CS2110 Recitation 07.
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:

```java
C<String> ob = new C<String>();
Populate ob with some elements;
for (String s: ob) {
    do something with s
}
```

### Start with interface `Iterator`

A class that implements `Iterator` needs three functions that make it easy to "enumerate" the elements of a collection — a bag, a set, a list, whatever.

**Required functions:**

- `hasNext()`
- `next()`  
- `remove()`

In `java.util` to enumerate:

```java
interface Iterator<T> {
    /** Return true iff 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.

```java
public class HashSetIterator<T> implements Iterator {
    // all elements in b[0..pos] have been enumerated
    private int pos = -1;
    // number of elements that have been enumerated
    private int enumerated = 0;
    /** = "there is another element to enumerate". */
    public @Override boolean hasNext() {
        return enumerated != size;
    }
    // continued on next slide
    // field size of class HashSet
}
```

Or as

```java
public class HashSetIterator {
    ... = "there is another element to enumerate". */
    public @Override boolean hasNext() {
        return enumerated != size;
    }
    // continued on next slide
    // field size of class HashSet
}
```
HashSetIterator has to be an inner class

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

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

These refer to size and b

These refer to size and b

HashSet has to be an inner class

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

    private class HashSetIterator<T> implements Iterator {
        public boolean hasNext() {
            ... // These refer to size and b
        }
        public T next() {
            ... // These refer to size and b
        }
        public void remove() {
            ... // These refer to size and b
        }
    }
}

Using the iterator

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

// Print all elements in hs
Iterator<Integer> it = hs.iterator();
while (it.hasNext()) {
    Integer k = it.next();
    System.out.println(k);
}

HashSet implements Iterable, so you can replace the declaration of it and the while loop by the foreach loop. “syntactic sugar”

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

    private class HashSetIterator<T> implements Iterator {
        public boolean hasNext() {
            return false; // This is always false for HashSet
        }
        public T next() {
            return null; // This will never return
        }
        public void remove() {...}
    }
}
Don’t try to change the set in a foreach!!

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 cause rehash. hs’s class invariant (meanings of hs.pos and it.enumerated) no longer holds.

Don’t do this either →

HashSet 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

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

A foreach loop within a foreach loop

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
    }}

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

Nested class Inner class static nested class

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

Nested class: a class declared inside another:

HashSetIterator and HashEntry are declared within class HashSet, so they are nested classes.

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 is objects can reference fields b and size. See slide 15!
**Nested class**    **Static nested class**    **Inner class**

```java
public class HashSet<T> implements Iterable<T> {
    public boolean add(T x) {
        ...
    }

    @Override
    public Iterator<T> iterator() {
        ...
    }

    private class HashSetIterator<T> implements Iterator<T> {
        ...
    }

    private static class HashEntry<T> {
        ...
    }
}
```

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

HashSet 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**

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.

**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.