In-Class Extended Example Ch. 6.4

- Form teams of two to three neighbors
- Hand out printouts of Iterator.html
  - http://docs.oracle.com/javase/7/docs/api/java/util/Iterator.html
- Close books
- We will go through the steps for designing an IDM for Iterator
- After each step, we will stop & discuss as a class

Task I: Determine Characteristics

Step 1: Identify:

- Functional units
- Parameters
- Return types and return values
- Exceptional behavior

work …

Task I: Determine Characteristics

Step 1: Identify:

- hasNext() – Returns true if more elements
- E next() – Returns next element
  - Exception: NoSuchElementException
- void remove() – Removes the most recent element returned by the iterator
  - Exception: Unsupported-OperationException
  - Exception: IllegalStateException
- parameters: state of the iterator
  - iterator state changes with next(), and remove() calls
  - modifying underlying collection also changes iterator state

Task I: Determine Characteristics

Step 2: Develop Characteristics

Table A:

Method	Params	Returns	Values	Exception	Ch ID	Character -istic	Covered by
hasNext	state	boolean	true, false				
next	state	E element generic	E, null				
remove	state						

work …

Task I: Determine Characteristics

Step 2: Develop Characteristics

Table A:

Method	Params	Returns	Values	Exception	Ch ID	Character -istic	Covered by
hasNext	state	boolean	true, false		C1	More values	
next	state	E element generic	E, null				
remove	state						

Task I: Determine Characteristics

Step 2: Develop Characteristics

Table A:

Method	Params	Returns	Values	Exception	Ch ID	Character -istic	Covered by
hasNext	state	boolean	true, false		C1	More values	
next	state	E element generic	E, null		C2	Returns non-null object	
remove	state						

Task I: Determine Characteristics

Step 2: Develop Characteristics

Table A:

Method	Params	Returns	Values	Exception	Ch ID	Character-istic	Covered by
hasNext	state	boolean	true, false		C1	More values	
next	state	E element generic	E, null		C2	Returns non-null object	
				NoSuchElement			C1
remove	state						

Task I: Determine Characteristics

Step 2: Develop Characteristics

Table A:

Method	Params	Returns	Values	Exception	Ch ID	Character -istic	Covered by
hasNext	state	boolean	true, false		C1	More values	
next	state	E element generic	E, null		C2	Returns non-null object	
				NoSuchElement			C1
remove	state			Unsupported	C3	remove() supported	

Task I: Determine Characteristics

Step 2: Develop Characteristics

Table A:

Method	Params	Returns	Values	Exception	Ch ID	Character-istic	Covered by
hasNext	state	boolean	true, false		C1	More values	
next	state	E element generic	E, null		C2	Returns non-null object	
				NoSuchElement			C1
remove	state			Unsupported	C3	remove() supported	
				IllegalState	C4	remove() constraint satisfied	

Done!

Task I: Determine Characteristics

Step 4: Design a partitioning

Which methods is each characteristic relevant for?

How can we partition each characteristic?

Table B:

ID	Characteristic	hasNext()	next()	remove()	Partition
C1	More values				
C2	Returns non-null object				
C3	remove() supported				
C4	remove() constraint satisfied				

work …

Task I: Determine Characteristics

Step 4: Design a partitioning

Relevant characteristics for each method

Table B:

ID	Characteristic	hasNext()	next()	remove()	Partition
C1	More values	X	X	X	
C2	Returns non-null object		X	X	
C3	remove() supported			X	
C4	remove() constraint satisfied			X	

Task I: Determine Characteristics

Step 4: Design a partitioning

Table B:

ID	Characteristic	hasNext()	next()	remove()	Partition
C1	More values	X	X	X	{true, false}
C2	Returns non-null object		X	X	{true, false}
C3	remove() supported			X	{true, false}
C4	remove() constraint satisfied			X	{true, false}

Done with task I!

Task II: Define Test Requirements

- Step 1: Choose coverage criterion
- Step 2: Choose base cases if needed

work …

Task II: Define Test Requirements

- Step 1: Base coverage criterion (BCC)
- Step 2: Happy path (all true)
- Step 3: Test requirements …

Task II: Define Test Requirements

- Step 3: Test requirements

Table C:

Method	Characteristics	Test Requirements	Infeasible TRs
hasNext	C1		
next	C1 C2		
remove	C1 C2 C3 C4		

work …

Task II: Define Test Requirements

- Step 3: Test requirements

Table C:

Method	Characteristics	Test Requirements	Infeasible TRs
hasNext	C1	{**T**, F}	
next	C1 C2	{**TT**, FT, TF}	
remove	C1 C2 C3 C4	{**TTTT**, FTTT, TFTT, TTFT, TTTF}	

Task II: Define Test Requirements

- Step 4: Infeasible test requirements

Table C:

C1=F: has no values
C2=T: returns non-null object

Method	Characteristics	Test Requirements	Infeasible TRs
hasNext	C1	{**T**, F}	none
next	C1 C2	{**TT**, FT, TF}	FT
remove	C1 C2 C3 C4	{**TTTT**, FTTT, TFTT, TTFT, TTTF}	FTTT

Task II: Define Test Requirements

- Step 5: Revised infeasible test requirements

Table C:

Method	Characteristics	Test Requirements	Infeasible TRs	Revised TRs	# TRs
hasNext	C1	{**T**, F}	none	n/a	2
next	C1 C2	{**TT**, FT, TF}	FT	FT → F**F**	3
remove	C1 C2 C3 C4	{**TTTT**, FTTT, TFTT, TTFT, TTTF}	FTTT	FTTT → F**F**TT	5

Done with task II!

Task III: Automate Tests

- First, we need an implementation of Iterator
  - (Iterator is just an interface)
  - ArrayList implements Iterator
- Test fixture has two variables:
  - List of strings
  - Iterator for strings
- setUp()
  - Creates a list with two strings
  - Initializes an iterator

Task III: Automate Tests

- remove() adds another complication …

"The behavior of an iterator is unspecified if the underlying collection is modified while the iteration is in progress in any way other than by calling this method."

- Subsequent behavior of the iterator is undefined!
  - This is a constraint on the caller: i.e. a precondition
- Preconditions are usually bad:
  - Legitimate callers often make the call anyway and then depend on whatever the implementation happens to do
  - Malicious callers deliberately exploit "bonus behavior"

Task III: Automate Tests

A merely competent tester would not test preconditions

All specified behaviors have been tested!

A good tester …

… with a mental discipline of quality …

would ask …

What happens if a test violates the precondition?

Tests That Violate Preconditions

- Finding inputs that violate a precondition is easy
  - But what assertion do you write in the JUnit test?

```
List<String> list = … // [cat, dog]
Iterator<String> itr = list.iterator();
itr.next();              // can assert!   return value is "cat"
list.add("elephant");   // just killed the iterator
itr.next();              // cannot assert!
```

- Note: In the Java collection classes, the Iterator precondition has been replaced with defined behavior
  - ConcurrentModificationException
- That means we can write tests in this context

Task I: Determine Characteristics

Cycle back to add another exception—Table A revised:

Method	Params	Returns	Values	Exception	Ch ID	Character -istic	Covered by

work …

Task I: Determine Characteristics

Cycle back to add another exception—Table A revised:

Method	Params	Returns	Values	Exception	Ch ID	Character-istic	Covered by
hasNext	state	boolean	true, false		C1	More values	
				Concurrent Modification			**C5**
next	state	E element generic	E, null		C2	Returns non-null	
				NoSuchElement			C1
				Concurrent Modification			**C5**
remove	state			Unsupported	C3	remove() supported	
				IllegalState	C4	remove() constraint satisfied	
				Concurrent Modification	**C5**	Collection not modified	

Task II: Define Test Requirements

- Cycle back to Step 5: Revised infeasible test requirements

Table C revised:

Method	Characteristics	Test Requirements	Infeasible TRs	Revised TRs	# TRs

work …

Task II: Define Test Requirements

- Cycle back to Step 5: Revised infeasible test requirements

Table C revised:

Method	**Characteristics**	**Test Requirements**	**Infeasible TRs**	**Revised TRs**	**# TRs**
hasNext	C1 **C5**	{**TT**, FT, TF}	none	n/a	3
next	C1 C2 **C5**	{**TTT**, FTT, TFT, TTF}	FTT TTF	FTT → F**F**T TTF → T**F**F	4
remove	C1 C2 C3 C4 **C5**	{**TTTTT**, FTTTT, TFTTT, TTFTT, TTTFT, TTTTF}	FTTTT	FTTTT → F**F**TTT	6

Task III: Automate Tests

All tests are on the book website:

http://cs.gmu.edu/~offutt/softwaretest/java/IteratorTest.java