CS 367 Announcements
Thursday, January 29, 2015

Email skrentny@cs.wisc.edu by TOMORROW if you:

- Have a conflict with any of the exam dates.
  Email with subject "CS 367 Exam # Conflict" as in "CS 367 Exams 1 & 2 Conflict"
  and include an appropriate explanation:
  - course/exam – include course name/number, time, instructor name and email
  - VISA – include any accommodation(s) requested
  - other – include concise explanation
- Participate in religious observances that may interfere with course requirements.
  Email with subject put "CS 367 Religious Observance" and include a date and explanation.
- Have a VISA from the McBurney Disability Resource Center.
  Email with subject "CS 367 VISA" and request an appointment.

Course Website - http://pages.cs.wisc.edu/~cs367-1/
Sign Up for Piazza

Program 1 assigned Thursday morning
Homework 1 assigned late tomorrow

Last Time

Coursework
Lists
- implementing the ListADT using an array (SimpleArrayList)

Java API Lists
Iterators
- concept

Today

Iterators
- coding iterator interfaces and the Java API
- using iterators
- making a class iterable
- options for implementing iterators

Exceptions Review
- throwing
- handling

Next Time

Read: *Exceptions*
Exceptions Review
- execution
- practice with exception handling
- throws and checked vs. unchecked
- defining

Interfaces - Iterators in Java API

Iterable<T> **interface in** **`java.lang`** we need to specify that the container is ITERABLE
specifies the operation to get an iterator to step through a collection:

- Iterator<T> iterator()

- Returns an iterator "pointing" at the 1st item of the collection
  OR: the "end mark" if the container is empty
- This method is implemented in the container class

How to make the class iterable?

Iterator<E> **interface in** **`java.util`**
specifies the operations that iterators can do:

- boolean hasNext()
  returns true IFF the iterator is pointing an item !!!

- E next()
  returns a reference to the item the iterator is pointing at
  AND advance the iterator to the next position (we might don't have the next item…)

- void remove() //"optional"

  / / We will only learn how to make this optional

Use - Iterators

Suppose `words` is a `SimpleArrayList<String>` that implements the `Iterable` Interface.
→ **Write a code fragment** that gets an iterator, named `itr`, from `words`.

```
Iterator itr =  new Iterator < String >; / / WRONG

Iterator < String > itr = words.iterator ( ); / / CORRECT
```

Suppose `words` is a `SimpleArrayList<String>` and `itr` is an iterator for `words`.
→ **Write a code fragment** that uses `itr` to print each item in `words`.

```
for ( int i = 0; i < words.length; i + + ){
        S.o.pln ( itr.next() );
}                                       / /     WRONG


while ( itr.hasNext() ){
        s.o.pln ( itr.next() );
}                                   / / Correct


itr = words.iterator  ( );
while ( itr.iterator ( ) ) {
        s.o.pln( itr.next().length() );
}


– – – – – – – –
Note:
iterator has only ONE time use,
if you want to go through the collection again
just ask for the container for another iterator!
```

Use - Iterators

homework!

Assume `SimpleArrayList<String>` implements the `Iterable` Interface.
→ **Complete the method** using iterators to determine `list` contains duplicates.

```
public boolean hasDups(SimpleArrayList<String> list) {
```

Making Array Bags `Iterable`

how to make a class iterable?

Modify the Generic `BagADT` Interface?

```
import java.util.*;
public interface BagADT<E> {
    void add(E item);
    E remove() throws NoSuchElementException;
    boolean isEmpty();

    Iterable<E> iterator();  //← ADD to Bag operations?
}
```

This is the approach in the reading, which is WRONG!
We will follow JAVA's approach…

Generic `ArrayBag` Class

, Iterable <E>

```
import java.util.*;
public class ArrayBag<E> implements BagADT<E> {

    // *** Data members (fields) ***
    private E[] items;
    private int numItems;
    private static final int INIT_SIZE = 100;

    //*** required BagADT methods ***
    void add(E item) { ... }
    E remove() throws NoSuchElementException { ... }
    boolean isEmpty() { ... }

    / / * * * required Iterable methods * * *
    / / we only need one…

    public Iterator < E > iterator ( ) {
        return new ArrayBagIterator ( items, numItems ) ;
    }



}
```

How to implement an iterator!!!

Implementation - Options for Iterator Classes

Indirect Access

- Iterator uses the container's operations (methods) to access the collection of items
  We use size( ) & get( )

- container class constructs the iterator are passing a reference to itself (this)

- it only works if the container class has sufficient operations that the iterator can use to step through the collection

Direct Access

- Go right to the container's instance variable to access the collection of items

- Advantage: Direct access is faster
- Disadvantage: More error prone

- Container class constructs the iterator by passing the instances variables needed by the iterator for that direct access

Implementation - Indirect Access `SimpleArrayListIterator` Class

```
import java.util.*;
public class SimpleArrayListIterator<E> implements Iterator<E> {

    private SimpleArrayList < E >  myList;
    private int currentPosition;        / / sometimes abbreviated as “curr”, “currPos”


    public SimpleArrayListIterator(  SimpleArrayList <E> list          ) {

        myList = list;
        currentPosition = 0;


    }

    public boolean hasNext() {
       return currentPosition < myList.size( );      / / indirect access on the last page

    }

    public E next() {

        if ( currentPosition > = myList.size( )  )
              throw new NullSuchElementException ( );

        E result = myList.get ( currentPosition );    / / indirect access
        currentPosition + +;  / / advance
        return result;

    }

    public void remove() {

        throw new UnsupportedOperationException ( );
        / / This is the “I don’t want to implement it excepetion” !

    }
}
```

Implementation - Direct Access `ArrayBagIterator` Class

```
import java.util.*;
public class ArrayBagIterator<E> implements Iterator<E> {

      private E [ ] myItems;
      private int myNumItems;
      private int currPos;


    public ArrayBagIterator(  E [ ] items, numItems          ) {

        myItems = items;
        myNumItems = numItems;
        currPos = 0;

    }

    public boolean hasNext() {
       return currPos < myNumItems;

    }

    public E next() {
       if ( currPos > = myNumItems )
             throws new NoSuchElementException( );

       E result = myItems[ currPos ];
       currPos + + ;
       return result;

       / / OR:  return myItems [ currPos ++ ];

    }

    public void remove() {

        throw new UnsupportedOperationException();

    }
}
```

After you get a iterator, if you add an item, the container is changed!

result: the iterator is invalid!

fastFail

Exception Throwing – Signaling a Problem

Java Syntax

```
throw exceptionObject;
```

Example

Exception Handling – Resolving a Problem

Java Syntax

```
try {
   // try block
   code that might cause an exception to be thrown

} catch (ExceptionType1 identifier1) {
   // catch block
   code to handle exception type 1

} catch (ExceptionType2 identifier2) {
   // catch block
   code to handle exception type 2

}
... more catch blocks

finally {
   // finally block - optional
   code always executed when try block is entered
}
```

Example