

Project #05– Exception Handling and an Immutable Time Class

Due: Friday, November 1st

You must upload a single ZIP file containing all of your JAVA source code files on the course MOODLE site before the start of the lecture on the due date. Include screencaps of your output from testing your program a few times (the Snipping Tool is handed to capture portions of your screen).

Points will be deducted from late submissions unless previously cleared with me.

An *immutable* class is one where instances of the object cannot be changed once created. If the properties are declared as **private**, and there are NO mutator methods that allow a method to change the properties, this ensures that the property values cannot be changed after the object has been created.

Your task is to design a class called **Time**. It has three properties:

- hours – must be between 0 and 23, inclusive
- minutes – must be between 0 and 59, inclusive
- seconds – must be between 0 and 59, inclusive

(1) The class should have the traditional constructor methods. It should have a **toString()** method. It must **NOT** have mutator methods. And it may have accessor methods (although they probably are not needed). For the toString() method, make sure that the returned string is properly formatted with each field taking up two digits to display (e.g., "09:17:43" or "22:09:08").

(2) Your class should have a set of overloaded methods called **later()**, which is a nonstatic method. When invoked, it will **create and return a brand new** Time object, set to the specified number of hours, minutes, and/or seconds later than the current Time object's time. There should be three versions of this method – (1) one that takes in just the number of hours to advance, (2) one that takes in the number of hours and minutes, and (3) one that takes in the number of hours, minutes, and seconds.

(3) Now here is where the Exception handling comes in. Your class and your program should handle all appropriate standard exceptions that may arise. In addition, you will create a new exception type called **BadTimeException**, that will be generated whenever you attempt to create a new Time object whose values do not meet the requirements. However, instead of simply exiting out of the program, your class will handle the BadTimeException by calling a **helper method** that will "fix" the properties so they make sense.

A Time object's can be "fixed" according to the rules of time. If you have more than 60 seconds, that amount is equivalent to a certain number of minutes plus the leftover

seconds (e.g., 189 seconds is equivalent to 3 minutes & 9 seconds). Likewise with overflow of minutes can be adjusted by adding the required number of hours (117 minutes = 1 hour and 57 minutes). For hours, we are not keeping track of number of days, so we can just cycle 24 hours to get the value in the proper range (e.g., 37 hours becomes 13 hours (we drop the 1 day that is generated)).

Think carefully about how you want to organize and structure your program. Think of when the `BadTimeException` may be generated and try to confine the exception handling code to as few places in your code as possible.

(4) Once you are done, create a driver class and test your code. Make sure you use sufficient examples and numerous invocations of the different versions of your **later()** method to make sure the methods work and the exception handling works properly.

Highlights:

- Time class should be immutable
- Create a `BadTimeException` class that will be thrown whenever a `Time` object has values out of bounds
- Write a method to "fix" the time
- Structure and organize your program
- Test thoroughly and plan out good test cases