ISP Example

Dr. Isaac Griffith

Idaho State University

Inspiration

“Blame doesn’t fix bugs.” – Anonymous

Outcomes

After today's lecture you will be able to:

- Utilize ISP to develop tests

In-Class Extended Example

- Download the Iterator handout
- 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: `UnsupportedOperationException`
  - 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

Develop Characteristics
Table A:

Method	Params	Returns	Values	Exception	ChID	Characteristic	Covered By
`hasNext`	state	boolean	true, false				
`next`	state	E	E, null				
`remove`	state						

Work...

Task I: Determine Characteristics

Develop Characteristics
Table A:

Method	Params	Returns	Values	Exception	ChID	Characteristic	Covered By
`hasNext`	state	boolean	true, false		C1	More values	
`next`	state	E	E, null				
`remove`	state						

Task I: Determine Characteristics

Develop Characteristics
Table A:

Method	Params	Returns	Values	Exception	ChID	Characteristic	Covered By
`hasNext`	state	boolean	true, false		C1	More values	
`next`	state	E	E, null		C2	Returns non-null object	
`remove`	state						

Task I: Determine Characteristics

Develop Characteristics
Table A:

Method	Params	Returns	Values	Exception	ChID	Characteristic	Covered By
`hasNext`	state	boolean	true, false		C1	More values	
`next`	state	E	E, null		C2	Returns non-null object	
				NoSuchElement			C1
`remove`	state						

ROAR

Task I: Determine Characteristics

Develop Characteristics
Table A:

Method	Params	Returns	Values	Exception	ChID	Characteristic	Covered By
`hasNext`	state	boolean	true, false		C1	More values	
`next`	state	E	E, null		C2	Returns non-null object	
				NoSuchElement			C1
`remove`	state			Unsupported	C3	remove() supported	

Task I: Determine Characteristics

Develop Characteristics
Table A:

Method	Params	Returns	Values	Exception	ChID	Characteristic	Covered By
`hasNext`	state	boolean	true, false		C1	More values	
`next`	state	E	E, null		C2	Returns non-null object	
				NoSuchElement			C1
`remove`	state			Unsupported	C3	remove() supported	
				IllegalState	C4	remove() constraint sat	

Task I: Determine Characteristics

Step4: 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 sat				

Work...

Task I: Determine Characteristics

Step4: 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 sat			X	

Task I: Determine Characteristics

Step4: 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 sat			X	{True,False}

Done with task I!

ROAR

Task II: Define Test Req'ts

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

Work...

Task II: Define Test Req’ts

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

Task II: Define Test Req'ts

- 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 3: Test Requirements

Table C:

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

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

Task II: Define Test Req'ts

- 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 -> FF	3
`remove`	C1 C2 C3 C4	{TTTT, FTTT, TFTT, TTFT, TTTF}	FTTT	FTTT -> FFTT	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…

Task III: Automate Tests

- A merely competent tester…
  would not test preconditions

Task III: Automate Tests

- A merely competent tester…
  would not test preconditions
  All specified behaviors have been tested!

Task III: Automate Tests

- A merely competent tester…
  would not test preconditions
  All specified behaviors have been tested!

- A good tester…

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 …

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 …

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?

ROAR

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	ChID	Characteristic	Covered By

Work...

Task I: Determine Characteristics

Cycle back to add another exception – Table A revised

Method	Params	Returns	Values	Exception	ChID	Characteristic	Covered By
`hasNext`	state	boolean	true,false		C1	More values	
				Concurrent Modification			C5
`next`	state	E	E, null		C2	Returns non-null	
				NoSuchElement			C1
				Concurrent Modification			C5
`remove`	state			Unsupported	C3	remove() supported	
				IllegalState	C4	remove() constraint sat	
				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...

ROAR

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	4
`remove`	C1 C2 C3 C4 C5	{**TTTTT**, FTTTT, TFTTT, TTFTT, TTTFT, TTTTF}	FTTTT	FTTTT -> F**F**TTT	6

Task III: Automate Tests

Test Availability

All tests are available on the Moodle

For Next Time

- Review the Reading
- Review this Lecture
- Come to Class

Are there any questions?