Chulalongkorn University	Name ______________________
International School of Engineering	Student ID __________________
Department of Computer Engineering	Station No. __________________
2140105 Computer Programming Lab.	Date ________________________

Lab 4 – Unit Testing

Objectives:

- Study concept of unit test.
- Practice development under JUnit framework.

Test-Driven Development

Test-driven development is an approach to program software. This methodology builds program towards testing and simplicity.

- Before writing any code, build an automated-test (unit test) that fails.
- You write only as much code as required for passing the test.
- If you want more functionality, write the test first and then implement the code to make the test passed.
- Any change to the previous code must pass the test.
- All unit tests have to run successfully to guarantee that the program run correctly due to the API specification.

Unit Testing

Unit testing is a procedure used to validate the behavior of a distinct *unit of work* (a single method, in Java application, but not always). A unit may be an individual program, function, procedure, etc. in a procedural design, while may be a class in object-oriented design.

Unit tests usually focus on whether a method is following the terms of its API contract or specification.

JUnit

JUnit is a unit testing framework for the Java programming language created by Kent Beck and Erich Gamma.

The Eclipse's JDT (Java Development Toolkit) tools include a plug-in that integrates JUnit into the Java IDE. The JUnit plug-in allows you to define regression tests for your code and run them from the Java IDE.

To enable the Java IDE to run JUnit, you need to add JUnit library into your project as following steps:

Adding JUnit library into the project.

1. In the explore view, right-click the project name to add the JUnit runtime library (in this case will be **lab4**) and select "**Properties**".

2. The Properties dialog will show up as shown in Figure 1. Click on "**Java Build Path**", and "**Libraries**" Libraries and "**Add Library…"** Add Library... .

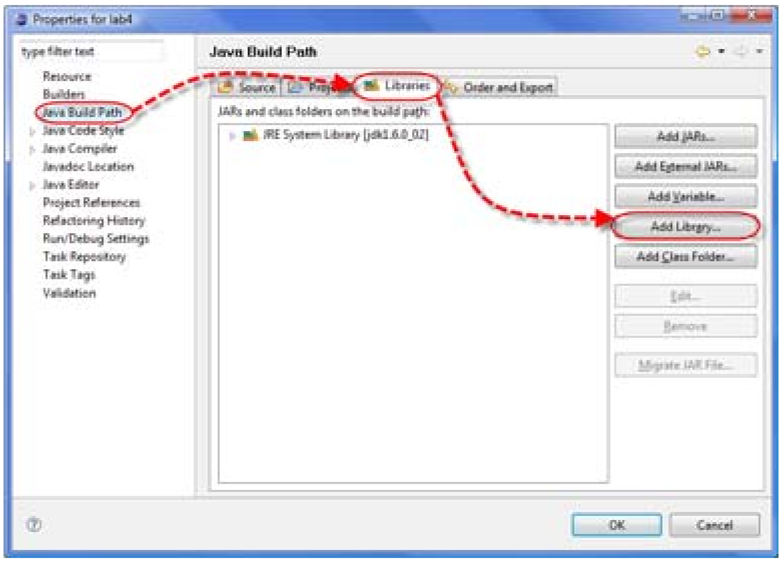


Figure 1 Adding JUnit

3. In the "**Add Library**" dialog, select "**JUnit**" and click "**Next**" and then select "**JUnit 4**" in the JUnit library version and then click "**Finish**". You will see the result as in Figure 2. Click "**OK**".

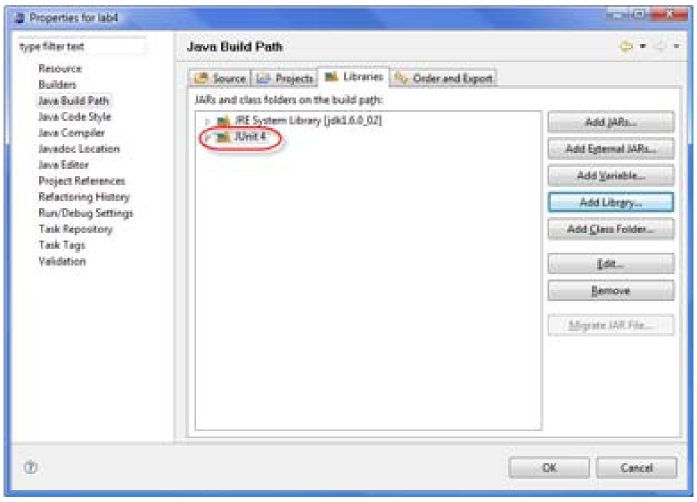


Figure 2 After JUnit library is added

4. If you have done all previous steps correctly, you will see that the JUnit library will be added into your project in Explorer view as shown in Figure 3.

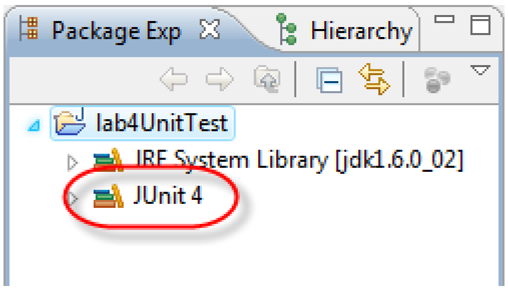


Figure 3 JUnit library is added in Java Project

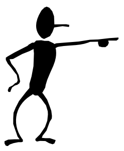
Now your project is ready for running JUnit.

We will introduce the concept of Test-Driven Development (TDD). In TDD, you write the test for your program first and then implement the program according to the test. The test consists of test cases. A test case is a set of conditions or variables under which a tester will determine that the requirement is satisfied.

The cycle in the development will be:

1. Create a test case,
2. Implement the program,
3. Test it,
4. Repeat 2 – 3 until the test is passed.
5. Add more tests to satisfy the requirements, and repeat 2 – 3 – 4 – 5, until all tests are passed.

Let's start the exercise



Your turn

Complete the following steps.

1. To begin our tutorial, **import `IntegerSet.java` into your project**.
2. In the package explore view, click on "**`IntegerSet.java`**."
3. Create new JUnit test case from menu **File > New > JUnit Test Case** as shown in Figure 4.

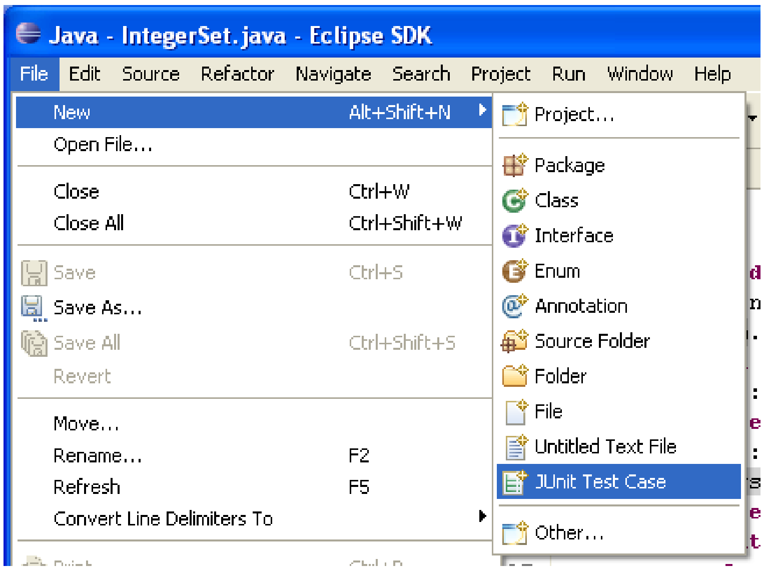


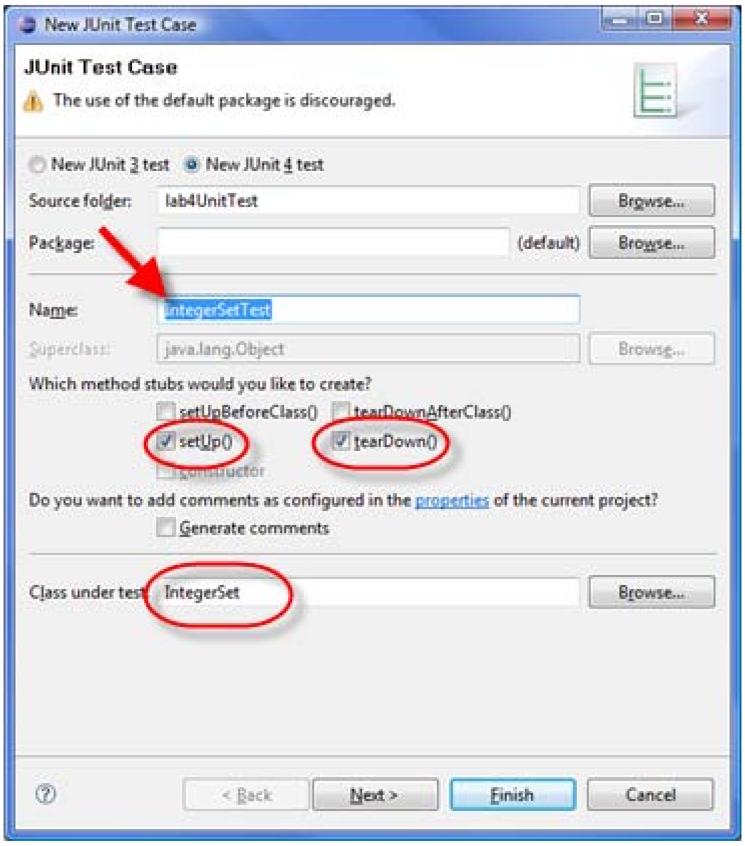
Figure 4 Creating a new JUnit Test Case

4. A new JUnit Test Case dialog wizard will pop-up. Check that in the "**Name**:" textbox has "**`IntegerSetTest`**" and "**Class under test**:" is **`IntegerSet`**. Make sure that **`setup()`** and **`teardown()`** boxes are selected as shown in Figure 5 and then click **Next**.
5. Next dialog will show up. Click all methods in the class **`IntegerSet`** except the two constructors as shown in Figure 6 then click **Finish**.

6. Verify that your JUnit test case has been added to your project by looking in the package explorer view. You will see that the file "**`IntergerSetTest.java`**" has been added to the project as shown in Figure 7. A JUnit test case is a class that extends from **`junit.framework.TestCase`** class. The wizard will create test cases for you by adding "**`test`**" in front of the name of your methods to be tested. For example, "**`testIsEmpty()`**" is added for testing "**`isEmpty().`**"

   In JUnit 4, a test case is not required to have "**`test`**" as prefix in the method name. You can make a method to be a test case by annotate a method with **`@Test`**. For example, in **`IntegerSetTest.java`**, you can see that there is **`@Test`** before the method **`testIsEmpty()`**.

7. Select menu **Run > Run As > JUnit Test** to run the JUnit test cases. The test will run and fail because we have not implemented our class. Figure 8 shows the test result.



New JUnit Test Case

JUnit Test Case

The use of the default package is discouraged.

New JUnit 3 test [✓ New JUnit 4 test]

Source folder: lab4UnitTest

Package: (default)

Name: IntegerSetTest

Superclass: java.lang.Object

Which method stubs would you like to create?

setUpBeforeClass() tearDownAfterClass()

[✓ setUp()] [✓ tearDown()]

constructor

Do you want to add comments as configured in the properties of the current project?

Generate comments

Class under test: IntegerSet

Figure 5 New JUnit Test Case Dialog

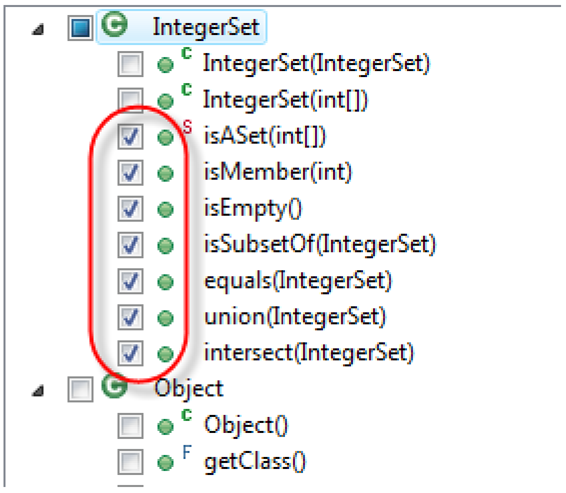


Figure 6 Selecting Test Methods

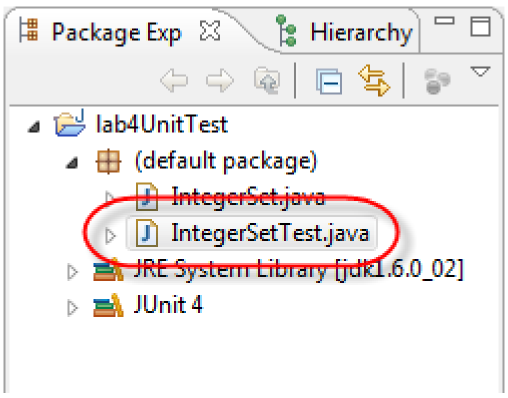
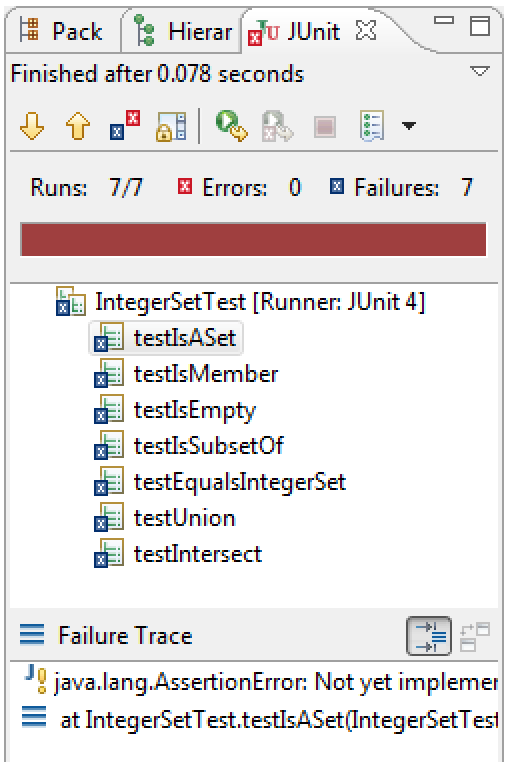


Figure 7 IntegerSetTest.java in Package Explorer view



```
Finished after 0.078 seconds
Runs: 7/7   Errors: 0   Failures: 7
IntegerSetTest [Runner: JUnit 4]
    testIsASet
    testIsMember
    testIsEmpty
    testIsSubsetOf
    testEqualsIntegerSet
    testUnion
    testIntersect
java.lang.AssertionError: Not yet implemer
at IntegerSetTest.testIsASet(IntegerSetTest
```

Figure 8 Test Result

8. Now we will start our development cycle. First we will modify the **`testIsEmpty()`** that test the method **`isEmpty()`**.

   The concept of test case is that we will create know input and compare with expected output. Here are a few tips for writing effective tests:

   - Test for boundary conditions, i.e. check the minimum and maximum indices for arrays and lists. Also check indices that are just out of range.
   - Test for illegal inputs to methods.
   - Test for null strings and empty strings. Also test strings containing unexpected whitespace.

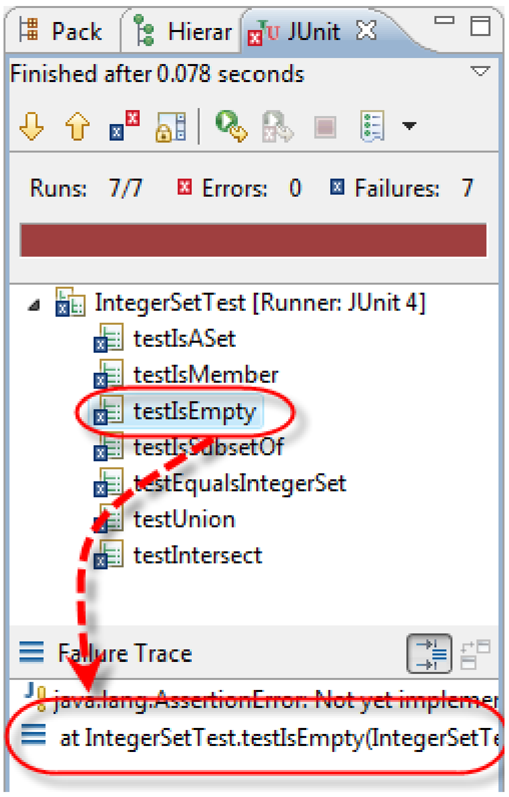
For the method **`isEmpty()`**, we know that if the input is an empty set the result must be **`true`**, and **`false`** otherwise. The statement **`fail("not yet implemented")`** causes this test to fail. We will modify this test (method **`testIsEmpty()`** by remove the line **`fail("not yet implemented")`** and replace it with the following instructions:

```
    int[] a = { };
    int[] b = { 1, 2, 3};
    IntegerSet emptySet = new IntegerSet(a);
    IntegerSet aSet = new IntegerSet(b);
    assertTrue(emptySet.isEmpty());
    assertFalse(aSet.isEmpty());
```

The first four lines create two instances of **`IntegerSet`**; **`emptySet`** and **`aSet`**. **`emptySet`** represents an empty integer set, and **`aSet`** represents an un-empty set.

- **`assertTrue()`** evaluates its parameter, if **`true`**, the test is passed. The test is fail if the parameter evaluates to **`false`**.
- **`assertFalse()`**, on the other hand, passes the test if its parameters is evaluated to **`false`**, and fail otherwise.

Run the test, it must fail because we have not yet implemented **`isEmpty()`**. To see where the location of the failure is, click **`testIsEmpty()`** in the JUnit result view as shown in Figure 9.



Finished after 0.078 seconds

Runs: 7/7 Errors: 0 Failures: 7

IntegerSetTest [Runner: JUnit 4]
testIsASet
testIsMember
testIsEmpty
testIsSubsetOf
testEqualsIntegerSet
testUnion
testIntersect

Failure Trace
java.lang.AssertionError: Not yet implemer
at IntegerSetTest.testIsEmpty(IntegerSetTe

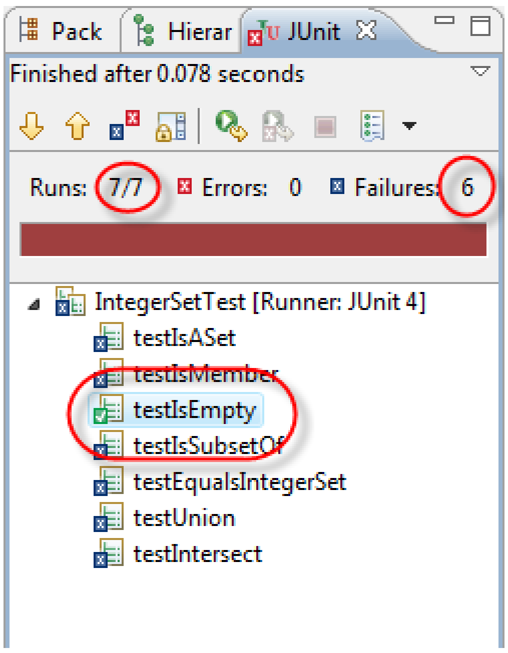
Figure 9 Location of Failure

9. To implement **`isEmpty()`**, select file **`IntegerSet.java`**, go to method **`isEmpty()`** and add the following statements:

```
    if (this.element.length == 0)
        return true;
    return false;
```

A set is an empty set if the element array has size 0.

Click to run the last test. You will see that the test passes as shown in Figure 10. The number of failures reduces from 7 to 6.



Finished after 0.078 seconds

Runs: 7/7 Errors: 0 Failures: 6

IntegerSetTest [Runner: JUnit 4]
testIsASet
testIsMember
testIsEmpty
testIsSubsetOf
testEqualsIntegerSet
testUnion
testIntersect

Figure 10 Passed Test Case

The following are some assertion statements that are used to test for the conditions and expected results:

- **`assertEquals(boolean expected, boolean actual)`**
  Asserts that two booleans are equal.
- **`assertEquals(char expected, char actual)`**
  Asserts that two chars are equal.
- **`assertEquals(double expected, double actual, double delta)`**
  Asserts that two doubles are equal concerning a delta.
- **`assertEquals(float expected, float actual, float delta)`**
  Asserts that two floats are equal concerning a delta.
- **`assertEquals(int expected, int actual)`**
  Asserts that two ints are equal.
- **`assertEquals(long expected, long actual)`**
  Asserts that two longs are equal.
- **`assertEquals(java.lang.Object expected, java.lang.Object actual)`**
  Asserts that two objects are equal.
- **`assertEquals(short expected, short actual)`**
  Asserts that two shorts are equal.
- **`assertEquals(java.lang.String expected, java.lang.String actual)`**
  Asserts that two Strings are equal.
- **`assertFalse(boolean condition)`**
  Asserts that a condition is false.
- **`assertNotNull(java.lang.Object object)`**
  Asserts that an object isn't null.
- **`assertNotSame(java.lang.Object expected, java.lang.Object actual)`**
  Asserts that two objects do not refer to the same object.

- **`assertNull(java.lang.Object object)`**
  Asserts that an object is null.
- **`assertSame(java.lang.Object expected, java.lang.Object actual)`**
  Asserts that two objects refer to the same object.
- **`assertTrue(boolean condition)`**
  Asserts that a condition is true.
- **`fail(java.lang.String message)`**
  Fails a test with the given message.

For the complete lists of assertions go to http://junit.sourceforge.net/javadoc/junit/framework/Assert.html

10. Let's continue with implementing the method **`isASet().`** First we need to change modify the **`testIsASet()`** by removing the **`fail()`** statement and replacing with the test cases.

    If you want to write a test similar with the one you have already written, write a Fixture instead.

    Fixture

    If you have two or more tests that operate on the same or similar set of objects, these set of objects are called a test fixture. Instead of writing a lot of object declaration statements in you test methods, you write a shared test fixture that will be used among those methods. You can use the same test fixture for different tests.

    To add test fixture:

    - Add a field for each part of the fixture
    - Put the fields initialization statements in the method **`setup()`**. The annotation **`@Before`** force the method **`setup()`** to be executed before each test method annotated by **`@Test`**. If you want any method to be executed before each test, annotate with **`@Before`**.
    - Reinitialize the variables back to their original states in the method **`tearDown()`**. The annotation **`@After`** force the method **`tearDown()`** to be executed after each test method annotated by **`@Test`**. If you want any method to be executed after each test, annotate with **`@After`**.
    - We will use the default **`setup()`** and **`tearDown()`** methods since we do not have any extra method.
    - When the test start the sequence will be

    ```
    @Before method(s)          setup()
    @Test method #1            testIsASet()
    @After method(s)           tearDown()
    @Before method(s)          setup()
    @Test method #2            testISEmpty()
    @After methods()           tearDown()
     . . .
    ```

11. Remove the **`int`** arrays, a and b, from methods **`isEmpty()`**.
12. Define the fields

```
    int[] empty, a, b;
    IntegerSet emptySet, setA, setB;
```

13. In method **`setup()`** add the following statements:

```
    empty = new int[] { };
    emptySet = new IntegerSet(empty);
    a = new int[] { 1, 2, 3 };
    setA = new IntegerSet(a);
    b = new int[] { 1, 1, 2 };
    setB = new IntegerSet(b);
```

14. Add the following statements into method **`testIsAset()`**.

```
    assertTrue(IntegerSet.isASet(a));
    assertFalse(IntegerSet.isASet(b));
    assertFalse(IntegerSet.isASet(null));
```

15. Run the test, still fail.
16. Modify the **`isASet()`** with the following statements and run the test.

```
    public static boolean isASet(int[] data) {
          if (data == null) return false;
          java.util.Arrays.sort(data);
          for (int i = 0; i < data.length - 1; i++)
                if (data[i] == data[i+1]) return false;
          return true;
    }
```

17. Modify **`testEqualsIntegerSet().`**

```
    public void testEqualsIntegerSet() {
          int[] a = { 1 };
          int[] b = { 1 };
          int[] c = { 2 };
          int[] d = { 1, 2 };
          int[] e = { 2, 1 };

          IntegerSet setA = new IntegerSet(a);
          IntegerSet setB = new IntegerSet(b);
          IntegerSet setC = new IntegerSet(c);
          IntegerSet setD = new IntegerSet(d);
          IntegerSet setE = new IntegerSet(e);

          assertTrue(setA.equals(setB));
```

```
        assertTrue(setB.equals(setA));
        assertFalse(setA.equals(setC));
        assertFalse(setA.equals(setD));
        assertTrue(setD.equals(setE));
    }
```

18. Implement method **`equals()`** in **`IntegerSet.java`**

```
    public boolean equals(IntegerSet set) {
        if (set == null) return false;
        if (this.element.length != set.element.length)
            return false;
        for (int i = 0; i < element.length; i++) {
            if (element[i] != set.element[i])
                return false;
        }
        return true;
    }

    public boolean equals(Object o) {
        IntegerSet that = (IntegerSet) o;
        return this.equals(that);
    }
```

19. Run **`IntegerSetTest.java`** as JUnit test again, it should pass.
20. Modify the following test cases.

```
    public void testIsSubsetOf() {
        int[] a = {};
        int[] b = {1};
        int[] c = {1, 2};
        IntegerSet setA = new IntegerSet(a);
        IntegerSet setB = new IntegerSet(b);
        IntegerSet setC = new IntegerSet(c);
        assertTrue(setA.isSubsetOf(setB));
        assertTrue(setA.isSubsetOf(setC));
        assertTrue(setB.isSubsetOf(setC));
        assertFalse(setB.isSubsetOf(setA));
        assertFalse(setC.isSubsetOf(setB));
        assertTrue(setC.isSubsetOf(setC));
    }
```

21. Implement the method **`isSubsetOf()`** to make the test pass.

22. Modify **`testIsMember()`**, **`testUnion()`** and **`testIntersection()`** by adding some statements and assertion statements as appropriated. The assertions test should test for the combination of correct and incorrect results.
23. Implement the corresponding methods in **`IntegerSet.java`** and re- run the test until all the tests have passed.

You can find the specification for each method in the comment before the method definition or you can look at the API document in the following pages.

References

Vincent Massol, *JUnit in Action*, Manning, 2003.

Johannes Link and Peter Frohlich, *Unit Testing in Java*, Morgan Kaufmann Publishers, 2003.

resource

Class IntegerSet

```
java.lang.Object
  └ IntegerSet
```

```
public class IntegerSet
extends java.lang.Object
```

This IntegerSet is a representation of a set of unordered distinct integers. An array will be used to represent this set.

Author:

USER

Constructor Summary

`IntegerSet(int[] array)` Create an integer set from the array provided.
`IntegerSet(IntegerSet set)` Create a new IntegerSet from a set provided.

Method Summary

`boolean`	`equals(IntegerSet set)` Check whether this IntegerSet is equal to the set provided.
`IntegerSet`	`intersect(IntegerSet set)` Intersect this IntegerSet with the set provided, the result is a new IntegerSet which all of it elements are member of this IntegerSet and the provided set.
`static boolean`	`isASet(int[] data)` This method will check that the array provided is considered a set or not.
`boolean`	`isEmpty()` Check that this set is empty or not

`boolean`	`isMember(int number)` This method will check that the number is a member of this set or not.
`boolean`	`isSubsetOf(IntegerSet set)` Check whether this IntegerSet is a subset of the set provided.
`IntegerSet`	`union(IntegerSet set)` Union this IntegerSet with the set provided, the result is a new IntegerSet which all of its elements are member of this IntegerSet or the provided set.

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

IntegerSet

`public` **`IntegerSet`**`(IntegerSet set)`

Create a new IntegerSet from a set provided. The new set will have the same elements as the set provided.

Parameters:

`set` -

IntegerSet

`public` **`IntegerSet`**`(int[] array)`

Create an integer set from the array provided. If the array has many element with the same value, initialize the set's element with only the distinct array's elements.

Parameters:

`array` - original array

Method Detail

isASet

```
public static boolean isASet(int[] data)
```

This method will check that the array provided is considered a set or not. An array is considered a set if all of its elements are distinct.

Returns:

true if this array is a set, and false otherwise.

---

isMember

```
public boolean isMember(int number)
```

This method will check that the number is a member of this set or not.

Parameters:

`number` - an integer to check

Returns:

true if the number is a member of this set, and false otherwise.

---

isEmpty

```
public boolean isEmpty()
```

Check that this set is empty or not

Returns:

true if this set is empty, and false otherwise

---

isSubsetOf

```
public boolean isSubsetOf(IntegerSet set)
```

Check whether this IntegerSet is a subset of the set provided. Set A is a subset of B if all elements in A are in B.

Parameters:

`set` - the set provided.

Returns:

true if this IntegerSet is a subset of the set provided, and false otherwise.

---

equals

```
public boolean equals(IntegerSet set)
```

Check whether this IntegerSet is equal to the set provided. Two set is considered to be equal if they are subset of each other.

Parameters:

`set` - the set provided.

Returns:

true if both set are equal, and false otherwise.

---

union

```
public IntegerSet union(IntegerSet set)
```

Union this IntegerSet with the set provided, the result is a new IntegerSet which all of its elements are member of this IntegerSet or the provided set.

Parameters:

`set` - the set provided.

Returns:

new IntegerSet that is the result from union

---

intersect

```
public IntegerSet intersect(IntegerSet set)
```

Intersect this IntegerSet with the set provided, the result is a new IntegerSet which all of it elements are member of this IntegerSet and the provided set.

Parameters:

`set` - the set provided

Returns:

new IntegerSet that is the result from intersection

Chulalongkorn University		Name	
International School of Engineering		Student ID.	
Department of Computer Engineering		Station No.	
2140-105 Computer Programming Lab.		Data	

Lab 4 – Unit Test

Task	Description	Result	Note
1	`isSubsetOf`		
2	`testIsMember`		
3	`testUnion`		
4	`testInterSection`		
5	`isMember`		
6	`union`		
7	`intersection`		