# Java Collections Overview



## The Collections Framework

- Started with Java 1.2
- Numerous classes for common data structures
- Consistent interfaces
- Common algorithms
- Iterators
- All are in package java.util
- Convenient, interoperable
- Conversion to/from arrays
- Easily extendable



## Major Interfaces

- Collection
- List
  - LinkedList, ArrayList implementations
- Map
  - HashMap, TreeMap implementations
- Set
  - HastSet, TreeSet implementations



## Interface *Collection*

- All lists and sets are subtypes
- Interface methods: add, clear, contains, get, remove, set, size, toArray
- All collections store and return Objects
  - Must cast to specific actual type before using
  - Can't store elementary values (ints, chars, etc.) without wrapping (Integer, Character, etc.)
  - Unlike arrays, where the contents type is declared
  - Fix coming in Java 1.5.

## Lists

- Sequential access to data
  - elements have an integer index
- Interface List
- Abstract classe AbstractList
- Concrete classes ArrayList, LinkedList



## Sets

- Duplicates automatically eliminated (.equals)
- Subtype (interface) SortedSet maintains an order
- Concrete implementations: HashSet, TreeSet



## equals

- Many collections methods depend on *equals*
  - duplicate checking, containment checking, etc.
- Objects stored in collection need a proper *equals*
  - reflexive
  - symmetric
  - transitive



## compareTo

- Many situations depend on a proper compareTo method
- Signaled by Comparable interface
- Should be
  - reflexive
  - transitive
  - anti-symmetric

## Iterators

- iterator: an object that identifies a position within a collection
- All collection classes support iterators
  - List iterators: will follow index order
  - Other iterators: either no order guaranteed, or class-dependent
- Interface Iterator
  - Concrete inner classes usually not visible to user



## Maps

- Map: association between key and value
- Main operations
  - put (key, value)
  - get(key) returns value
- Maps per se do not implement the Collections interface
- Can get Collections (Set) of the keys and values separately



## Problem-Solving with Collections

- "Unique" -- think sets
- "Properties" -- think maps
- "Order" -- think Comparable and sorting



## Generic Algorithms

- Class *Collections*
  - not to be confused with *Collection*
  - handy static methods
- *Collections.sort(List)*
- *Collections.binarySearch(List)*
- *Collections.copy ...*

## Interoperability

- Via common methods of the interfaces
- Via addAll method
  - *mycollectionobject.addAll(existingCollection)*
- Via constructor
  - *List myList = new ArrayList(existingCollection)*



## Interoperability Advice

- Advice: use the most general type possible
- Example: instead of

*ArrayList myList = new ArrayList( );*
consider *List myList = new ArrayList( );*
or even *Collection myList = new ArrayList( );*

- Example: instead of

*LinkedList myOperation(HashSet s);*
consider *Collection myOperation(Collection s);*
not always possible



## Arrays

- Class Arrays has handy methods
- .sort, etc
- Interoperating between arrays and collections



## Wrapped Collections (Views)

- Unmodifiable (Read-only)
  - Protects the collection structure, not the object contents
  - Created by Collections factory methods

  example:

  *Set myReadOnlySet =*
  *Collections.unmodifiableSet(mySet);*
- Synchronized
  - Safe simultaneous access to an object
  - Needed for multi-thread programming

## Generics

- Coming in Java 1.5
- Types as parameters
- Can specify the types of the objects stored in the collections
- Greater type-safety
- Eliminates annoying casts



## Summary

- Java 1.2 and above has numerous useful collections facilities
- Great programming convenience
- Get familiar with them!