Objectives

- Collections
- Jar Files
- Compiled vs Interpreted

Review

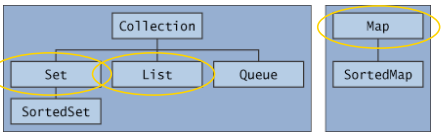
- Quickly: what are the different types of streams?
- How can we use enumerated types?
- What is the Java Collection Framework made up of?

Review: Collections Framework

- **Interfaces**
  - Abstract data types that represent collections
  - Collections can be manipulated *independently* of implementation
- **Implementations**
  - Concrete implementations of the collection interfaces
  - Reusable data structures
- **Algorithms**
  - Methods perform useful computations on collections, e.g., searching and sorting
  - Polymorphic: same method can be used on many different implementations of collection interface
  - Reusable functionality

Core Collection Interfaces

- Encapsulate different types of collections



`List` Interface

- An ordered collection of elements
- Can contain duplicate elements
- Has control over where objects are stored in the list
- `boolean add(<E> o)`
  - Boolean so that List can refuse some elements
    - e.g., refuse adding `null` elements
- `<E> get(int index)`
  - Returns element at the position index
- `int size()`
  - Returns the number of elements in the list
- And more! (`contains`, `remove`, `toArray`, …)

Differences from Python

- No shorthand
  - `list[pos]`

List Implementations

- ArrayList
  - Resizable array
  - Used most frequently
  - Fast
- LinkedList
  - Use if adding elements to beginning of list
  - Use if often delete from middle of list

cards.Deck.java

Implementation vs. Interface

- Implementation choice affects only performance
- Preferred Style:
  - Choose an implementation
  - Assign collection to variable of corresponding **interface** type or pass collection to method expecting argument of interface type
- Why?
  - Program does not depend on methods in a given implementation
  - Programmer can change implementations
    - Performance concerns or behavioral details

Generics Aside

- Can only contain Object types, not primitive types
  - Autoboxing and Autounboxing to the rescue!
  - Example: If collecting ints, use Integer

Set Interface

- No duplicate elements
  - Needs to determine if two elements are "logically" the same (equals method)
- Models mathematical set abstraction
- boolean add(<E> o)
  - Add to set, only if not already present
- int size()
  - Returns the number of elements in the list
- And more! (contains, remove, toArray, …)
  - Note: no get method -- get #3 from the set?

Set Implementations

- HashSet
  - Implements set using hash table
    - add, remove, and contains each execute in O(1) time
  - Used more frequently
  - Faster than TreeSet
  - No ordering
- TreeSet
  - Implements set using a tree
    - add, remove, and contains each execute in O(log n) time
  - Sorts

FindDuplicates Problem

- From the array of command-line arguments, identify the duplicates

FindDuplicates

```
public static void main(String args[]) {
    Set<String> s = new HashSet<String>();
    for (String a : args) {
      if (!s.add(a)) {
         System.out.println(
            "Duplicate detected: " + a);
      }
    }
    System.out.println(s.size() +
       " distinct words detected: " + s);
}
```

Note how much code changes if s is a TreeSet

Map Interface

- Maps keys (of type <K>) to values (of type <V>)
- No duplicate keys
  - Each key maps to at most one value
- `<V> put(<K> key, <V> value)`
  - Returns old value that key mapped to
- `<V> get(Object key)`
  - Returns value at that key
- `Set<K> keySet()`
  - Returns the set of keys

Map Implementations

- HashMap
  - Fast
- TreeMap
  - Sorting
  - Key-ordered iteration
- LinkedHashMap
  - Fast
  - Insertion-order iteration
  - Remove stale mappings --> custom caching

Declaring Maps

- Declare types for both keys and values
- `Class HashMap<K,V>`

```
Map<String, List<String>> map
    = new HashMap<String, List<String>>();
```

Keys are Strings
Values are Lists of Strings

Rethinking PetSurvey.java

- How did we keep track of a pet's votes in PetSurvey.java?
- Could we do better?

PetSurvey.java

4 Week Checklist

- Primitive types
- Static typing
- Java library classes: String, Object, Math, Arrays, …
- Object-oriented concepts
  - Encapsulation, inheritance, polymorphism, abstract classes, interfaces
- Static methods, fields
- Javadocs
- I/O Streams
- Collections
- Jar files
- Your job: representing data, leverage classes, *ilities

For Next Week

- Assignment 8: Due Monday
  - Practice with Collections
- Wednesday: Midterm
  - Come with your questions on Monday