CS 145 Half-Homework 3
Trutilities

1 Overview

Your objective in this homework is to reason about data using logical and relational operators. Expressions built out of these operators yield true/false or yes/no answers, which can ultimately be used to steer our code one way or another. You will use operators to solve several disconnected problems that have no overarching story. Sorry again.

2 Warning

Some of you who have programmed before may be familiar with `if` statements. You might be tempted to solve some of these problems with code like this:

```
public static boolean meetsSomeCriteria(String name) {
  if (someCondition) {
    return true;
  } else {
    return false;
  }
}
```

This is bad form. When the condition is `true`, you return `true`. When it's `false`, you return `false`. Why not just return the value of the condition directly?

```
public static boolean meetsSomeCriteria(String name) {
  return someCondition;
}
```

It's true that boolean expressions do usually end up embedded in an `if` statement (or a loop), but you won't need them for this assignment, which focuses only on boolean operators. Do not use `if` statements in this homework.

3 Another Warning

You might be tempted to see if a boolean value is `true` by writing something like this:

```
return isImportant == true;
```

There's something awkward here. Let's break down this expression into a truth table:

`isImportant`	`isImportant == true`
`false`	`false`
`true`	`true`

The second column is identical to the first. Do you see that comparing `isImportant` to `true` doesn't give us any information we didn't already have with just `isImportant`? `== true` is a lot like `+ 0` or `* 1`. It is the *identity* operation of boolean logic, only giving back what you put in. Dispense with comparing to `true`:

```
return isImportant;
```

Okay, okay, you say. But what about these two constructs?

```
return isImportant == false;
return isImportant != true;
```

Let's again check out the truth table:

isImportant	isImportant == false	isImportant != true
false	true	true
true	false	false

Notice how these operations just flip the value of `isImportant`. Equality with `false` and inequality with `true` are better expressed using the `!` operator:

```
return !isImportant;
```

This construction is much more readable. Which would you rather say: a) "Is important is `false`?", b) "Is important is not `true`?", or c) "Is not important?"

4 Requirements

Complete the two classes described below. Place all classes in package `hw3`. Make all methods `static`.

4.1 Main

Write a class `Main` with a `main` method, which you are encouraged to use to test your code. Nothing in particular is required of it, but it must exist.

4.2 Trutilities

Write a class `Trutilities` with the following:

1. Method `isOrdered`, which accepts five parameters of type `int`. It returns `true` if the numbers are ordered in either ascending or descending order. Otherwise, it returns `false`. For example:
   - `Trutilities.isOrdered(1, 2, 3, 4, 5)` → `true`
   - `Trutilities.isOrdered(1, 2, 75, 4, 5)` → `false`
2. Method `isGreenish`, which accepts one parameter: a hexadecimal color of type `String`. Assume the color is of the format `#rrggbb`. It returns `true` if the green intensity is greater than both the red and blue intensities. For example:

- `Trutilities.isGreenish("#00ff00")` → `true`
- `Trutilities.isGreenish("#fffeff")` → `false`

3. Method `isMilitary`, which accepts one parameter: a possible military time, of type `String`. Assume the time is of the format `HHMM`, comprised of exactly four digits. It returns `true` if the time is a valid military time, with the hours in $[0, 23]$ and the minutes in $[0, 59]$. For example:

   - `Trutilities.isMilitary("0037")` → `true`
   - `Trutilities.isMilitary("2499")` → `false`

4. Method `isImage`, which accepts one parameter: a file of type `File`. It returns `true` if the name of the file ends with `.png`, `.gif`, `.jpg`, or `.jpeg`, regardless of case. Otherwise, it returns `false`. The portion of a file name after the last period is called the file's extension. Determining file types through extensions is not always reliable (a file may be misnamed), but it's often good enough. For example:

   - `Trutilities.isImage(new File("a.png"))` → `true`
   - `Trutilities.isImage(new File("bowie.docx"))` → `false`

5. Method `hasMultipleDots`, which accepts one parameter: some text of `String`. It returns `true` if the text contains at least two periods. Otherwise, it returns `false`. For example:

   - `Trutilities.hasMultipleDots("...")` → `true`
   - `Trutilities.hasMultipleDots("e e cummings")` → `false`

6. Method `fitsAspect`, which accepts three parameters:

   (a) a width of type `int`
   (b) a height of type `int`
   (c) an aspect ratio of type `double`

   Aspect ratio is a term from the media industry that refers to the ratio of a display's width to its height. An aspect ratio of 2 means that the display is twice as wide as it is tall. A 15" MacBook Pro has a display that's 1440 pixels wide and 900 pixels tall. Its aspect ratio is 1.6. This method returns `true` if the given `int` dimensions are of the given aspect ratio. Otherwise, it returns `false`. For example:

   - `Trutilities.fitsAspect(2000, 1000, 2)` → `true`
   - `Trutilities.fitsAspect(640, 480, 1.5)` → `false`

   Comparing `doubles` for equality with `==` is a dangerous business because computers are finite and at some point digits in long numbers get lost. To avoid danger, first compute the difference between the dimensions' aspect ratio and the target. Then determine if the gap between these values (the absolute difference) is sufficiently small. For this problem, let's say two aspect ratios are equivalent if the difference between them is less than 0.001.

7. Method `fitsWithin`, which accepts four parameters:

(a) the width of rectangle A, of type `int`

(b) the height of rectangle A, of type `int`

(c) the width of rectangle B, of type `int`

(d) the height of rectangle B, of type `int`

This method returns `true` is rectangle A fits within rectangle B, rotating 90 degrees if necessary. Otherwise, it returns `false`. For example:

- `Trutilities.fitsWithin(2, 2, 2, 4)` → `true`
- `Trutilities.fitsWithin(1, 10, 20, 2)` → `true`
- `Trutilities.fitsWithin(5, 5, 4, 3)` → `false`

8. Method `isFaster`, which accepts two parameters:

   (a) time A of type `String`

   (b) time B of type `String`

   Both times are of the format `H:M`. Assume that both the hours and minutes are comprised of one or more digits, and that they are positive values. Do not assume that they are within a particular numeric range. Minutes, in particular, may be greater than 60. This method returns `true` if time A is faster than time B. Otherwise, it returns `false`. For example:

   - `Trutilities.isFaster("1:15", "2:30")` → `true`
   - `Trutilities.isFaster("0:90", "1:30")` → `false`

9. Method `isIllegal`, which accepts one parameter: a possible direction of type `String`. It returns `false` if the direction is exactly one of `north`, `south`, `east`, or `west`, regardless of case. Otherwise, it returns `true`. For example:

   - `Trutilities.isIllegal("North")` → `false`
   - `Trutilities.isIllegal("right")` → `true`

10. Method `isOld`, which accepts four parameters:

    (a) a file of type `File`

    (b) a year of type `int`, assumed valid

    (c) a month of type `int`, in $[1, 12]$

    (d) a day of type `int`, assumed valid

    This method returns `true` if the file was last modified on the given date or earlier. Otherwise, it returns `false`.

    There are several thousand ways to solve this. If we can reduce our two times down to a single dimension in the same units, we can solve this with a single comparison. Good news: we can. A method in the `File` class provides the file's modification time as a number of milliseconds since January 1, 1970. Reducing the other date is a little more involved. Construct an instance of the `GregorianCalendar` class, and call `getTimeInMillis`. Be wary that `GregorianCalendar` considers January as month 0 and December as month 11.

5 Submission

To submit your work for grading:

1. Put the SpecChecker for this homework in your Build Path. Run the SpecChecker as a Java Application and fix problems until all tests pass.

2. Commit and push your work to your repository. Verify that your solution is on Bitbucket.

A passing SpecChecker does not guarantee you credit. Your grade is conditioned on a few things:

- You must meet the requirements described above. The SpecChecker checks some of them, but not all.

- You must not plagiarize. Write your own code. Talk about code with your classmates. Ask questions of your instructor or TA. Do not look at others' code. Do not ask questions specific to your homework anywhere online but Piazza. Your instructor employs a vast repertoire of tools to sniff out academic dishonesty, including: drones, CS 1 moles, and a piece of software called MOSS that rigorously compares your code to every other submission. You don't want to live in a world serviced by those who squeaked by through questionable means. For your future self, career, and family, do your own work.

- Your code must be submitted correctly and on time. Most excuses devolve into, "I started too late." The fix for this problem is not an extension.