CS 2334: Lab 5
Maps, Sets and Lists

Collections, Maps, Sets and Lists in Java

- The abstract concepts of collections, maps, sets and lists are (or should be) easy to understand
- But:
  - There are many different ways to implement these concepts programmatically
  - Different approaches have different properties, including the amount of computational time or memory required to represent and operate on the collections

Implementing Lists: Some Tradeoffs

- Array
  - Access: fast (constant time)
  - Insertion: slow on average
- Linked List
  - Access: slow on average
  - Insertion: fast (constant time, after access)
- Tree
  - Access: medium on average
  - Insertion: medium on average

Note: more coming in your Data Structures class

Maps

Maps are an important component of large database systems

- Maps allow for fast access and insertion of data
- Keys & Values
  - Values are the stored data
  - Keys are mapped to values
- Each key uniquely maps to one value
  - Keys therefore form a proper “set”

Non-Java Example

- Suppose *a*:*b* means that “*a* maps to *b*”
- An example map M:
  - M = {“a”:1, “b”:3, “c”:2}
- Example accesses:
  - M[“a”] returns 1
  - M[“b”] returns 3
- Example modifications:
  - M[“c”] = 4
  - M[“d”] = 7
  - M = {“a”:1, “b”:3, “c”:4, “d”:7}

Java Maps

- *Map* is an interface
  - put(Object key, Object value)
  - get(Object key)
- Keys form a set
  - keySet()
  - Implications?
- Values form a collection
  - values()
  - Implications?
- For full details, consult the Java API and your book

Map Implementations

- HashMap
  - Fast access (constant time)
  - (key, value) pairs are not ordered in any meaningful way
  - Uses a hash function
    - Converts keys into indices for an internal array
- TreeMap
  - (key, value) pairs are stored in a tree
  - Ordering of pairs determined by the natural order of the keys or by a Comparator
  - Slower access time

Generic Maps

- HashMap<T,E> foo;
  - Specifies that foo accepts keys of type T and values of type E
- Example:

```
HashMap<String,Integer> map;
map = new HashMap<String,Integer>();
map.put(“a”,1); map.put(“b”,3); map.put(“c”,2);
// map contains {“a”:1, “b”:3, “c”:2}
map.get(“a”);  // returns 1
map.put(“c”,4);
map.put(“d”,7);
//map contains {“a”:1, “b”:3, “c”:4, “d”:7}
```

Map Example Continued…

```
// Number of entries in a map:
int num = map.size();

// A set that contains all the keys
Set<String> set = map.keySet();

// A collection that contains all the values
Collection<Integer> c = map.values();

// A set of (key, value) entries
 Set<Map.Entry<String,Integer>> pairs =
  map.entrySet();
```

General “To Do”

- Download Lab5.zip class web site
  - Also available: these slides + relevant book sections
- Milestone 1: How are entries organized in different Map implementations?
- Milestone 2: What is the access performance of the different Map implementations?
- Milestone 3: Creating a Maps with Different Key Variables

You should complete the assignment and demo by the end of the lab session