CompSci 201, L5: Sets and Maps
Logistics, Coming up

• Today, Monday 9/12
  • Project 0: Person201 due

• This Wednesday, 9/14
  • APT2 due
  • Hashing

• This Friday 9/16
  • Discussion

• Next Monday, 9/19
  • Project 1: NBody due
  • Runtime efficiency
Go to duke.is/v3mgn

Not graded for correctness, just participation.

Try to answer *without* looking back at slides and notes.

But do talk to your neighbors!
Array vs. Collection

**Array**
- Stores primitives (int, char, etc.) or objects (String, etc.)
- Does not print “nicely” to terminal, print one at a time.
- API static utility methods in `java.util.Arrays`

**Collection**
- Stores objects only, use wrapper (e.g., Integer) for primitives
- Prints ”nicely” to the terminal.
- API static utility methods in `java.util.Collections`
ArrayList is basically an Array of objects, easy to convert with API methods.

```java
ArrayList<String> strList = new ArrayList<>();
String[] strArray = {"CS", "201", "is", "the", "best"};

// Convert a String Array to a List using
// the static Arrays.asList method and the
// ArrayList addAll method
strList.addAll((Arrays.asList(strArray)));

// Convert a String List to a String Array the
// ArrayList toArray() method and casting
String[] newStrArray = strList.toArray(new String[0]);
```
ArrayList <-> Array Conversion, Primitive Types

Primitive types more “manual”, remember Lists only use Object types (int vs. Integer)

```java
ArrayList<Integer> intList = new ArrayList<>();
int[] intArray = {2, 0, 1};

// Convert a int (or other primitive type) Array
// to a List by adding one at a time
for (int number : intArray) {
    intList.add(number);
}

// Convert an Integer list to an int[] or
// other primitive type array one at a time
int[] newIntArray = new int[intList.size()];
for (int i=0; i<intList.size(); i++) {
    newIntArray[i] = intList.get(i);
}
```
Sets
Set Review

• Stores UNIQUE elements
• Check if element in Set (using .contains())
• Add element to set (using .add())
  • Returns false if already there
• Remove element (with .remove())
• Not guaranteed to store them in the order added

public interface Set<E>
extends Collection<E>

A collection that contains no duplicate elements.
Java API documentation
Set FAQs

1. How do I loop over a Set?

```java
[jshell> for (String s : mySet) { System.out.println(s); }
CS
201
```

2. How do I convert between lists and sets?

```java
[jshell> List<String> myList = new ArrayList<>();
myList ==> []

[jshell> myList.addAll(mySet);
$21 ==> true

[jshell> myList
myList ==> [CS, 201]
```
HashSet implementation of Set is very efficient

```java
public class HashSet<E>
extends AbstractSet<E>
implements Set<E>, Cloneable, Serializable {

This class implements the Set interface backed by a hash table (actually a HashMap instance). It makes no guarantees as to the iteration order of the set; in particular, it does not guarantee that the order will remain constant over time. This class permits the null element.

This class offers constant time performance for the basic operations (add, remove, contains and size), assuming the hash function disperses the elements properly among the buckets. Iterating over this set requires time proportional to the sum of the HashSet instance's size (the number of elements) plus the "capacity" of the backing HashMap instance (the number of buckets). Thus, it's very important not to set the initial capacity too high (or the load factor too low) if iteration performance is important.

Java API documentation
```
Count Unique Words?

```java
public static int countWordsHashSet(String[] words) {
    HashSet<String> mySet = new HashSet<>();
    for (String w : words) {
        mySet.add(w);
    }
    return mySet.size();
}

public static int countWordsArrayList(String[] words) {
    ArrayList<String> myList = new ArrayList<>();
    for (String w : words) {
        if (!myList.contains(w)) {
            myList.add(w);
        }
    }
    return myList.size();
}
```

For each word, constant time operation. “Linear complexity.”

For each word, must check all the words so far. “Quadratic complexity.”
TreeSet stores sorted

Two important implementations of Set interface:
• HashSet – Very efficient add, contains
• TreeSet – Nearly as efficient, keeps values sorted.

```java
String message = "computer science is so much fun";
char[] messageCharArray = message.toCharArray();
TreeSet<Character> uniqueChars = new TreeSet<>();
for (char c : messageCharArray) {
    uniqueChars.add(c);
}
System.out.println(uniqueChars);
```

[ , c, e, f, h, i, m, n, o, p, r, s, t, u]  

Prints all unique characters in order.
HashSet and TreeSet Implementations

HashSet and HashMap both implemented with a hash table data structure, will discuss next time.

TreeSet and TreeMap both implemented using a special kind of binary tree, will discuss later in the course.

public class HashSet<E>
extends AbstractSet<E>
implements Set<E>, Cloneable, Serializable

This class implements the Set interface, backed by a hash table (actually a HashMap instance). It makes no guarantees as to the iteration order of the set; in particular, it does not guarantee that the order will remain constant over time. This class permits the null element.

A NavigableSet implementation based on a TreeMap. The elements are ordered using their natural ordering, or by a Comparator provided at set creation time, depending on which constructor is used.

public class TreeMap<K,V>
extends AbstractMap<K,V>
implements NavigableMap<K,V>, Cloneable, Serializable

A Red-Black tree based NavigableMap implementation. The map
WOTO

Go to duke.is/gd9hy

Not graded for correctness, just participation.

Try to answer *without* looking back at slides and notes.

But do talk to your neighbors!
Maps
Map pairs keys with values

• Like an **address book**, lookup the value (address) of a key (person). Like a dictionary in Python.

<table>
<thead>
<tr>
<th>Keys</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>101 E. Main St.</td>
</tr>
<tr>
<td>Naomi</td>
<td>200 Broadway</td>
</tr>
<tr>
<td>Xi</td>
<td>121 Durham Ave.</td>
</tr>
</tbody>
</table>

• Map is an interface, must have methods like:
  • `put(k, v)`: Associate value $v$ with key $k$
  • `get(k)`: Return the value associated with key $k$
  • `containsKey(k)`: Return true if key $k$ is in the Map
Implementations: **HashMap, TreeMap**

Two major implementations:
- **HashMap**: Very efficient put, get, containsKey
- **TreeMap**: Nearly as efficient, keeps keys sorted

```java
import java.util.HashMap;
import java.util.Map;
import java.util.TreeMap;

Map<KEY_TYPE, VALUE_TYPE> addressBook = new TreeMap<>();
addressBook.put("Bob", "101 E. Main St.");
addressBook.put("Naomi", "200 Broadway");
addressBook.put("Xi", "121 Durham Ave.");
System.out.println(addressBook);
```

```
{Bob=101 E. Main St., Naomi=200 Broadway, Xi=121 Durham Ave.}
```
Check before you get

If you call `get(key)` on a key not in the map, returns null, can cause program to crash.

```java
6       Map<String, Integer> myMap = new HashMap<>();
7       int val = myMap.get("hi");
```

Exception in thread "main" java.lang.NullPointerException: Cannot invoke "java.lang.Integer.intValue()" because the return value of "java.util.Map.get(Object)" is null

Instead, check first with `.containsKey()`.

```java
6       Map<String, Integer> myMap = new HashMap<>();
7       if (myMap.containsKey("hi")) {
8           int val = myMap.get("hi");
9       }
```
Adding “default” values

Often want a “default” value associated with new keys (examples: 0, empty list, etc.). Two options:

• `.putIfAbsent(key, val)`
• Check if does not contain key

```java
6       Map<String, Integer> myMap = new HashMap<>();
7
8       myMap.putIfAbsent("hi", 0);
9
10      // Equivalent to line 8
11      if (!myMap.containsKey("hi")) {
12          myMap.put("hi", 0);
13      }
```
Updating maps

Single values
• `.get()` returns a *copy of the value*.
• Must use `.put()` again to update.

Collection values
• `.get()` returns *reference to collection*.
• Update the collection directly.

```
8     Map<String, Integer> myMap = new HashMap<>();
9     myMap.put("hi", 0);
10    int currentVal = myMap.get("hi");
11    myMap.put("hi", currentVal + 1);

14    Map<String, List<Integer>> otherMap = new HashMap<>();
15    otherMap.put("hi", new ArrayList<>());
16    otherMap.get("hi").add(0);
```
Counting with a Map

In this example we count how many of each character occur in message.

```java
String message = "computer science is so much fun";
char[] messageCharArray = message.toCharArray();
TreeMap<Character, Integer> charCounts = new TreeMap<>();

for (char c : messageCharArray) {
    if (!charCounts.containsKey(c)) {
        charCounts.put(c, 1);
    } else {
        int currentValue = charCounts.get(c);
        charCounts.put(c, currentValue + 1);
    }
}

System.out.println(charCounts);
{ =5, c=4, e=3, f=1, h=1, i=2, m=2, n=2, o=2, p=1, r=1, s=3, t=1, u=3}
```

Check if we have not seen c yet
Else get current value and increase
Comes in order because using TreeMap