**Project: SelectionMethods**

This project asks you to solve ten problems related to strings, Boolean expression, and selection control with if, if...else, and nested if...eloses. You are asked to write ten methods in one class named SelectionMethods along with at least one @Test methods for each in another class named SelectionMethodsTest. Suggestion: Develop one method at a time.

Here is the beginning of two classes needed for this project with a test method for the first problem.

```java
// This class has @Test methods for the all ten methods in SelectionMethods.java
// Programmer: YOUR NAME
//
import static org.junit.Assert.*;
import org.junit.Test;

public class SelectionMethodsTest {
    @Test
    public void testIsLeapYear() { // TODO Complete this method
        SelectionMethods sm = new SelectionMethods();
        assertTrue(sm.isLeapYear(2008));
        assertFalse(sm.isLeapYear(2009));
        assertFalse(sm.isLeapYear(2100));
        assertTrue(sm.isLeapYear(2400));
        // add more assertions . . .
    }

    // At least nine more test methods with many, many,
    // many assertions must be completed below . . .
}
```

```java
// This class has ten unrelated methods that process primitive types and
// Strings, most requiring selective control structures.
// Programmer: YOUR NAME
//
public class SelectionMethods {

    public boolean isLeapYear(int year) {
        // TODO Complete this method
        return false;
    }

    // Nine more methods and will be completed in this class . . .
}
```

1) **public boolean isLeapYear(int year)**

Complete method isLeapYear that returns true if the integer argument represents a leap year. A leap year is a positive integer that is evenly divisible by 4 except the last year of a century. These are the years evenly divisible by 100. To be a leap year the last year of a century must also be divisible by 400. This mans that 2000 was a leap year and 2400 will be a leap year because they are evenly divisible by 400. On the other hand, 2100, 2200, and 2300 will not be leap years because they are the last year of a century that is not evenly divisible by 400. Arguments can only be positive.

isLeapYear(2008) → true
isLeapYear(2009) → false
isLeapYear(2100) → false
2) public int teenSum(int a, int b)

Given 2 ints, a and b, return their sum. However, "teen" values in the range 13..19 inclusive, are extra lucky. So if either value is a teen, just return 19.

teenSum(3, 4) → 7
teenSum(10, 13) → 19
teenSum(13, 2) → 19

Here are two assertions provided as an example (you need more assertions):

```java
@Test
public void testTeenSumWhenResultShouldBe19() {
    SelectionMethods sm = new SelectionMethods();
    assertEquals(19, sm.teenSum(13, 15));
    assertEquals(19, sm.teenSum(1, 19));
    // add more assertions . . .
}
```

3) public int redTicket(int a, int b, int c)

You have a red lottery ticket showing ints a, b, and c, each of which is 0, 1, or 2. If they are all the value 2, the result is 10. Otherwise if they are all the same, the result is 5. Otherwise so long as both b and c are different from a, the result is 1. Otherwise the result is 0.

redTicket(2, 2, 2) → 10
redTicket(2, 2, 1) → 0
redTicket(0, 0, 0) → 5

Here are two assertions provided as an example (you need more assertions):

```java
@Test
public void testRedTicket1() {
    SelectionMethods sm = new SelectionMethods();
    assertEquals(1, sm.redTicket(1, 2, 0));
    // add more assertions . . .
}
```

4) public int teaParty(int tea, int candy)

We are having a party with amounts of tea and candy. Return the int outcome of the party encoded as 0=bad, 1=good, or 2=great. A party is good (1) if both tea and candy are at least 5. However, if either tea or candy is at least double the amount of the other one, the party is great (2). However, in all cases, if either tea or candy is less than 5, the party is always bad (0). Arguments can only be 0, 1, or 2.

teaParty(6, 8) → 1
teaParty(3, 8) → 0
teaParty(20, 6) → 2
5) **public int** daysInMonth(int month)

Write daysInMonth to return the number of days in a month. Assume there are no leap years so daysInMonth(2) for February always returns 28. Arguments can only be 1 through 12.

- daysInMonth(1) → 31
- daysInMonth(2) → 28

6) **public String** firstOf3Strings(String a, String b, String c)

Given three String arguments, return a reference to the String that is not "greater than" the other two. Use String's compareTo method. Note: "A" is less than "a" and "abc" is less than "abc ".

- firstOf3Strings("c", "b", "a") → "a"
- firstOf3Strings("B", "B", "A") → "A"
- firstOf3Strings("ma", "Ma", "ma") → "Ma"
- firstOf3Strings("a ", "a ", "a ") → "a 

Here is one assertion provided as an example (you need more assertions):

```java
@Test
public void testFirstOf3Strings() {
    SelectionMethods sm = new SelectionMethods();
    assertEquals("First", sm.firstOf3Strings("Third", "Second", "First"));
    // add more assertions . . .
}
```

7) **public String** season(int month, boolean inNorthernHemisphere)

Given an integer for the month (1 is January and 12 is December) and a Boolean argument that represents the northern hemisphere when true, return the current season in that hemisphere using the table below. Arguments for month can only be 1 through 12.

- season(12, true) → "Winter"
- season(12, false) → "Summer"
- season(3, true) → "Spring"
- season(3, false) → "Fall"

Use the following table to determine the season for each month.

<table>
<thead>
<tr>
<th>month</th>
<th>inNorthernHemisphere</th>
<th>!inNorthernHemisphere</th>
</tr>
</thead>
<tbody>
<tr>
<td>12, 1, or 2</td>
<td>&quot;Winter&quot;</td>
<td>&quot;Summer&quot;</td>
</tr>
<tr>
<td>3, 4, or 5</td>
<td>&quot;Spring&quot;</td>
<td>&quot;Fall&quot;</td>
</tr>
<tr>
<td>6, 7, or 8</td>
<td>&quot;Summer&quot;</td>
<td>&quot;Winter&quot;</td>
</tr>
<tr>
<td>9, 10, 11</td>
<td>&quot;Fall&quot;</td>
<td>&quot;Spring&quot;</td>
</tr>
</tbody>
</table>

Make sure you test all branches through the nested if-else. If you don't, the code coverage check in WebCat will deduct points from your score.
8) **public int squareSum(int a, int b)**

Given two integers, if they are both even or both odd, return the sum of their squares (a*a+b*b). Otherwise return the square of their sums (a+b)*(a+b). Assume both arguments are non-negative.

```
squareSum(2, 4) → 20
squareSum(2, 3) → 25
squareSum(5, 4) → 81
```

9) **public String letterGrade(double percentage)**

Return a letter grade according to this scale. Assume percentage is always in the range of 0.0 through 100.0. Arguments for percentage can only be 0.0 through 100.0

<table>
<thead>
<tr>
<th>Percentage</th>
<th>Letter Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>90.0 ≤ percentage ≤ 100.0</td>
<td>A</td>
</tr>
<tr>
<td>80.0 ≤ percentage &lt; 90.0</td>
<td>B</td>
</tr>
<tr>
<td>70.0 ≤ percentage &lt; 80.0</td>
<td>C</td>
</tr>
<tr>
<td>60.0 ≤ percentage &lt; 70.0</td>
<td>D</td>
</tr>
<tr>
<td>0.0 ≤ percentage &lt; 60.0</td>
<td>E</td>
</tr>
</tbody>
</table>

`letterGrade(90.0) → "A"
letterGrade(85.5) → "B"`

```
@Test
def testLetterGradeWhenA():
    sm = new SelectionMethods()
    assertEquals("A", sm.letterGrade(100))
    assertEquals("A", sm.letterGrade(90))
```

10) **public int tDate(int firstDay)**

In the US, Thanksgiving falls on the 4th Thursday of each November. Complete method tDate that determines the day Thanksgiving falls on (an integer from 1 to 30) no matter the day on which November begins. November can begin on any day where 1 represents Monday, through 7 which represents Sunday. A valid call would be `tDate(2)` to indicate the first day of November is Tuesday. tDate should then return the day of the month upon which Thanksgiving falls, which is 24. Arguments can only be 1 (for Monday) through 7 (for Sunday).

```
tDate(2) → 24
tDate(4) → 22
```
Grading Criteria (100pts)

When you have completely tested all 10 methods, turn in your project to WebCat. You will be graded as follows:

___+90 Web-Cat correctness and code coverage.
   • Your code must compile using the specified method names
   • Rick's tests must pass
   • You must have tested all of your methods with assertions in test methods
   • You must execute all statements in all 10 methods at least once  (See WebCat Code Coverage)
   • All of your assertions must pass on WebCat
   • The final WebCat submission will be the one that is graded unless you inform us
   • WebCat employs a multiplication feature that means 90% Problem coverage and 90% Code Coverage results in $0.81 \times 90$ or 72.9/90 points.

___+10 pts Style and Design
   • 2pts You named your unit test SelectionMethods.java
   • 2pts You have your name as a comment at the top of the both files
   • 3pts You used meaningful identifiers
   • 3pts All code is formatted consistently (use Eclipse’s Source > Format)