Collections
Announcements

• Casual CS Dinner for Women Studying Computer Science: **Thursday, March 7 at 6PM in Gates 219**!

• RSVP through the email link sent out earlier this week.
Announcements

- Assignment 6 (NameSurfer) out, due next Wednesday, March 13 at 3:15PM.
- YEAH hours (assignment review hours) tonight from 7PM – 9PM in Hewlett T-175.
- Second Midterm exam next Monday, March 11 from 7PM – 10PM in MemAud.
  - Covers material up through and including today's lecture.
  - Solutions to first practice exam released now; another practice exam available (solutions out Friday).
  - Email Gil no later than 11:59PM tonight if you need to take the exam at an alternate time.
- Midterm review session this Saturday from 1PM – 3PM in Hewlett 200.
Organizing Data

- We have many ways of storing and organizing data in our programs:
  - **Strings** for holding sequences of characters.
  - **ArrayLists** for holding sequences of general objects.
  - Arrays for holding fixed-sized sequences.
  - **HashMaps** for associating data with one another.
- Are there other ways of organizing data?
- What do they look like?
The Collections Framework

• Java has a variety of collections classes for holding groups of data.
The Collections Framework

• Java has a variety of collections classes for holding groups of data.
• The three major ways of organizing data are
  • Lists, which store sequences
The Collections Framework

• Java has a variety of collections classes for holding groups of data.

• The three major ways of organizing data are
  • Lists, which store sequences,
  • Maps, which store key/value pairs
The Collections Framework

- Java has a variety of **collections classes** for holding groups of data.
- The three major ways of organizing data are
  - **Lists**, which store sequences,
  - **Maps**, which store key/value pairs, and
  - **Sets**, which store unordered data.
The Collections Framework

- Java has a variety of collections classes for holding groups of data.
- The three major ways of organizing data are
  - Lists, which store sequences,
  - Maps, which store key/value pairs, and
  - Sets, which store unordered data.
What is a Set?

- A set is a collection of distinct elements.
- Similar to an ArrayList, but elements are not stored in a sequence.
- Major operations are:
  - Adding an element.
  - Removing an element.
  - Checking whether an element exists.
- Useful for answering questions of the form “have I seen this before?”
HashSet<String> mySet = new HashSet<String>();
HashSet<String> mySet = new HashSet<String>();
mySet.add("CS106A");

To add a value to a HashSet, use the syntax

```
set.add(value)
```

HashSet<String> mySet = new HashSet<String>();
mySet.add("CS106A");
mySet.add("Ibex");
HashSet<String> mySet = new HashSet<String>();
mySet.add("CS106A");
mySet.add("Ibex");
mySet.add("137");
HashSet<String> mySet = new HashSet<String>();
mySet.add("CS106A");
mySet.add("Ibex");
mySet.add("137");
mySet.add("CS106A");

**If you **add** a value where the value already exists, nothing happens.**
HashSet<String> mySet = new HashSet<String>();
mySet.add("CS106A");
mySet.add("Ibex");
mySet.add("137");
mySet.add("CS106A");
mySet.add("CS106A");
mySet.contains("Ibex");
HashSet<String> mySet = new HashSet<String>();
mySet.add("CS106A");
mySet.add("Ibex");
mySet.add("137");
mySet.add("CS106A");

To see if a value exists:
\texttt{mySet.contains(value)}
HashSet<String> mySet = new HashSet<String>();
mySet.add("CS106A");
mySet.add("Ibex");
mySet.add("137");
mySet.add("CS106A");

mySet.contains("Ibex");
mySet.contains("CS106A");
HashSet<String> mySet = new HashSet<String>();
mySet.add("CS106A");
mySet.add("Ibex");
mySet.add("137");
mySet.add("CS106A");

mySet.contains("Ibex");
mySet.contains("CS106A");
HashSet<String> mySet = new HashSet<String>();
mySet.add("CS106A");
mySet.add("Ibex");
mySet.add("137");
mySet.add("CS106A");

mySet.contains("Ibex");
mySet.contains("CS106A");
mySet.contains("<(^_^)>")
HashSet<String> mySet = new HashSet<String>();
mySet.add("CS106A");
mySet.add("Ibex");
mySet.add("137");
mySet.add("CS106A");

mySet.contains("Ibex");
mySet.contains("CS106A");
mySet.contains("<(^_^)>");
Basic `set` Operations

- To insert an element:
  ```java
  set.add(value)
  ```

- To check whether a value exists:
  ```java
  set.contains(value)
  ```

- To remove an element:
  ```java
  set.remove(value)
  ```
Free Association
The “for each” Loop

• You can iterate across all elements of a HashSet by using the “for each” loop:

```java
for (Type value: set) {
    /* ... do something with value ... */
}
```

• This same loop can also be used on ArrayList, String, and arrays.
“For each” and Maps

- You can iterate across the keys of a `HashMap` by using the “for each” loop and the `keySet` method:

  ```java
  for (Type value : map.keySet()) {
    /* ... do something with value ... */
  }
  ```
Iterators

- To visit every element of a collection, you can use the “for each” loop:

  ```java
  for (ElemType elem : collection) {
      ...
  }
  ```

- Alternatively, you can use an `iterator`, an object whose job is to walk over the elements of a collection.

- The iterator has two commands:
  - `hasNext()`, which returns whether there are any more elements to visit, and
  - `next()`, which returns the next element and moves the iterator to the next position.
Java Iterators

ArrayList<Integer> myList = /* ... */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    Integer curr = iter.next();
    /* ... use curr ... */
}
/* ... use curr ... */
Java Iterators

```java
ArrayList<Integer> myList = /* ... */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    Integer curr = iter.next();
    /* ... use curr ... */
}
```
Java Iterators

```java
ArrayList<Integer> myList = /* ... */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    Integer curr = iter.next();
    /* ... use curr ... */
}
```
Java Iterators

ArrayList<Integer> myList = /* … */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    Integer curr = iter.next();
    /* … use curr … */
}
/* … use curr … */
Java Iterators

ArrayList<Integer> myList = /* ... */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    Integer curr = iter.next();
    /* ... use curr ... */
}

/* ... */
Java Iterators

```java
ArrayList<Integer> myList = /* ... */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    Integer curr = iter.next();
    /* ... use curr ... */
}
```
Java Iterators

```
ArrayList<Integer> myList = /* … */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    Integer curr = iter.next();
    /* … use curr … */
}
```

Java Iterators

```java
ArrayList<Integer> myList = /* … */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    Integer curr = iter.next();
    /* … use curr … */
}
```
Java Iterators

```java
ArrayList<Integer> myList = /* … */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    Integer curr = iter.next();
    /* … use curr … */
}
```

```
137 42 2718
```
ArrayList<Integer> myList = /* ... */

Iterator<Integer> iter = myList.iterator();

while (iter.hasNext()) {
    Integer curr = iter.next();
    /* ... use curr ... */
}

/* ... */
Java Iterators

```java
ArrayList<Integer> myList = /* ... */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    Integer curr = iter.next();
    /* ... use curr ... */
}
```

Java Iterators

```java
ArrayList<Integer> myList = /* ... */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    Integer curr = iter.next();
    /* ... use curr ... */
}
```
```java
ArrayList<Integer> myList = /* … */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    Integer curr = iter.next();
    /* … use curr … */
}
```
Java Iterators

```java
ArrayList<Integer> myList = /* ... */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    Integer curr = iter.next();
    /* ... use curr ... */
}
```
Java Iterators

ArrayList<Integer> myList = /* ... */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    Integer curr = iter.next();
    /* ... use curr ... */
}

/* ... use curr ... */
Java Iterators

```java
ArrayList<Integer> myList = /* ... */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    Integer curr = iter.next();
    /* ... use curr ... */
}
/* ... use curr ... */
```
Java Iterators

```java
ArrayList<Integer> myList = /* ... */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    Integer curr = iter.next();
    /* ... use curr ... */
}
```
Java Iterators

ArrayList<Integer> myList = /* … */

Iterator<Integer> iter = myList.iterator();

while (iter.hasNext()) {
    Integer curr = iter.next();
    /* … use curr … */
}

/* … */
Java Iterators

```java
ArrayList<Integer> myList = /* ... */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    Integer curr = iter.next();
    /* ... use curr ... */
}
```
Java Iterators

```java
ArrayList<Integer> myList = /* ... */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    Integer curr = iter.next();
    /* ... use curr ... */
}
```
Java Iterators

```java
ArrayList<Integer> myList = /* … */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    Integer curr = iter.next();
    /* … use curr … */
}
```
Java Iterators

ArrayList<Integer> myList = /* … */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    Integer curr = iter.next();
    /* … use curr … */
}
Java Iterators

ArrayList<Integer> myList = /* ... */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    Integer curr = iter.next();
    /* ... use curr ... */
}
/* ... */
ArrayList<Integer> myList = /* ... */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    Integer curr = iter.next();
    /* ... use curr ... */
}
/* ... use curr ... */
Java Iterators

```java
ArrayList<Integer> myList = /* … */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    Integer curr = iter.next();
    /* … use curr … */
}
```
Java Iterators

```java
ArrayList<Integer> myList = /* ... */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    Integer curr = iter.next();
    /* ... use curr ... */
}
```

Done!
Java Iterators

ArrayList<Integer> myList = /* ... */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
  Integer curr = iter.next();
  /* ... use curr ... */
}

/* ... use curr ... */
A Use Case for Iterators

- Because all collections have iterators, a method can return an iterator to indicate “here is some data to look at.”
- Internally, that data can be stored in any format.
- Separates the implementation (how the class works) from the interface (how the class is used).
A Word of Warning
A Word of Warning

- The following will loop forever on a nonempty collection:
  ```java
  while (collection.iterator().hasNext()) {
      /* ... */
  }
  ```
- Every time that you call `.iterator()`, you get back a new iterator to the start of the collection.
A Word of Warning

- The following will loop forever on a nonempty collection:

```java
while (collection.iterator().hasNext()) {
    /* … */
}
```

- Every time that you call `collection.iterator()`, you get back a new iterator to the start of the collection.
The Collections Framework

Collection

List
  * ArrayList

Set
  * HashSet

Map
  * HashMap
The Collections Framework

- Collection
  - List
    - ArrayList
    - LinkedList
  - Set
    - HashSet
    - TreeSet
  - Map
    - HashMap
    - TreeMap

Plus a lot more!
The Collections Framework

Collection

List
- ArrayList
- LinkedList

Set
- HashSet
- TreeSet

Map
- HashMap
- TreeMap

Plus a lot more!
TreeSet

- TreeSet is similar to HashSet, except that the values in a TreeSet are stored in sorted order.
- Iterating over a TreeSet guarantees that the elements are visited in ascending order.
- TreeSet is a bit slower than HashSet, so it's best used only when you really need things in sorted order.
Levels of Specificity

- To create a map, set, or list, you must choose a specific implementation (i.e. `ArrayList`, `HashMap`, etc.)
- You can store maps, sets, or lists in variables of type `Map`, `Set`, or `List`.
  - Similar to `GObject` versus `GOval`, `GRect`, etc.
- Lets you say “I just need key/value pairs” rather than “I need key/value pairs specifically stored as a `HashMap`”
**TreeMap**

- **TreeMap** is similar to **HashMap**, except that the *keys* in a **TreeMap** are stored in sorted order.
- Like **TreeSet**, iteration over the keys visits the keys in sorted order.
- The **TreeMap** has several impressive methods that don't exist on the normal **HashMap**.
- There is slight performance cost to using **TreeMap**.