TESTING ARRAYS

CSC Software Testing Materials (Testing: Arrays)

@ Dr. Jessica Young Schmidt and NCSU Computer Science Faculty

Input

The Paycheck program prompts the user for information about each Employee, including the name, level (1, 2, or 3), hours worked, retirement percent, and whether he or she has medical, dental, and vision insurances. This version of the program has extensive error checking and loops to allow for processing more than one paycheck at a time. The user is prompted for the number of paychecks and stores paycheck information.

Output

The following information is printed about each employee's pay check:

1. employee's name
2. hours worked for a week
3. hourly pay rate
4. regular pay for up to 40 hours worked
5. overtime pay (1.5 pay rate) for hours over 40 worked
6. gross pay (regular + overtime)
7. total deductions
8. net pay (gross pay – total deductions).

If the net pay is negative, meaning the deductions exceeds the gross pay, then an error is printed.

System Testing with Arrays

For System Tests for Robust Paycheck with Storage, like with Robust Paycheck System Testing, we want to test different number of paychecks.

Unit and Integration Testing with Arrays

When testing arrays, we examine array size and array contents. For methods that involve array traversals, you should write tests that potentially will traverse some of the array, all of the array, and none of the array.

Testing Equality of Arrays

The JUnit library includes many different types of assert methods. We discussed many method in the Unit and Integration Testing section. For testing arrays, the `assertArrayEquals` methods will be helpful to us.

Method	Description

`assertArrayEquals(int[] expected, int[] actual, String message)`
`assertArrayEquals(char[] expected, char[] actual, String message)`
`assertArrayEquals(Object[] expected, Object[] actual, String message)`
	asserts that `expected` array equals the `actual` array. Otherwise, the test case will fail with the given.

`assertArrayEquals(double[] expected, double[] actual, double delta, String message)`	asserts that `expected` and `actual` double arrays are equal to within a non-negative `delta`. Otherwise, the test case will fail.

RobustPaycheckStorage.java:

➤JAVA

```java
    /**
     * Returns 2D array of strings where the rows represent the paychecks. The
     * first column is the employee name. The second column is the formatted
     * paycheck information.
     *
     * @param numPaychecks number of paychecks (rows of array)
     * @return empty 2D array of strings
     * @throws IllegalArgumentException if non-positive numPaychecks
     */
    public static String[][] createPaychecks(int numPaychecks) {
        if (numPaychecks <= 0) {
            throw new IllegalArgumentException("Invalid number of paychecks");
        }
        return new String[numPaychecks][2];
    }
```

RobustPaycheckStorageTest.java:

➤JAVA

```java
    /**
     * Test the RobustPaycheckStorage.createPaychecks() method.
     */
    @Test
    public void testCreatePaycheck() {
        String[][] exp1 = { { null, null } };
        String[][] act1 = RobustPaycheckStorage.createPaychecks(1);
        // Test the size of array
        assertEquals(1, act1.length, "Check length of first dimension");
        assertEquals(2, act1[0].length, "Check length of second dimension");
        // test results
        assertArrayEquals(exp1, act1, "Check that array of single paycheck created");

        String[][] exp3 = { { null, null }, { null, null }, { null, null } };
        String[][] act3 = RobustPaycheckStorage.createPaychecks(3);
        // Test the size of array
        assertEquals(3, act3.length, "Check length of first dimension");
        assertEquals(2, act3[0].length, "Check length of second dimension");
        // test results
        assertArrayEquals(exp3, act3, "Check that array of three paychecks created");

        // Test invalid parameters
        Exception exception = assertThrows(IllegalArgumentException.class,
            () -> RobustPaycheckStorage.createPaychecks(0),
                "Testing non-positive number of paychecks");
        assertEquals("Invalid number of paychecks", exception.getMessage(),
                "Testing non-positive number of paychecks message");
        exception = assertThrows(IllegalArgumentException.class,
            () -> RobustPaycheckStorage.createPaychecks(-1),
                "Testing non-positive number of paychecks");
        assertEquals("Invalid number of paychecks", exception.getMessage(),
                "Testing non-positive number of paychecks message");
        exception = assertThrows(IllegalArgumentException.class,
            () -> RobustPaycheckStorage.createPaychecks(-100),
                "Testing non-positive number of paychecks");
        assertEquals("Invalid number of paychecks", exception.getMessage(),
                "Testing non-positive number of paychecks message");
    }
```

If a method should not modify the contents of the array, we should additionally test that the array is not modified.

RobustPaycheckStorage.java:

➤JAVA

```java
    /**
     * Returns paycheck information for the given name. If multiple paychecks
     * for the given name, return the first paycheck (smallest index). If name
     * is not in the array, then return "No such user."
     *
     * @param paychecks array of paychecks
     * @param name name of employee
     * @return paycheck information for given name
     * @throws IllegalArgumentException when either parameter is null
     */
    public static String getPaycheck(String[][] paychecks, String name) {
        if (paychecks == null || name == null) {
            throw new IllegalArgumentException("Invalid parameters");
        }
        for (int i = 0; i < paychecks.length; i++) {
            if (paychecks[i][0].equals(name)) {
                return paychecks[i][1];
            }
        }
        return "No such user.";
    }
```

RobustPaycheckStorageTest.java:

➤JAVA

```java
    /**
     * Test the RobustPaycheckStorage.getPaycheck() method.
     */
    @Test
    public void testGetPaycheck() {

        // Array with single user
        String[][] single = { { "Carol Cole",
                "Carol Cole                10.00     19.00    190.00      "
                        + "0.00    190.00      0.00    190.00\n" } };

        // Test the size of array
        assertEquals(1, single.length, "Check length of first dimension");
        assertEquals(2, single[0].length, "Check length of second dimension");
        // Test returns correct paycheck
        assertEquals(
                "Carol Cole                10.00     19.00    190.00      "
                        + "0.00    190.00      0.00    190.00\n",
                RobustPaycheckStorage.getPaycheck(single, "Carol Cole"),
                "Check that correct paycheck returned for single paycheck");

        // Test that method doesn't modify array
        String[][] singleCopy = { { "Carol Cole",
                "Carol Cole                10.00     19.00    190.00      "
                        + "0.00    190.00      0.00    190.00\n" } };
        assertArrayEquals(singleCopy, single, "Check that array not modified");

        // Test returns no user with name
        assertEquals("No such user.",
                RobustPaycheckStorage.getPaycheck(single, "Carol"),
                "Check that message for no user.");

        // Test that method doesn't modify array
        assertArrayEquals(singleCopy, single, "Check that array not modified");

        // Array with single user
        String[][] multiple = { { "Carol Cole",
                "Carol Cole                10.00     19.00    190.00      "
                        + "0.00    190.00      0.00    190.00\n" },
            { "Carol Cole",
                "Carol Cole                11.00     19.00    209.00      "
                        + "0.00    209.00      0.00    209.00\n" } };

        // Test the size of array
        assertEquals(2, multiple.length, "Check length of first dimension");
        assertEquals(2, multiple[0].length, "Check length of second dimension");
        // Test returns correct paycheck
        assertEquals(
                "Carol Cole                10.00     19.00    190.00      "
                + "0.00    190.00      0.00    190.00\n",
                RobustPaycheckStorage.getPaycheck(single, "Carol Cole"),
                "Check that correct paycheck returned for multiple paychecks");

        // Test that method doesn't modify array
        String[][] multipleCopy = { { "Carol Cole",
                "Carol Cole                10.00     19.00    190.00      "
                        + "0.00    190.00      0.00    190.00\n" },
            { "Carol Cole",
              "Carol Cole                11.00     19.00    209.00      "
                        + "0.00    209.00      0.00    209.00\n" } };
        assertArrayEquals(multipleCopy, multiple, "Check that array not modified");

        // Test invalid parameters
        Exception exception = assertThrows(IllegalArgumentException.class,
            () -> RobustPaycheckStorage.getPaycheck(null, "CSC"),
                "Testing null array");
        assertEquals("Invalid parameters", exception.getMessage(),
                "Testing null array");
        String[][] arr = { { "Cat", "Dog", "Puppy" }, { "A", "B", "C" } };
        exception = assertThrows(IllegalArgumentException.class,
            () -> RobustPaycheckStorage.getPaycheck(arr, null),
                "Testing null name");
        assertEquals("Invalid parameters", exception.getMessage(),
                "Testing null name");
    }
```