CS409
Software Testing

TAN, Shin Hwei
陈馨慧
Southern University of Science and Technology

Slides adapted from Introduction to Software Testing, Edition 2 (Ch 6)

Administrative Info

- MP1 due on 10 October, 11.59pm. Late submission will get 0!

*If you have question on MP1, post it or read the discussion on https://github.com/orgs/cs409-software-testing2020/teams/allstudents

Q&A for MP1: Post your questions here

stan6
15 hours ago

1. There is a example test in the Android app that you uploaded on the Github. Is this test a example for task vi?

> task vi. Add one JUnit test for each arithmetic operation ("+","-","*","/") (4 points)

```
@Test
public void addition_isCorrect() {
        assertEquals(4, 2 + 2);
}
```

Answer: The test is not an example for task VI. It is a test added by the author of the tutorial. You should create your own test (your test should be different) for task VI.

2. Is the task VII "implement the logic of "." button" the replication of task IV,V?

> iv.Add little code to make the test pass in Step iii. Commit this code to GitHub repository with the commit message "Initial code for passing TDD test". (2 points)
> v.Refactor the code for adding the functionality to support entering decimal points. In this step, you should remove any code duplication. (2 points)
> vii.Implement the logic for the decimal point button "." (4 points)

Answer: No. Task VII is not replication of task IV and V. Task IV should have simple code that makes test in Step III passing and this task requires a commit to the repository. Task V clearly states that there should be a refactoring that removes duplication. Task VII requires implementing the complete logic of "." button.

Reply...

Android / Android Studio Problems

- Android常见Errors

https://blog.csdn.net/qq_16092901/article/details/63076916

- Android Studio常见问题以及解决方式

https://blog.csdn.net/demon_zero/article/details/52535855

ISP lab

- ISP-lab in GitHub Classroom:
- https://classroom.github.com/g/tCTOdiKH

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

Base Choice Coverage (BCC) : A base choice block is chosen for each characteristic, and a base test is formed by using the base choice for each characteristic. Subsequent tests are chosen by holding all but one base choice constant and using each non-base choice in each other characteristic.

Base Choice Notes

- The base test must be feasible
- Happy path tests often make good base choices

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}	

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

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 competent tester would stop there

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 violates 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

Write JUnit tests for the TR

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

ID	Characteristic
C1	More values
C2	Returns non-null object
C3	remove() supported
C4	remove() constraint satisfied
C5	Concurrent Modification

All tests are in:

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