

CS2110 Recitation Week 9.
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
}
```

foreach loop

1

Interface Iterator

Start with interface *Iterator*. in java.util

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()
```

To enumerate: to provide a list of

2

To implement interface Iterator<T>
in java.util

```
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();
}
```

3

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.

```
b [ ]@xy [ ]@xy
"abc"
"235"
"aaa"
"1$2"
"xy"
```

4

Class HashSetIterator

```
/** An instance is an Iterator of this HashSet */
private 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;
    }
}
```

field size of class HashSet

// continued on next slide

```
/** = the next element to enumerate.
 * Throw a NoSuchElementException if no elements left */
public @Override T next() {
    if (!hasNext()) throw new NoSuchElementException();
    pos= pos+1;
    while (b[pos] == null || !b[pos].isInSet) {
        pos= pos+1;
    }
    enumerated= enumerated+1;
    return (T)b[pos].element;
}

/** Remove is not supported. */
public @Override void remove() throws ... {
    throw new UnsupportedOperationException();
}
```

Class
HashSetIterator

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() { ... }
    }
}
    
```

It has to be defined inside class HashSet

These refer to size and b

```

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);
}
    
```

Using the iterator

```

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

Using the iterator

```

hs HS@24    it HSI@bc

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);
}
    
```

```

... add(...) iterator()
HashSetIterator
b [ ... ]
    HSI@bc
    hasNext() {...}
    next() {...}
    
```

```

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

Interface Iterable<T>

In java.lang

Requires one method:

```

/** Return an Iterator over a set of elements of type T */
public Iterator<T> iterator()
    
```

Java API says "set", but should say "collection" –a set, a bag, a list, whatever

If class C implements Iterable<T>, we can write

```

for (T v : object) {...}
    
```

```

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

    /** Return an Iterator for enumerating the set. */
    public @Override Iterator<T> iterator() {
        return new HashSetIterator<T>();
    }

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

Using the foreach loop

```

HashSet<Integer> hs= new HashSet<Integer>();
Add a bunch of strings to hs;
// Print all elements in hs
Iterator<Integer> it= hs.iterator();
while (it.hasNext()) {
    Integer k= it.next();
    System.out.println(k);
}
    
```

for (Integer k : hs) {
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> {
    public @Override Iterator<T> iterator()
    private class HashSetIterator<T> implements Iterator
    ...
}
    
```

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 change array **b** and int field **size**. May cause rehash. **hs**'s class invariant (meanings of **hs.pos** and **it.enumerated**) no longer holds.

```

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

Don't do this either →

13

HashSetIterator is an **inner class** of HashSet

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

```

hs HS@24
HS@24
... add(...) iterator()
HashSetIterator
size 20 b C[]@24
    
```

HashSetIterator is in each HashSet object

```

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

14

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();
    
```

```

it1 HSI@bc
hs HS@24 it2 HIS@d
HS@24
... add(...) iterator()
HashSetIterator
size 20 b C[]@24
HIS@d
hasNext() {...}
next() {...}
HSI@bc
hasNext() {...}
next() {...}
    
```

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

15

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 @Override Iterator<T> iterator()
    private class HashSetIterator<T> implements Iterator
}
    
```

16

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.

17

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> {}
}
    
```

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!

18

Nested class Static nested class Inner 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> {}
}
```

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.

19

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`.

20

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.

21