

## Objectives

- Collections Framework
  - Maps
  - Algorithms
  - Traversing
- Enumerated Types
- Comparators

Oct 10, 2011

Sprenkle - CSCI209

1

## Review

- What are the components of the Java Collection Framework?
- What collection interfaces and implementations did we discuss?
- Why do we use interface object variables instead of implementations in our programs?
- How do we declare/initialize a new Collection object?

Oct 10, 2011

Sprenkle - CSCI209

2

## 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

Oct 10, 2011

Sprenkle - CSCI209

3

## MAPS

Oct 10, 2011

Sprenkle - CSCI209

4

## Maps

- Maps keys (of type  $\langle K \rangle$ ) to values (of type  $\langle V \rangle$ )
- No duplicate keys
  - Each key maps to at most one value

Oct 10, 2011

Sprenkle - CSCI209

5

## Map Interface

- $\langle V \rangle$  put( $\langle K \rangle$  key,  $\langle V \rangle$  value)
  - Returns old value that key mapped to
- $\langle V \rangle$  get(Object key)
  - Returns value at that key (or null if no mapping)
- Set $\langle K \rangle$  keySet()
  - Returns the set of keys

And more ...

Oct 10, 2011

Sprenkle - CSCI209

6

## A few Map Implementations

- **HashMap**
  - Fast
- **TreeMap**
  - Sorting
  - Key-ordered iteration
- **LinkedHashMap**
  - Fast
  - Insertion-order iteration

Oct 10, 2011

Sprenkle - CSCI209

7

## Declaring Maps

- Declare types for both keys and values

● `class HashMap<K,V>`

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

Keys are Strings

Values are Integers

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

Keys are Strings

Values are Lists of Strings

Oct 10, 2011

Sprenkle - CSCI209

8

## Pet Survey

- What is the best way to keep track of people's votes for their favorite pet—dog, cat, bird, fish, snake, other?
- What are the options? Tradeoffs of the options?

Implement: `castVote`, `validVote`, `getAnimals`

`PetSurvey3.java`

Oct 10, 2011

Sprenkle - CSCI209

9

## ALGORITHMS

Oct 10, 2011

Sprenkle - CSCI209

10

## Collections Framework's Algorithms

- *Polymorphic algorithms*
- Reusable functionality
- Implemented in the `Collections` class
  - Static methods, 1<sup>st</sup> argument is the collection
  - Similar to `Arrays` class, which operates on arrays

Oct 10, 2011

Sprenkle - CSCI209

11

## Overview of Available Algorithms

- **Sorting** – optional `Comparator`
- **Shuffling**
- **Searching** – `binarySearch`
- **Routine data manipulation**: `reverse*`, `copy*`, `fill*`, `swap*`, `addAll`
- **Composition** – frequency, disjoint
- **Finding min, max**

\* Only Lists

Oct 10, 2011

Sprenkle - CSCI209

12

## TRAVERSING COLLECTIONS

Oct 10, 2011

Sprenkle - CSCI209

13

## Two Ways to Iterate over Collections

- For-each loop
- Iterator

Oct 10, 2011

Sprenkle - CSCI209

14

## Traversing Collections: For-each Loop

- For-each loop:

```
for (Object o : collection)
    System.out.println(o);
```

- Valid for all Collections

- Maps (and its subclasses) are not Collections
- But, Map's keySet() is a Set and values() is a Collection

Oct 10, 2011

Sprenkle - CSCI209

15

## Traversing Collections: Iterator

- Iterator: Java Interface
- To get an Iterator from a Collection object:

```
Iterator<E> iterator()
```

- Returns an Iterator over the elements in this collection
- Example:

```
Iterator<String> iter = keys.iterator();
```

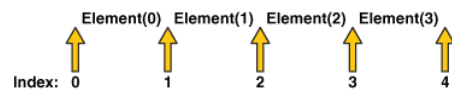
Oct 10, 2011

Sprenkle - CSCI209

16

## Iterator: Like a Cursor

- Always between two elements



```
Iterator<Integer> i = list.iterator();
while( i.hasNext()) {
    int value = i.next();
    ...
}
```

Oct 10, 2011

Sprenkle - CSCI209

17

## Iterator API

- <E> next()
  - Get the next element
- boolean hasNext()
  - Are there more elements?
- void remove()
  - Remove the previous element
  - Only safe way to remove elements during iteration
    - Not known what will happen if remove elements in for-each loop

Oct 10, 2011

Sprenkle - CSCI209

18

## Polymorphic Filter Algorithm

```
static void filter(Collection c) {
    Iterator i = c.iterator();
    while( i.hasNext() ) {
        // if the next element does not
        // adhere to the condition, remove it
        if ( ! condition(i.next()) ) {
            i.remove();
        }
    }
}
```

Polymorphic: works regardless of Collection implementation

Oct 10, 2011

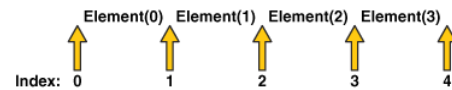
Sprenkle - CSCI209

19

## Traversing Lists: ListIterator

- Methods to traverse list backwards too
  - `hasPrevious()`
  - `previous()`
- To get a `ListIterator`:
  - `listIterator(int position)`
    - Pass in `size()` as position to get at end of list

Key difference



Oct 10, 2011

Sprenkle - CSCI209

20

## Enumeration

- Legacy class
- Similar to Iterator
- Example methods:
  - `boolean hasMoreElements()`
  - `Object nextElement()`
- Longer method names
- Doesn't have remove operation

Oct 10, 2011

Sprenkle - CSCI209

21

## How Not to Iterate

- Don't use `get` to access List
  - If implementation a `LinkedList`, performance is reeeeeally slow

```
for (int i = 0; i < list.size(); i++) {
    count += list.get(i); // do something
}
```

Oct 10, 2011

Sprenkle - CSCI209

22

## Synchronized Collection Classes

- For multiple threads sharing same collection
- Slow down typical programs
  - Avoid for now
- e.g., `Vector`, `Hashtable`
- See `java.util.concurrent`

Another example: `StringBuffer` is synchronized, whereas `StringBuilder` is not

Oct 10, 2011

Sprenkle - CSCI209

23

## Benefits of Collections Framework

Oct 10, 2011

Sprenkle - CSCI209

24

## Benefits of Collections Framework

- **Provides common, well-known interface**
  - Allows interoperability among unrelated APIs
  - Reduces effort to learn and to use new APIs for different implementations
- **Reduces programming effort:** provides useful, reusable data structures and algorithms
- **Increases program speed and quality:** provides high-performance, high-quality implementations of data structures and algorithms; interchangeable implementations → tuning
- **Reduces effort to design new APIs:** use standard collection interface for your collection
- **Fosters software reuse:** New data structures/algorithms that conform to the standard collection interfaces are reusable

Oct 10, 2011

Sprenkle - CSCI209

25

## TODO

- Assignment 8: Due NEXT Wednesday
  - Practice with Generics, Collections, User interface

Oct 10, 2011

Sprenkle - CSCI209

26