OOP Lab 5 Exercises: Arrays, Testing and Using the Java API

Authors: Joshua Ritterman
Ewan Klein
Version: 1.15
Date: 2009-02-09

Contents

Problems?

If any of the following exercises are unclear or seem to contain errors, please don't hesitate to let us know! Just send an email, preferably to both authors.

Preamble

The first couple of exercises are intended to help start you off with testing your code. Having software check itself when it is built has become very popular in the software engineering world. This helps to cut down on Quality Assurance resources and produces better quality releases. In Java there is a framework to do this called *JUnit*. For the sake of simplicity, we will be using a version of JUnit (namely version 3.8.*x*, where *x* is 1 or 2.) where the test file contains a class that extends `TestCase` and implements a number of methods that must start with `'test'` in the name. The body of a test method can contain any valid Java code and typically makes use of the statement `assertEquals(trueExpectedValue, testValue)`.

If you are interested, you can read a bit more about JUnit here: http://junit.sourceforge.net/junit3.8.1/index.html.

The third exercise gives you an opportunity to play around with a couple of data structures provided by the Java API.

These exercises cover material up to and including Chapter 6 of *Head First Java*. For information on `HashMap`s and on the `split()` method, have a look at these lecture slides from Friday 6th February.

Exercises

Exercise 1: `ArrayShift` and Your First Unit Test

In this exercise, you will need to do two things. First, implement a class `ArrayShift` which carries out simple manipulation of an array, and second, implement a JUnit test `ArrayShiftTest` to check that your code does what it is supposed to do.

Now, `ArrayShift` is going to have only one testable method, which should have the following declaration:

```
public int[] shiftOne(int[] inArray) {
...
}
```

If we pass the argument `inArray` (an array of non-negative `int`s) to `shiftOne()`, it should return a new array `outArray` with the following properties:

1. the length of `outArray` is the same as `inArray`;
2. the first element of `outArray` is `-1`.

3. all but the last element of `inArray` appear in `outArray` and the elements of `inArray` are shifted one to the right in `outArray`,

For example, if `inArray` is of the form:

`{4, 21, 55, 2, 7}`

then `outArray` will be of the form:

`{-1, 4, 21, 55, 2}`.

Testing with JUnit

So let's think now about how we might test this method. Here's a skeleton of the JUnit test file ArrayShiftTest:

```
    import junit.framework.TestCase;

    public class ArrayShiftTest extends TestCase {

            private ArrayShift as;
            private int[] inArray = {4, 6, 0, 3, 4, 5, 4, 4, 6, 2 };

            public final void setUp()  {
                    as = new ArrayShift();
            }

            public final void testShiftOne() {
                    int[] outArray = as.shiftOne(inArray);
                    // Add some statments of the form
                    // assertEquals(testValue, trueExpectedValue)


            }

    }
```

Notice first of all that this is just like the kind of classes you are familiar with, in the sense that we are declaring a new class `ArrayShiftTest` that lives in file called `ArrayShiftTest.java`. However, the first line tells us that we need to import the relevant JUnit library class `TestCase`, while the next line tells us that our new class `ArrayShiftTest` *extends*, i.e., inherits from, `TestCase`. Inside the class, we declare a couple of instance variables. The first of these, namely the variable `as`, has the reference type `ArrayShift`. This is what connects the JUnit test file to the class whose methods we want to test, and presupposes that we can get access to the class defined in a file `ArrayShift.java`

The `setUp()` method provides an opportunity to set the stage for any test methods. Here, we just use it to create an instance of the class `ArrayShift` and assign a reference to that instance as the value of the variable `as`.

Now we can look at the method `testShiftOne()`. This has no return value, and is going to include some tests. For convenience, we declare and initialize a new local variable, `outArray`, which is a reference to the result of calling `ArrayShift`'s `shiftOne()` method on the array `inArray`.

Your task is to complete the body of `testShiftOne()` with some tests that check that `outArray` does indeed have the required properties. To illustrate, we'll write the first test for you. We want to check the first property listed above, namely that input and output of `shiftOne()` have the same length. As we mentioned earlier, we typically do this with a statement of the form `assertEquals(trueExpectedValue, testValue)`. (Note that `assertEquals()` is a method that is defined in the class `TestCase`, and inherited by our new class `ArrayShiftTest`.) So in this case, our test value is going to be the length of `outArray`. What's the 'true expected value'? Well, we said that the output array length should be the same as the input array , so our expected value can be `inArray.length`. Putting this all together, our partially completed method will look as follows:

```
public final void testShiftOne() {
        int[] outArray = as.shiftOne(inArray);
        assertEquals(outArray.inArray, length.length);
        // Add more statments of the form
        // assertEquals(trueExpectedValue, testValue)
}
```

Using this model, go ahead and add some more `assertEquals()` statements that check additional properties of `outArray` in terms of `inArray`.

But here's an important **warning.** In JUnit 3.8.x, you *cannot* directly check two arrays for equality using `assertEquals`. In other words, don't define a new array, say `expectedArray`, which contains exactly the elements you expect, and then try to add the statement `assertEquals(expectedArray, outArray)` to `testShiftOne()`. This test will *fail* in JUnit 3.8.x! Instead, you have to check equality on individuals elements of the arrays you care about.

Of course, as well as writing the test cases, you also need to implement the class `ArrayShift` and the method `shiftOne`.

In order to run JUnit from the command line on a DICE machine, you need to do something like the following. Assume, as above, that your source code is in the file `ArrayShift.java` and your JUnit test code is in the file `ArrayShiftTest.java`. First, type the following on the terminal command line:

```
export CLASSPATH=$CLASSPATH:/usr/share/java/junit-3.8.2.jar
```

Next, use `javac` as usual to compile both your source code and your test code.

Finally, enter the following command on the terminal:

```
java junit.textui.TestRunner ArrayShiftTest
```

This should run the tests in the compiled class `ArrayShiftTest` and report back whether the tests passed or not.

In order to avoid having to manuallyt enter the command for setting your `CLASSPATH` each time you login, open up the file `.brc` which will be sitting in your home directory on DICE, and just add the line given above. If your are using your own computer, there should be a similar file for declaring 'environment variables' on your platform.

Submit both the `ArrayShift` class and the `ArrayShiftTest` class to Web-CAT.

Sample solutions

`ArrayShift.java`:

```
public class ArrayShift {

        /**
         * Shift all the values of the input array along by one cell.
         * @param inArray The input array.
         * @return The shifted array, whose 0 element has the value -1.
         */
        public int[] shiftOne(int[] inArray) {
                int[] outArray = new int[inArray.length];

                outArray[0] = -1;

                for (int i = 0; i < inArray.length - 1; i++) {
                        outArray[i + 1] = inArray[i];
                }
                return outArray;
        }
}
```

`ArrayShiftTest.java`:

```
import junit.framework.TestCase;

public class ArrayShiftTest extends TestCase {

        private ArrayShift as;
        private int[] testArray = { 4, 6, 0, 3, 4, 5, 4, 4, 6, 2 };
        int[] shifted;

        public final void setUp() {
                as = new ArrayShift();
                shifted = as.shiftOne(testArray);
        }

        public final void testShiftFirst() {
                assertEquals(-1, shifted[0]);
        }

        public final void testShiftLength() {
                assertEquals(testArray.length, shifted.length);
        }

        public final void testShiftElements() {
                for (int i = 0; i < testArray.length - 1; i++) {
                        assertEquals(testArray[i], shifted[i + 1]);
                }
        }

}
```

Exercise 2: `ArrayMult` and Another Unit Test

This exercise is very similar to the preceding one. First, implement a class `ArrayMult` which carries out pointwise multiplication on two arrays of non-negative integers, and second, implement a JUnit test `ArrayMultTest` to check that your code does what it is supposed to do.

`ArrayMult` should define a method `mult()` with the following declaration:

```
public int[] mult(int[] array1, int[] array2) {
...
}
```

The method should be defined so that `array1` and `array2` can be of different lengths. Let `minlen` be the length of the shortest array (or of both arrays if they are the same size), and let `longArray` be whichever is the longest of the two. Then the following properties hold of the output array, `outArray`:

1. The length of `outArray` is the length of `longArray`.
2. For all `i < minlen`, `outArray[i] = array1[i] * array2[i]`.
3. For all `j >= minlen`, `outArray[j] = longArray[j]`.

Implement the `ArrayMult` class with method `mult()` as specified, and implement `ArrayMultTest` by analogy with `ArrayShiftTest`. (If you are not quite sure yet what the test file should look like, ArrayMultTest: is a skeleton to get you started.) Submit both of these to Web-CAT.

Sample solutions

`ArrayMult.java`:

```
public class ArrayMult {

        /**
         * Point-wise multiply two arrays of different lengths.
         * @param array1 First input array.
         * @param array2 Second input array.
         * @return A new array which is as long as the longest input.
         */
        public int[] mult(int[] array1, int[] array2) {
                int[] longArray = array1;
                int[] shortArray = array2;

                if (array2.length > array1.length) {
                        longArray = array2;
                        shortArray = array1;
                }

                int[] outArray = new int[longArray.length];

                for (int i = 0; i < longArray.length; i++) {
                        if (i < shortArray.length) {
                                outArray[i] = array1[i] * array2[i];
                        } else {
                                outArray[i] = longArray[i];
                        }
                }
                return outArray;
        }

}
```

`ArrayMultTest.java`:

```
import junit.framework.TestCase;

public class ArrayMultTest extends TestCase {

        // Declare some test inputs.
        private ArrayMult am;
        private int[] testArray1 = {4, 6, 0, 3, 4, 5, 4, 4, 6, 2 };
        private int[] testArray2 = {6, 0, 3, 4, 5, 4, 4, 0 };
        // an 'identity' array
        private int[] testArray3 = new int[0];

        int[] mult1;
        int[] mult2;
        int[] mult3;

        public void setUp()  {
                am = new ArrayMult();
                mult1 = am.mult(testArray1, testArray2);
                mult2 = am.mult(testArray2, testArray1);
                mult3 = am.mult(testArray1, testArray3);
        }

        public void testMultLength() {

                assertEquals(testArray1.length, mult1.length);
                assertEquals(testArray1.length, mult2.length);
                assertEquals(testArray1.length, mult3.length);

                assertEquals(testArray1[testArray1.length-1], mult1[mult1.length-1]);
        }

        public void testMultValues() {

                for (int i = 0; i < testArray2.length; i++) {
                        assertEquals(testArray1[i] * testArray2[i], mult1[i]);
                }

                for (int i = 0; i < testArray2.length; i++) {
                        assertEquals(testArray1[i] * testArray2[i], mult2[i]);
                }

                for (int i = 0; i < testArray1.length; i++) {
                        assertEquals(testArray1[i], mult3[i]);
                }
        }

}
```

Exercise 3: Using `ArrayList` and `HashMap`

The basic Java data types have worked fairly well so far, but it is time to move beyond what they can do. Last week made a simple class to wrap an array, that if there were was a class that did that but added a lot more functionality to the array (other then filling it with random numbers). Well the Java API provides classes like this, called the Collections classes. Today we are going to look at two of them, the ArrayList and the HashMap. Before we go on, take a look at the Java API docs for these two classes:

- ArrayList
- HashMap

`ArrayList` is a wrapper around an array that allows you to grow the array at run time, so you no longer are required to know the maximum capacity of the array when you declare it.

`HashMap` is a data structure that you may have heard called a 'dictionary' or an 'associative array' in other languages. A `HashMap` is a conceptually simple data structure consisting of a set of keys and values. You might want to think of it as a type of array that you access with keys rather than by indexing.

In this exercise you will be making a simple student 'Directory' application, A `Student` is an object that has a `matriculation number`, `name`, `age`, `mailbox`, `gender` and `department`, and all the basic getters and setters to deal with this set of fields. Here is a table with the initial data:

MN	Name	Age	MailBox	Gnd
s0189034	Peter	17	peter@math	M
s0289125	Michael	21	mike@geo	M
s0378435	Helen	28	helen@phys	F
s0412375	Mary	18	mary@inf	F
s0456782	John	22	john@inf	M
s0355689	Dana	33	dana@ling	F
s0768633	Lee	36	lee@chem	F

The only slightly tricky task is to get the information about the student's department, since this is not given directly in the data above. Your task is to compute this from the student's MailBox. First, build a `HashMap` which encodes the following key-value pairs:

Domain	Department Name
chem	Chemistry
geo	Geosciences
inf	Informatics
ling	Linguistics
math	Mathematics
phys	Physics

Next, you have to parse the value of the student's MailBox attribute; you can use the `String` method `split()` to do this. Finally, you can use the domain part of the MailBox string to look up the department name in the HashMap that you have constructed.

Here is the skeleton of the Roster.java class.

```
import java.util.*;

public class Roster {

    private ArrayList<Student> students = new ArrayList<Student>();

    public void makeRoster() {
            Student peter = new Student();
            peter.setMn("s0189034");
            peter.setName("Peter");
            peter.setAge(17);
            peter.setMbox("peter@math");
            peter.setGender("M");

            Student michael = new Student();
            // complete this object

            // add objects for all the other students

            students.add(peter);
            // add the other students
    }

    public HashMap<String, Student> getMedianStudentsMap() {
            // TODO: Return a HashMap with the student's name as
            // the key and the student record as a value for any
            // student aged over 18 but under 25.
    }

    public Student getStudentByName(String name) {
            // TODO: Return a student record of the student whose
            // name was passed in as argument to the method.
    }

    public String getStudentDepartment(String name) {
            // TODO: Return the department of the student whose
            // name was passed in as argument to the method.
    }

    public ArrayList<Student> getStudentsByAgeAndGender(int minAge, String gender) {
            // TODO: Return an ArrayList of student that is older
            // than the minAge, and of the Gender "M" or "F" given
            // in gender
    }

    public void printRoster() {
            for (Student s : students) {
                    System.out.println("Name: " + s.getName());
            }
    }
}
```

Implement the following methods:

- `getMedianStudentsMap()`,
- `getStudentByName()`,
- `getStudentDepartment()` and
- `getStudentsByAgeAndGender()`.

Now that you know how to make your own unit tests, make a JUnit test to test that the four methods that you wrote are working as you expect. Submit to Web-CAT the files `Student.java`, `Roster.java` and `RosterTest.java`.

Sample solutions

`Student.java`:

```
import java.util.HashMap;

public class Student {
        private String mn;
        private String name;
        private int age;
        private String mbox;
        private String gender;

        private HashMap<String, String> dept = new HashMap<String, String>();

        private void setDept() {
                dept.put("chem", "Chemistry");
                dept.put("geo", "Geosciences");
                dept.put("inf", "Informatics");
                dept.put("ling", "Linguistics");
                dept.put("math", "Mathematics");
                dept.put("phys", "Physics");
        }

        public String getDept() {
                setDept();
                String[] mailDomain = mbox.split("@");
                String domain = mailDomain[1];
                return dept.get(domain);
        }

        public String getMn() {
                return mn;
        }

        public void setMn(String mn) {
                this.mn = mn;
        }

        public String getName() {
                return name;
        }

        public void setName(String name) {
                this.name = name;
        }

        public int getAge() {
                return age;
        }

        public void setAge(int age) {
                this.age = age;
        }

        public String getMbox() {
                return mbox;
        }

        public void setMbox(String mbox) {
                this.mbox = mbox;
        }

        public String getGender() {
                return gender;
        }

        public void setGender(String gender) {
                this.gender = gender;
        }

}
```

`Roster.java`:

```
import java.util.*;

public class Roster {
        private ArrayList<Student> students = new ArrayList<Student>();

        public void makeRoster() {
                Student peter = new Student();
                peter.setMn("s0189034");
                peter.setName("Peter");
                peter.setAge(17);
                peter.setMbox("peter@math");
                peter.setGender("M");

                Student michael = new Student();
                michael.setMn("s0289125");
                michael.setName("Michael");
                michael.setAge(21);
                michael.setMbox("mike@geo");
                michael.setGender("M");

                Student helen = new Student();
                helen.setMn("s0378435");
                helen.setName("Helen");
                helen.setAge(28);
                helen.setMbox("helen@phys");
                helen.setGender("F");

                Student mary = new Student();
                mary.setMn("s0412375");
                mary.setName("Mary");
                mary.setAge(18);
                mary.setMbox("mary@inf");
                mary.setGender("F");

                Student john = new Student();
                john.setMn("s0456782");
                john.setName("John");
                john.setAge(22);
                john.setMbox("john@inf");
                john.setGender("M");

                Student dana = new Student();
                dana.setMn("s0355689");
                dana.setName("Dana");
                dana.setAge(33);
                dana.setMbox("dana@ling");
                dana.setGender("F");

                Student lee = new Student();
                lee.setMn("s0768633");
                lee.setName("Lee");
                lee.setAge(36);
                lee.setMbox("lee@chem");
                lee.setGender("F");

                students.add(peter);
                students.add(michael);
                students.add(helen);
                students.add(mary);
                students.add(john);
                students.add(dana);
                students.add(lee);
        }

        public ArrayList<Student> getRoster() {
                return students;
        }

        /**
         * Build a HashMap with the student's name as the key and the student record
         * as a value for any student aged over 18 but under 25.
         *
         * @return The constructed HashMap.
         */
        public HashMap<String, Student> getMedianStudentsMap() {
                HashMap<String, Student> result = new HashMap<String, Student>();
                for (Student s : students) {
                        if (s.getAge() > 18 && s.getAge() < 25) {
                                result.put(s.getName(), s);
                        }
                }
                return result;
        }
        /**
         * Return the Student record of student with a given name.
         *
         * @param name
         *            The name of the student to look-up.
         * @return The instance of Student with that name.
         */
        public Student getStudentByName(String name) {
                for (Student s : students) {
                        if (s.getName().equals(name)) {
                                return s;
                        }
                }
                return null;
        }

        /**
         * Return the department of the student whose name was passed in as argument
         * to the method.
         *
         * @param name
         *            The name of the student to look-up.
         * @return The student's department.
         */
        public String getStudentDepartment(String name) {
                for (Student s : students) {
                        if (s.getName().equals(name)) {
                                return s.getDept();
                        }
                }
                return null;
        }

        /**
         * Build an ArrayList of Student that is older than the minAge, and of the
         * Gender given in gender
         *
         * @param minAge
         *            The minimum age to be considered.
         * @param gender
         *            The required gender "M" or "F"
         * @return
         */
        public ArrayList<Student> getStudentsByAgeAndGender(int minAge,
                        String gender) {
                ArrayList<Student> result = new ArrayList<Student>();
                for (Student s : students) {
                        if (s.getAge() > minAge && s.getGender().equals(gender)) {
                                result.add(s);
                        }
                }
                return result;
        }

        /**
         * Print out the roster of students.
         */
        public void printRoster() {
                for (Student s : students) {
                        System.out.println("Name: " + s.getName());
                }
        }

}
```

RosterTest.java:

```
import java.util.*;

import junit.framework.TestCase;

public class RosterTest extends TestCase {

        private Roster r;
        private String[] names;
        private HashMap<String, String> studentDepts;

        protected void setUp() {
                r = new Roster();
                r.makeRoster();
                names = new String[7];
                names[0] = "Peter";
                names[1] = "Michael";
                names[2] = "Helen";
                names[3] = "Mary";
                names[4] = "John";
                names[5] = "Dana";
                names[6] = "Lee";

                studentDepts = new HashMap<String, String> ();
                studentDepts.put("Peter", "Mathematics");
                studentDepts.put("Michael", "Geosciences");
                studentDepts.put("Helen", "Physics");
                studentDepts.put("Mary", "Informatics");
                studentDepts.put("John", "Informatics");
                studentDepts.put("Dana", "Linguistics");
                studentDepts.put("Lee", "Chemistry");

        }

        public final void testGetMedianStudentsMap() {
                HashMap<String, Student> studentMap = r.getMedianStudentsMap();
                assertTrue(studentMap.containsKey("Michael"));
                assertTrue(studentMap.containsKey("John"));
        }

        public final void testGetStudentByName() {
                for (String name : names) {
                        assertNotNull(r.getStudentByName(name));
                }
        }

        public final void testGetStudentDepartment() {
                for (String s : studentDepts.keySet()) {
                        assertEquals(studentDepts.get(s), r.getStudentDepartment(s));
                }
        }

        public final void testGetStudentsByAgeAndGender() {
                ArrayList<Student> s1 = r.getStudentsByAgeAndGender(21, "M");
                assertEquals("John", s1.get(0).getName());
                ArrayList<Student> s2 = r.getStudentsByAgeAndGender(36, "M");
                assertTrue(s2.isEmpty());
                ArrayList<Student> s3 = r.getStudentsByAgeAndGender(0, "F");
                assertEquals(4, s3.size());
        }


}
```

OPTIONAL Exercise 4: Cryptography with a One-time Pad

The task in this exercise is to implement a class `Crypto` which contains methods for encrypting and decrypting a message using a one-time pad. A one-time pad is an encryption technique where the plaintext message is combined with a random key or "pad" that is as long as the plaintext and used only once. Modular addition is used to combine the plaintext with the pad. If the key is truly random, never reused, and kept secret, the one-time pad provides perfect secrecy. The pad is sometimes called a "keystream".

Here is a brief description of the technique by Bruce Schneier, taken from the Appendix of Neal Stephenson's novel *Cryptonomicon*.

Assume your cleartext is the string:

```
DONOTUSEPC
```

(i.e. "do not use PC"). Note that this technique only represents the 26 letter of the alphabet, and doesn't do anything with punctuation or spaces. Assume your keystream string is:

```
KDWUPONOWT
```

Encryption

Step 1. Convert the cleartext message from letters to numbers: *A* = 1, *B* = 2, etc:

```
4 15 14 15 20 21 19 5 16 3
```

Step 2. Convert the keystream letters similarly:

```
11 4 23 21 16 15 14 15 23 20
```

Step 3. Add the cleartext number stream to the keystream numbers, modulo 26. (I.e., if the sum is more than 26, substract 26 from the result.) For example, 1 + 1 = 2, 26 + 1 = 27, and 27 - 26 = 1, so 26 + 1 = 1:

```
15 19 11 10 10 10 7 20 13 23
```

Step 4. Convert the numbers back to letters:

```
OSKJJJGTMW
```

This string of letters is the *ciphertext*.

Decryption

Assume you have a ciphertext message like the string above. To decrypt, you have to use the same keystream that way used for encryption.

Step 1. Convert the ciphertext message into numbers:

```
15 19 11 10 10 10 7 20 13 23
```

Step 2. Convert the keystream letters similarly:

```
11 4 23 21 16 15 14 15 23 20
```

Step 3. Subtract the keystream numbers from the ciphertext numbers, modulo 26. For example, 22 - 1 = 20, 1 - 22 = 5 (If the first number is less than the second number, add 26 to the first number before subtracting. So 1 - 22 = ? becomes 27 - 22 = 5.):

```
4 15 14 15 20 21 19 5 16 3
```

Step 3. Convert the numbers back to letters:

```
DONOTUSEPC
```

For an alternative source of information, you can look at these Wikipedia articles:

- http://en.wikipedia.org/wiki/One-time_pad
- http://en.wikipedia.org/wiki/Keystream

In the `Crypto` class, you should implement methods corresponding to the following declarations:

```
public String encrypt(String clearText, String keyStream) {
...
}

public String decrypt(String cipherText, String keyStream) {
...
}
```

These should satisfy the following pair of equations:

1. cipherText = encrypt(clearText, keyStream)
2. clearText = decrypt(cipherText, keyStream)

Here is a ciphertext:

```
CUFNFLRQMLWBLARVJDALMURNGSHILXXMVDUMFPYG
```

which I encrypted using this keystream:

```
ACWZYRJYHGVNHUCARNVXJPUIFACBWOJFBOTIEBVBBZHANSLYOP
```

If your `decrypt()` method works, you should be able to figure out the original cleartext message. Email me the answer!

Date: Wed 18 Feb 2009 11:42:23 GMT

OOP Informatics Home