Life is great
VIA University College

Software Development with UML and Java 2

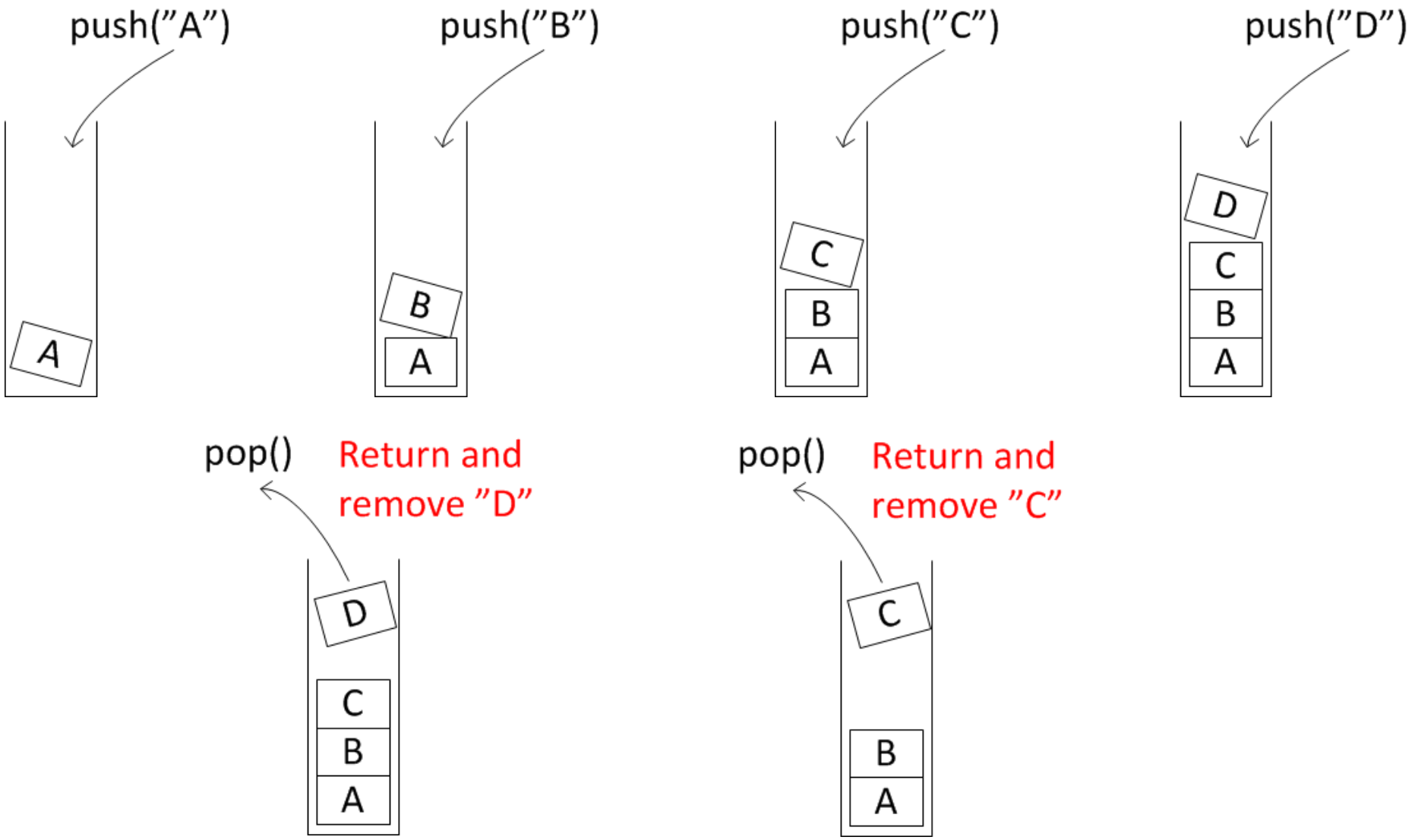
Learning Objectives

- Understand Software Testing
- Write and run JUnit Test

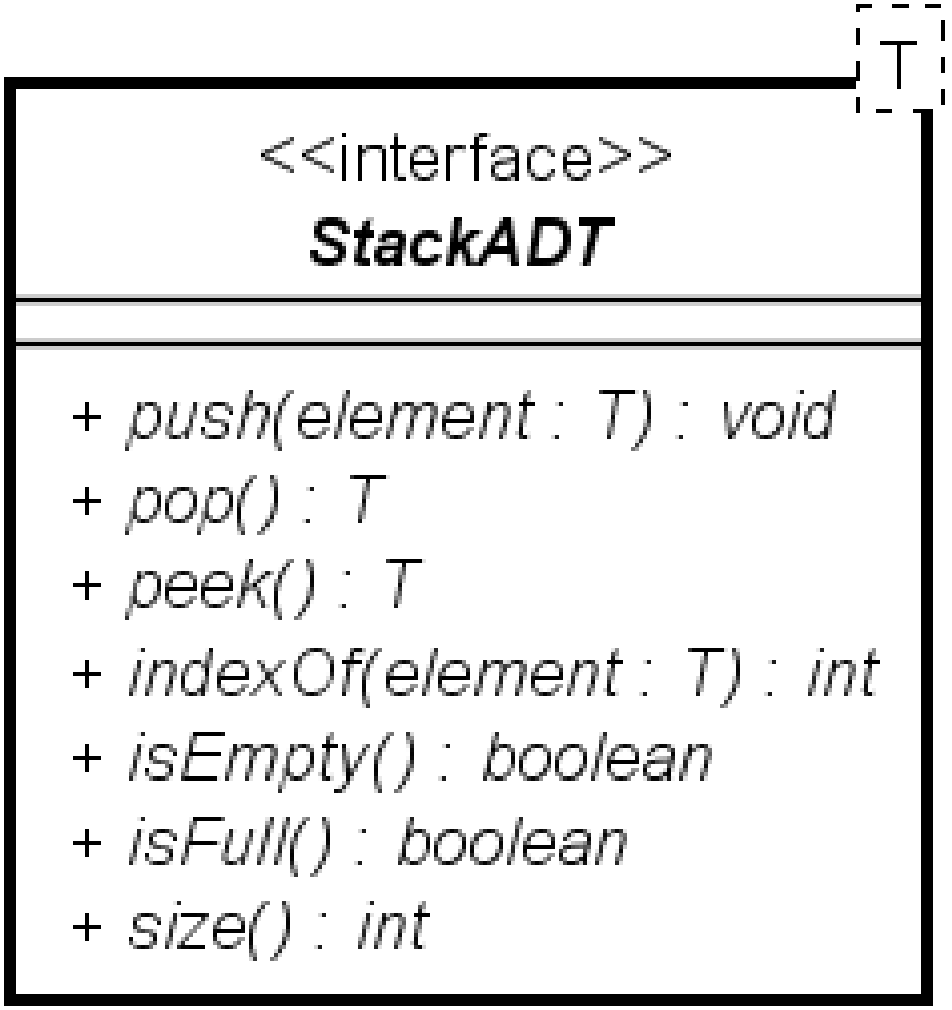
Software Testing

- Software Testing. Tutorial 1- 57. Videos.
  - http://www.guru99.com/software-testing.html
- 100 types of software testing you never knew existed
  - http://www.guru99.com/types-of-software-testing.html
- Overall testing types
  - Unit test
  - Integration test
  - System test
  - Acceptance test

Example: Stack - How does a stack work?



Example: Stack Interface



Example: ArrayStack specification

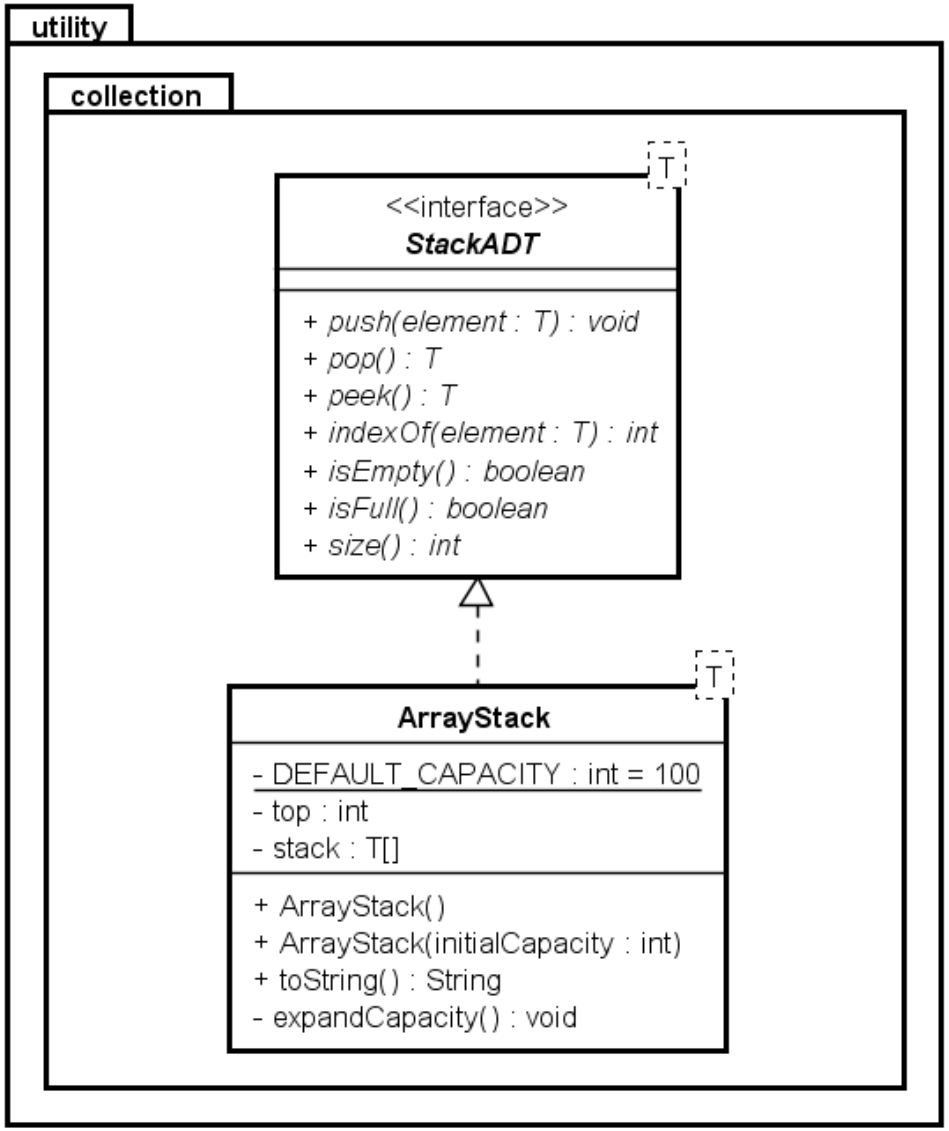
Specification for interface

- http://ict-engineering.dk/javadoc/Collections/utility/collection/StackADT.html

Further specifications for implementation

- The stack may contain `null` elements
- Duplicate elements are allowed
- A stack is never full
- Default capacity is 100 (initial capacity when calling the zero-args constructor)
- After trying to add more elements when the size is equal to the capacity, a new array with twice the capacity is created
- `toString()` method return a string with the elements separated with comas and encapsulated in a set of curly braces. Top element first, example: `"{C, B, A}"`

ArrayStack


utility
collection
T
<<interface>>
StackADT
+ push(element : T) : void
+ pop() : T
+ peek() : T
+ indexOf(element : T) : int
+ isEmpty() : boolean
+ isFull() : boolean
+ size() : int
T
ArrayStack
- DEFAULT_CAPACITY : int = 100
- top : int
- stack : T[]
+ ArrayStack()
+ ArrayStack(initialCapacity : int)
+ toString() : String
- expandCapacity() : void

Unit testing example: ArrayStack

How can we be sure that the ArrayStack implementation follows the specification / javadoc?

- Debugging by adding a lot of print statements in all methods to see if the output is as expected
- A main method creating one or more Stack objects
  - Calling all methods (also when the stack is empty)
  - Pushing, finding (indexOf), peeking and popping `null` elements
  - Trying to force it to throw exceptions (the right type of exceptions)
  - Pushing elements until the array has been resized and then popping all
  - Finding an element and calling indexOf with a value not in the Stack
  - …
- JUnit test
  - Test methods in a separate class/module
  - Each test should run independently of every other test
  - Each test should run at any time, in any order

How to make a Unit test

The purpose is not to show that your program is working but to force the program to fail the test (and then fix it)

- Black-box testing (from a users point of view)
  - Follow requirements and specifications (all cases and exceptions)
  - Equivalence partitioning and Boundary value analysis (video)
- White-box testing (from a programmers point of view)
  - Follow the implementation (all methods, selections and exceptions)
  - Branch testing and Condition/decision coverage

Specifications for method pop ( )

utility.collection

Interface StackADT<T>

Type Parameters:

`T - the data type of elements in the collection`

`public interface StackADT<T>`

StackADT defines the interface to a stack collection - the abstract data type: Stack. The Stack should allow duplicate elements and could allow `null` elements.

Version:

`1.2, 25/1/2016, (Version 1.0, 8/12/2008 by Lewis and Chase)`

Author:

`Steffen Vissing Andersen`

pop

`T pop()`

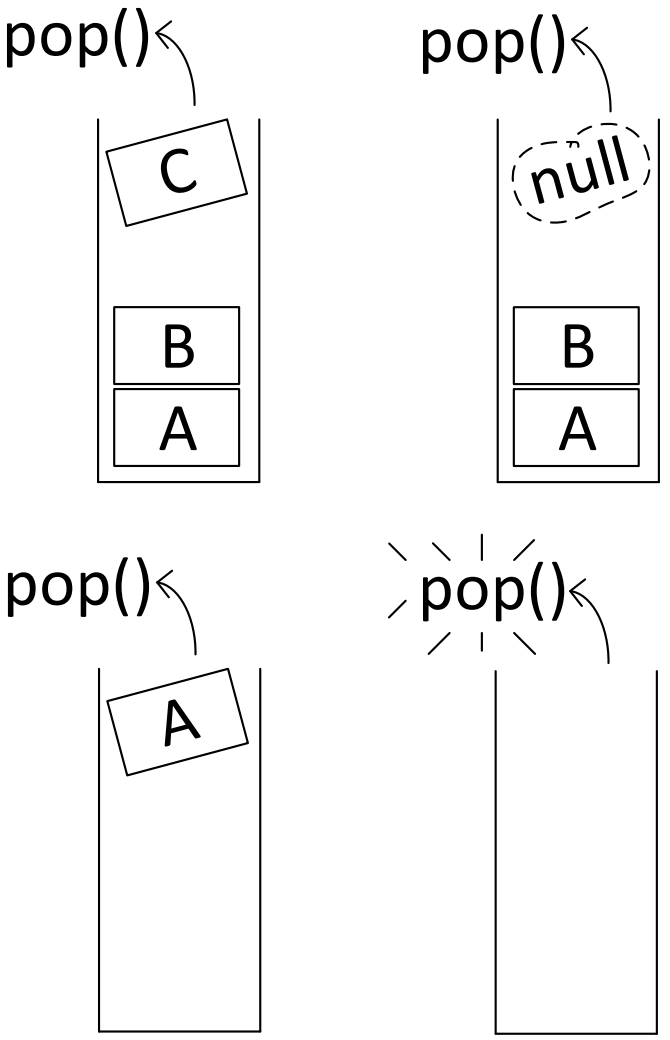
Removes and returns the top element from this stack.

Returns:

`a reference to the element removed from the top of the stack`

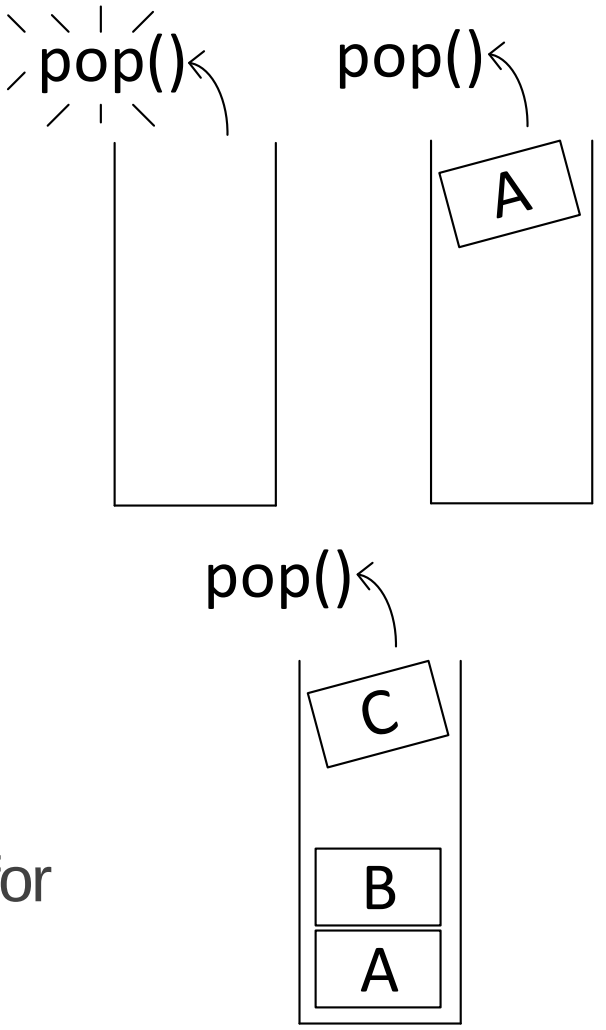
Throws:

`IllegalStateException - if the stack is empty`



Black-box testing method `pop()`

- Legal stack size before calling pop()
  - size ={1, 2, 3, …}
  - Lower limit: size = 1, no upper limit
- Boundary value analysis
  - size=0: should throw IllegalStateException
  - size=1: should pass
- Equivalence partitioning
  - Test for e.g. size=3: should pass
  - If passed, then it would probably also pass for size=2 and size={4, 5, 6, …}



White-box testing method `indexOf`

```
public int indexOf(T element)
{
    if (element == null)
    {
        for (int i = top - 1; i >= 0; i--)
        {
            if (element == (stack[i]))
            {
                return top - 1 - i;
            }
        }

    }
    else
    {
        for (int i = top - 1; i >= 0; i--)
        {
            if (element.equals(stack[i]))
            {
                return top - 1 - i;
            }
        }
    }
    return -1;
}
```

- Test case: element = null
- Test cases: top = 0, 1 and top > 1
- Test case: element is in stack
- Test case: element not null
- …
- Test case: element is not in stack

White-box testing method `indexOf`

- Test case #1: `top = 0` and `element = null`
- Test case #2: `top = 1`, `element = null` and `element` is in stack
- Test case #3: `top = 1`, `element = null` and `element` is not in stack
- Test case #4: `top = 5` (as example), `element = null` and `element` is in stack
- Test case #5: `top = 5` (as example), `element = null` and `element` is not in stack
- Test case #6: `top = 0` and `element` is not `null`
- Test case #7: `top = 1`, `element` is not `null`, `element` is in stack
- Test case #8: `top = 1`, `element` is not `null`, `element` is not in stack
- Test case #9: `top = 5` (as example), `element` is not `null` and `element` is in stack
- Test case #10: `top = 5` (as example), `element` is not `null` and `element` is not in stack

Gray-box testing

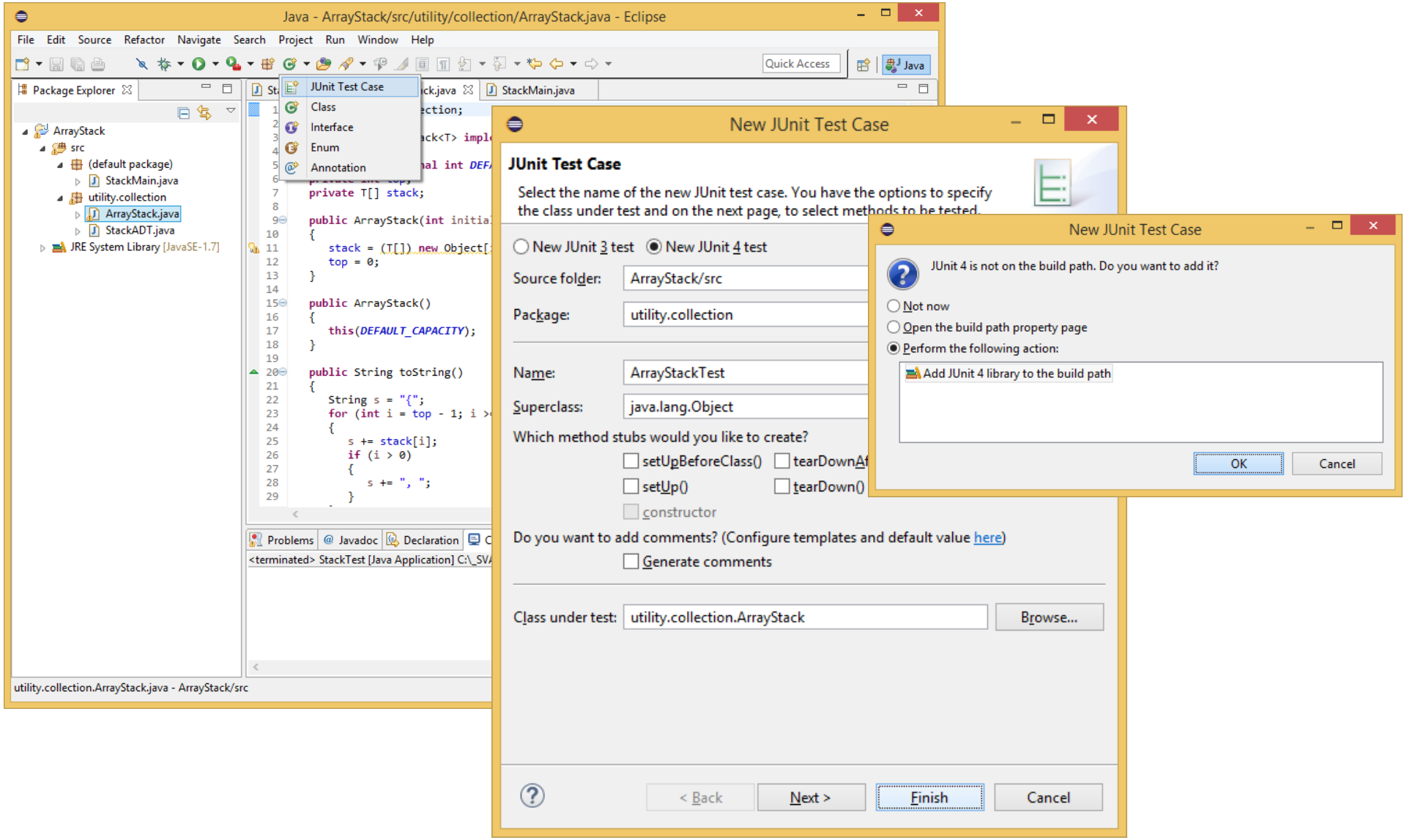
- Sometimes testing is a combination of black-box and white-box testing
- E.g. calling `push` in `ArrayStack` if the array is full
  - The elements will be copied to a larger array (this should be tested)

Unit testing with JUnit

- JUnit testing (JUnit 4)
  - See package `org.junit` (javadoc)
  - JUnit is integrated in Eclipse
- Do one of the following (not both)
  - `import static org.junit.Assert.*;`
  - `public class MyTest extends org.junit.Assert`
- and add various test methods using annotations
  - `@Test, @Before, @After,` …and some other
- How it works

```
setUp()
    testMethod1()
tearDown()
setUp()
    testMethod2()
tearDown()
//...
```

JUnit test in Eclipse



JUnit example: for ArrayStack

```
package test;

import utility.collection.*;

import static org.junit.Assert.*;
import org.junit.*;

public class ArrayStackTest
{
   private StackADT<String> stack;

   @Before
   public void setUp() throws Exception
   {
      stack = new ArrayStack<>();
   }

   @After
   public void tearDown() throws Exception
   {
      // nothing
   }
```

JUnit example: for ArrayStack (testing `push()`)

```
@Test
public void testPushAndPeekAFew()
{
   stack.push("$A$");
   assertEquals(1, stack.size());
   assertEquals("$A$", stack.peek());

   stack.push("$B$");
   assertEquals(2, stack.size());
   assertEquals("$B$", stack.peek());

   //...
   try
   {
      stack.push(null);
      assertEquals(5, stack.size());
      assertEquals(null, stack.peek());
     //...
   }
   catch (IllegalArgumentException e)
   {
     // OK
   }
   // ...
}
```

JUnit example: for ArrayStack (testing `pop()`)

```
@Test
public void testPushAndPopAFew()
{
  stack.push("$A$");
  assertEquals("$A$", stack.pop());
  stack.push("$B$");
  stack.push("$D$");
  assertEquals("$D$", stack.pop());
  assertEquals("$B$", stack.pop());
  try
  {
    stack.push(null);
    assertEquals(null, stack.pop());
    stack.push(null);
    stack.push("$A$");
    assertEquals("$A$", stack.pop());
    assertEquals(null, stack.pop());
  }
  catch (IllegalArgumentException e)
  {
    // OK
  }
}
```

```
@Test(expected =
 IllegalStateException.class)
public void testPopExcepion()
{
  stack.pop();
}
```

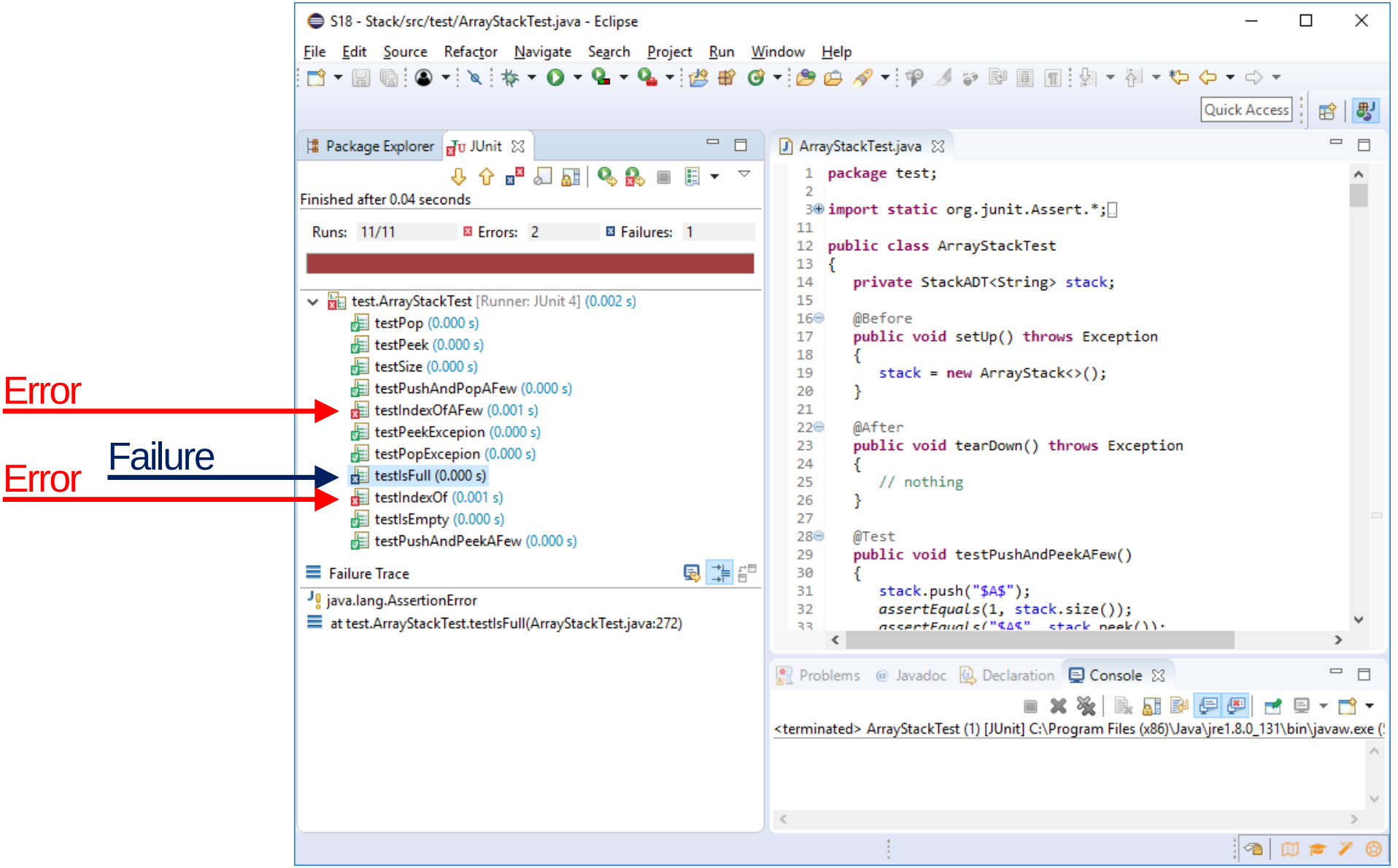
Method `testPopExcepion()`:

- Pop when size = 1 and when size > 1
- Both cases when popped element is not `null` and when element is `null`
- Throwing correct type of exception if `null` elements are not allowed

Method `testPopExcepion()`:

- Throwing correct type of exception when trying to pop when size = 0

JUnit test result: for ArrayStack



A few testing types

6. Ad-hoc Testing
20. Branch Testing
22. Black box Testing
38. Domain Testing
39. Error-Handling Testing
46. Functional Testing
49. Gray Box Testing
51. GUI software Testing
62. Loop Testing
72. Pair Testing
75. Path Testing
76. Penetration Testing
87. Statement Testing
92. Stress Testing
99. Unit Testing
104. White box Testing

What about test cases for SEP?

How to Write Test Cases:
Step by Step Guide with Examples
(http://www.guru99.com/test-case.html)