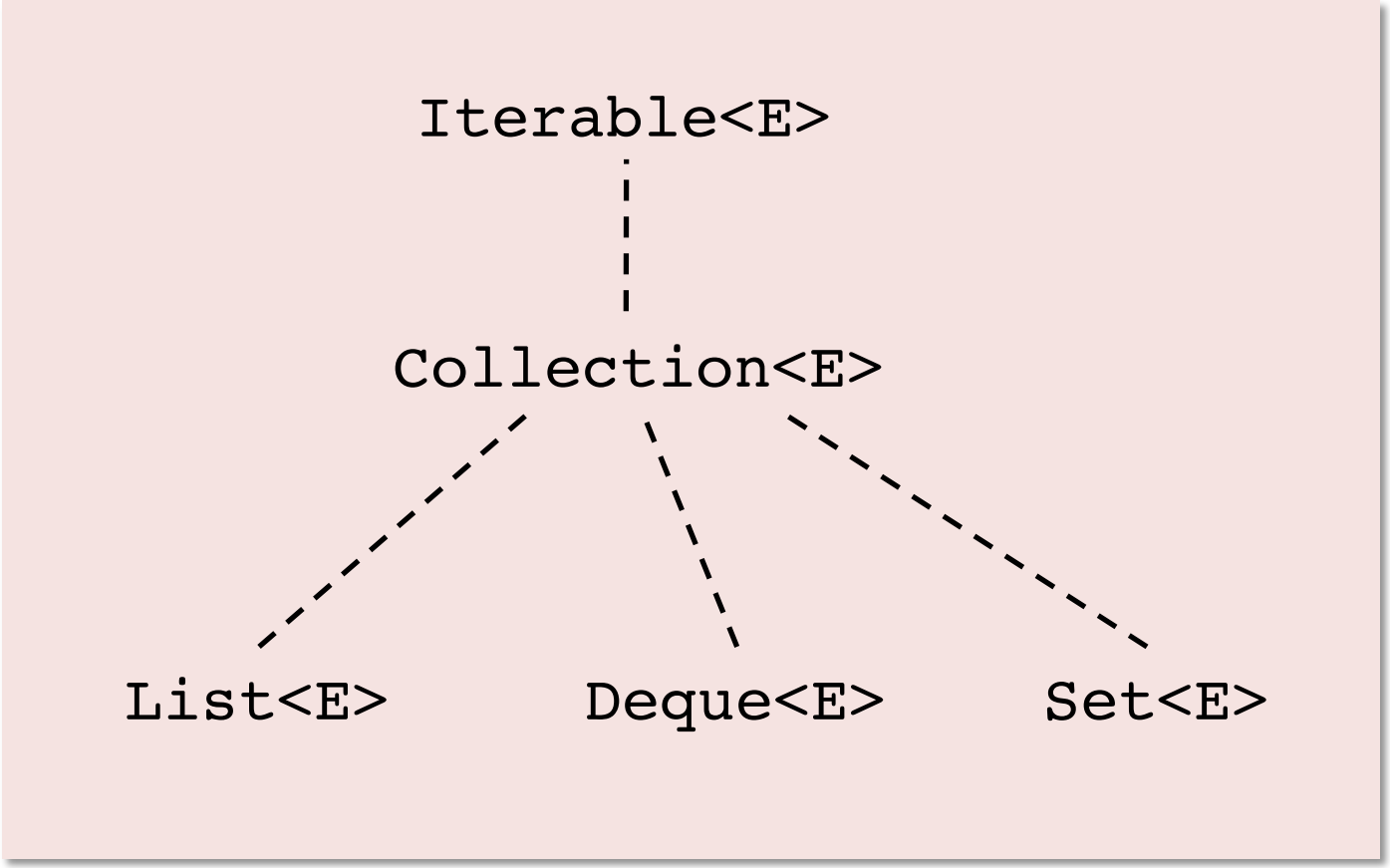

Extends

Implements

# Iterating over collections

iterators, while, for, for-each loops

# Interfaces of the Collections Libary



# Iterator and Iterable

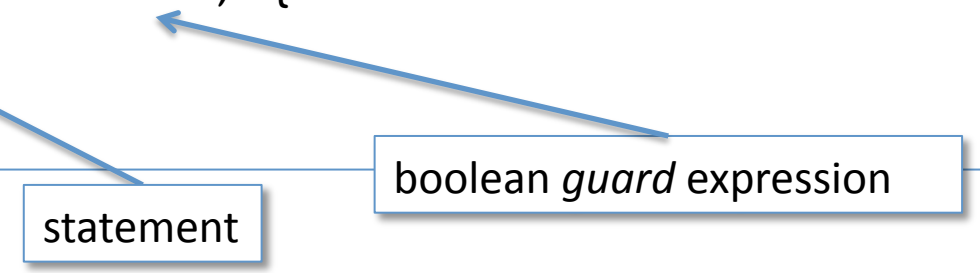
```
interface Iterable<E> {
    public Iterator<E> iterator();
}
```

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

# While Loops

syntax:

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




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-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).

# Reading Java Docs

1. Collection<E>
2. List<E> and Set<E>
3. Iterable<E> and Iterator<E>