CS2110 Recitation 09.
Interfaces Iterator and Iterable.
Nested, Inner, and static classes

We work often with a class C (say) that implements a
• bag: unordered collection of elements (duplicates allowed)
• set: bag in which no duplicated allowed (call it a unibag!)  
• list: ordered collection of elements

We show you how to fix class C<T> so that you can write:
```
C<String> ob = new C<String>();
Populate ob with some elements;
for (String s: ob) {
    do something with s  
}  
```

We show you how to fix class C<T> so that you can write:
```
C<String> ob = new C<String>();
Populate ob with some elements;
for (String s: ob) {  
    do something with s  
}
```

To implement interface Iterator<T> in java.util
```
interface Iterator<T> {
    /** Return true if the enumeration has more elements */
    public boolean hasNext();
    /** Return the next element of the enumeration.
       Throw a NoSuchElementException if there are no more. */
    public T next();
    /** Remove the last element returned by the iterator.
       Throw UnsupportedOperationException if you don't want
       to implement this operation. We don't. */
    public void remove();
}
```

To implement interface Iterator<T> in java.util
```
interface Iterator<T> {
    /** Return true if the enumeration has more elements */
    public boolean hasNext();
    /** Return the next element of the enumeration.
       Throw a NoSuchElementException if there are no more. */
    public T next();
    /** Remove the last element returned by the iterator.
       Throw UnsupportedOperationException if you don't want
       to implement this operation. We don't. */
    public void remove();
}
```

Example of a class that implements Iterator<T>
```
Recall implementation of hashing from last week. Each element of b is either
1. null
2. A HashEntry object with isInSet false
3. A HashEntry object with isInSet true

We need a class that enumerates the elements in the objects in alternative 3.
```

Class HashSetIterator
```
/** An instance is an Iterator of this HashSet */
private class HashSetIterator<T> implements Iterator<T> {
    private int pos = -1;
    private int enumerated = 0;
    /** = "there is another element to enumerate". */
    @Override public boolean hasNext() {
        return enumerated != size;
    }
    // continued on next slide
    field size of class HashSet
```

Class HashSetIterator
```
/** An instance is an Iterator of this HashSet */
private class HashSetIterator<T> implements Iterator<T> {
    private int pos = -1;
    private int enumerated = 0;
    /** = "there is another element to enumerate". */
    @Override public boolean hasNext() {
        return enumerated != size;
    }
    // continued on next slide
    field size of class HashSet
```

Class HashSetIterator
```
/** An instance is an Iterator of this HashSet */
private class HashSetIterator<T> implements Iterator<T> {
    private int pos = -1;
    private int enumerated = 0;
    /** = "there is another element to enumerate". */
    @Override public boolean hasNext() {
        return enumerated != size;
    }
    // continued on next slide
    field size of class HashSet
```

Class HashSetIterator
```
/** An instance is an Iterator of this HashSet */
private class HashSetIterator<T> implements Iterator<T> {
    private int pos = -1;
    private int enumerated = 0;
    /** = "there is another element to enumerate". */
    @Override public boolean hasNext() {
        return enumerated != size;
    }
    // continued on next slide
    field size of class HashSet
```

Class HashSetIterator
```
/** An instance is an Iterator of this HashSet */
private class HashSetIterator<T> implements Iterator<T> {
    private int pos = -1;
    private int enumerated = 0;
    /** = "there is another element to enumerate". */
    @Override public boolean hasNext() {
        return enumerated != size;
    }
    // continued on next slide
    field size of class HashSet
```

Class HashSetIterator
```
/** An instance is an Iterator of this HashSet */
private class HashSetIterator<T> implements Iterator<T> {
    private int pos = -1;
    private int enumerated = 0;
    /** = "there is another element to enumerate". */
    @Override public boolean hasNext() {
        return enumerated != size;
    }
    // continued on next slide
    field size of class HashSet
```

Class HashSetIterator
```
/** An instance is an Iterator of this HashSet */
private class HashSetIterator<T> implements Iterator<T> {
    private int pos = -1;
    private int enumerated = 0;
    /** = "there is another element to enumerate". */
    @Override public boolean hasNext() {
        return enumerated != size;
    }
    // continued on next slide
    field size of class HashSet
```

Class HashSetIterator
```
/** An instance is an Iterator of this HashSet */
private class HashSetIterator<T> implements Iterator<T> {
    private int pos = -1;
    private int enumerated = 0;
    /** = "there is another element to enumerate". */
    @Override public boolean hasNext() {
        return enumerated != size;
    }
    // continued on next slide
    field size of class HashSet
```

Class HashSetIterator
```
/** An instance is an Iterator of this HashSet */
private class HashSetIterator<T> implements Iterator<T> {
    private int pos = -1;
    private int enumerated = 0;
    /** = "there is another element to enumerate". */
    @Override public boolean hasNext() {
        return enumerated != size;
    }
    // continued on next slide
    field size of class HashSet
```
HashSetIterator has to be an inner class

```java
public class HashSet<T> {
    private HashEntry<T>[] b;
    private int size = 0;
    public boolean add(T x) {...}
}
```

private class HashSetIterator implements Iterator<T> {
    public boolean hasNext() {...}
    public T next() {...}
    public void remove() {...}
}

These refer to size and b

It has to be defined inside class HashSet

These refer to type T

HashSet implements Iterable<T>, so you can replace the declaration of it and the while loop by the foreach loop. "syntactic sugar"

Using the foreach loop

```java
HashSet<Integer> hs = new HashSet<Integer>();
Add a bunch of integers to hs;

// Print all elements in hs
for (Integer k : hs) {
    System.out.println(k);
}
```

Using the iterator

```java
public class HashSet<T> {
    public Iterator<T> iterator() {
        return new HashSetIterator();
    }
}
```

public class HashSet<T> implements Iterable<T> {
    ... add(…) iterator()
    Iterator<T> iterator() {
        return new HashSetIterator();
    }
    ... remove() ...
}

HashSet implements Iterable<T>, so you can replace the declaration of it and the while loop by the foreach loop. "syntactic sugar"

Using the foreach loop

```java
HashSet<Integer> hs = new HashSet<Integer>();
Add a bunch of integers to hs;

// Print all elements in hs
for (Integer k : hs) {
    System.out.println(k);
}
```

These refer to size and b

It has to be defined inside class HashSet

These refer to type T

Using the iterator

```java
public class HashSet<T> {
    public Iterator<T> iterator() {
        return new HashSetIterator();
    }
}
```

public class HashSet<T> implements Iterable<T> {
    ... add(…) iterator()
    Iterator<T> iterator() {
        return new HashSetIterator();
    }
    ... remove() ...
}

HashSet implements Iterable<T>, so you can replace the declaration of it and the while loop by the foreach loop. "syntactic sugar"
Don’t try to change the set in a foreach!!

```java
HashSet<Integer> hs = new HashSet<Integer>();
Add a bunch of strings to hs;
// Print all elements in hs
for (Integer k : hs) {
    hs.add(-k);
}
```

This may change array b and int field size. May cause rehash.

HashSet< Integer > hs = new HashSet< Integer >();
Add a bunch of strings to hs;
// Print all elements in hs
for (Integer k : hs) {
    hs.add(-k);
}

Don’t try to change the set!

```java
Iterator<Integer> it = hs.iterator();
while (it.hasNext()) {
    Integer k = it.next();
    hs.add(-k);
}
```

HashSet< Integer > hs = new HashSet< Integer >();
Add a bunch of strings to hs;
// Print all elements in hs
for (Integer k : hs) {
    hs.add(-k);
}

Think of HashSetIterator objects also as being inside a HashSet object. Then, normal inside-out rule shows you that hasNext() and next() can reference b and size.

HashSet<C> hs = new HashSet<C>();
... 
Iterator<C> it1 = hs.iterator();
Iterator<C> it2 = hs.iterator();

Diagram: two HashSetIterator objects in HashSet object. Two enumerations of set going on at same time?

HashSetIterator is an inner class of HashSet

Declared within HashSet, often made private so can’t be referenced directly from outside

HashSetIterator is in each HashSet object

A foreach loop within a foreach loop

```java
HashSet<Integer> hs = new HashSet<Integer>();
Add a bunch of strings to hs;
for (Integer k : hs) {
    for (Integer h : hs) {
        // Compare set elements k and h in some way
    }
}
```

A nested class with other nested classes

```java
public class HashSet<T> implements Iterable<T> {
    public boolean add(T x) {
        // ... 
    }
    @Override 
    public Iterator<T> iterator() {
        // ... 
    }
    private class HashSetIterator implements Iterator<T> {} 
    private static class HashEntry<T> {} 
}
```

A nested class and inner class

```java
public class HashSet<T> implements Iterable<T> {
    public boolean add(T x) {
        // ... 
    }
    @Override 
    public Iterator<T> iterator() {
        // ... 
    }
    private class HashSetIterator implements Iterator<T> {} 
    private class HashEntry<T> {} 
}
```

A nested class

```java
public class HashSet<T> implements Iterable<T> {
    public boolean add(T x) {
        // ... 
    }
    @Override 
    public Iterator<T> iterator() {
        // ... 
    }
    private class HashSetIterator implements Iterator<T> {} 
    private static class HashEntry<T> {} 
}
```
Nested class    Inner class    static nested class

```java
public class HashSet<T> implements Iterable<T>
{
    public boolean add(T x)
    ...
    @Override public Iterator<T> iterator()
    private class HashSetIterator implements Iterator<T> {}
    private static class HashEntry<T> {}
}
```

**Inner class**: a nested class that is not static. When instances are created, they live within an object of the outer class.

HashSetIterator is an inner class. It has to live within a HashSet object so that the objects can reference fields `b` and `size`. See slide 15.

Static nested class: a nested class that is static. When instances are created, they do not live within an object of the outer class.

HashEntry is a static nested class. Its objects do not need to be in HashSet objects because it does not reference HashSet fields or instance methods.

Nested class    Inner class    static nested class

Make a class an inner class so that its objects can reference fields or instance methods of the outer class.

Make a class `SNC` a static nested class within class `C` when:
1. `SNC` is used only within `C`, and there is no need for program parts outside `C` to know about `SNC`.
   Example: HashEntry
2. `SNC` does not reference any fields or instance methods of `C`.
   Example: HashEntry

**Effect**: Nesting `SNC` within `C` hides it from the outside world. Only those interested in how `C` is implemented need to know about it. Making `SNC` static is more efficient — there is only one copy of the class; it does not reside in objects of class `C`.

There are certain restrictions on inner classes and nested static classes. We don't go into them.

You have seen one nested static class: HashEntry

You have seen several inner classes: HashSetIterator and some classes that are used to help implement listening to GUI events — discussed in that lecture.