Introduction to Collections
Logistics

• Will post Homework 9 tonight
  – Due on Tuesday

• No class next Wednesday
  – Eat some turkey!
Collections

• A **collection** is a structured group of objects
• Java 1.2 introduced the Collections Framework
  – Collections are defined in **java.util**
  – The Collections framework is mostly about interfaces
  – There are a number of predefined implementations
• Java 5 introduced generics and “genericized” all the existing collections
  – **Vectors** have been redefined to implement **Collection**
  – Trees, linked lists, stacks, hash tables, and other classes are implementations of **Collection**
  – Arrays do not implement the **Collection** interfaces
Types of Collection

• Java supplies several types of **Collection**:  
  – **Set**: cannot contain duplicate elements, order is not important  
  – **SortedSet**: like a **Set**, but order is important  
  – **List**: may contain duplicate elements, order is important  

• Java also supplies some “collection-like” things:  
  – **Map**: a “dictionary” that associates keys with values, order is not important  
  – **SortedMap**: like a **Map**, but order is important
Collection Properties

• Here’s the good news about collections:
  – They are elegant: they combine maximum power with maximum simplicity
  – They are uniform: when you know how to use one, you almost know how to use them all
  – You can easily convert from one to another

• And the bad news:
  – Because there is no special syntax for them (as there is for lists, sets, and dictionaries in Python), you have to work with them using object notation
The Collection interface

- Much of the elegance of the Collections Framework arises from the intelligent use of interfaces
- The `Collection` interface specifies (among many other operations):
  - `boolean add(E o)`
  - `boolean contains(Object o)`
  - `boolean remove(Object o)`
  - `boolean isEmpty()`
  - `int size()`
  - `Object[] toArray()`
  - `Iterator<E> iterator()`
The List interface

• A list is an ordered sequence of elements
  – Similar to Python lists
• interface List<E> extends Collection, Iterable
  – Every list is declared with a specific type E – more about generics later
• Some important List methods are:
  – void add(int index, E element)
  – int size()
  – E remove(int index)
  – boolean remove(Object o)
  – E set(int index, E element)
  – E get(int index)
  – int indexOf(Object o)
  – int lastIndexOf(Object o)
Creating a list

• You cannot directly create a **List** – it’s an interface

• There are two common types of **List**
  – **ArrayList**
  – **LinkedList**
  – Both of these BEHAVE the same way, but are implemented differently!
  – In general, **ArrayList** is much faster at accessing whereas **LinkedList** is faster at adding/removing

• Syntax
  ```java
  List<E> = new ArrayList<E>();
  ```
  – If *E* is an **Object**, use the object name (e.g., **PezDispenser**)
  – For **int**, use **Integer**; for **double** use **Double**, etc.
  – Need to import **java.util.ArrayList**

• Eclipse example: ListExamples.java
The set interface

• A set is a collection in which:
  – There are no duplicate elements (according to equals), and
  – Order is not important

• interface Set<E> implements Collection, Iterable

• The methods of Set are exactly the ones in Collection

• The following methods are especially interesting:
  – boolean contains(Object o) // membership test
  – boolean containsAll(Collection<?> c) // subset test
  – boolean addAll(Collection<? extends E> c) // union
  – boolean retainAll(Collection<?> c) // intersection
  – boolean removeAll(Collection<?> c) // difference

• addAll, retainAll, and removeAll return true if the receiving set is changed, and false otherwise

• Eclipse example: SetExample.java
The SortedSet interface

• A **SortedSet** is a **Set** for which the order of elements *is* important

  interface SortedSet\<E\> implements Set, Collection, Iterable

• Two of the **SortedSet** methods are:
  – E first()
  – E last()

• More interestingly, only **Comparable** elements can be added to a **SortedSet**
  – The **Comparable** interface is covered in a separate lecture
The Map interface

• A map is a data structure for associating keys and values
  – Interface Map<K,V>

• The two most important methods are:
  – V put(K key, V value) // adds a key-value pair to the map
  – V get(Object key) // given a key, looks up the associated value

• Some other important methods are:
  – Set<K> keySet()
    – Returns a set view of the keys contained in this map.
  – Collection<V> values()
    – Returns a collection view of the values contained in this map
Map implementation

• The most popular implementation is a **HashMap**
  – Similar to Python’s dictionary
• Will talk more later about the implementation of a hash map
• Eclipse example: HashMapExample.java
Iterating through collections

• Java has a special for loop for iterating through collections
  – Very similar to Python

• Syntax
  
  ```java
  for (E item : collection) { ... }
  ```

• Implicitly uses an iterator and works because collections are Iterable
  – Will talk about Iterable/Comparable later in more detail

• Eclipse example: SetExample.java