Exceptions

1. Create a method, ReadInteger which reads a line from the command line and returns the input parsed as integer. Use the static method **int.Parse**. Your ReadInteger metod will only return on a successful parse. If the input is not an integer, an appropriate error message will written in the terminal, and the method will read another line. You need to use try/catch
    a. Call your newly created method, and
        i. type in "foo" – confirm the error message
        ii. type in "10000000000000000" – confirm

2. In the FileSystemsMenu – browse to a directory created for the purpose of this exercise (e.g. C:\foo with files a.txt and b.txt of random content)
    a. While displaying the content of the folder – use your file manager to delete the file a.txt. Then select a.txt – what happens? – Handle the where appropriate
    b. If your system supports it, change permissions of the file b.txt denying all access to the file, and select this file in the menu. – what happens? – Handle the where appropriate
    c. If you did the RSS exercise, handle exceptions after disabling internet access
3. Write a class to represent a bank account. An account has a balance. Add deposit and withdrawal methods. The balance of the account must always be non-negative. Write a class InsufficientFundsException, which extends Exception, and throw this exception (a) in the constructor if the balance is negative and (b) if a withdrawal would make the balance negative.
4. Write a class to represent a gearbox with five gears and a gear for reverse. Add a method changeGear(int gear) to change the current gear. The method must throw IllegalArgumentException if the gear is not one of -1, 1, 2, 3, 4, and 5. Here reverse is represented as -1. Write a class IllegalGearChangeException, which extends Exception, and throw this exception: (a) when switching from any gear other than the first gear into reverse and (b) when skipping one or more gears, e.g. it is illegal to switch directly from the first gear to the third gear
5. Write a class to represent a digital display with four digits[1]. Add a method GetDigit(int i) to return the value of the *i*th digit. Add a method SetDigit(int i, int v) to change the value of the *i*th digit to *v*. Add two exception classes: NoSuchDigitException and IllegalDigitException and throw these when appropriate.
6. Write a class to represent a printer from hell. The class should have a single method print(). Whenever this method is called, the printer randomly throws one of the following exceptions: OutOfPaperException, OutOfTonerException, PaperJamException. Write classes for these exceptions. Write a main method, which calls print(), catches any exception, prompts the user to take action (e.g. \replace toner"), waits for confirmation from the user, and then calls print() again. Bonus points for infuriating or vaguely worded instructions.
7. Write a class to represent a car. Add methods to (a) press the clutch, (b) release the clutch, (c) turn on the ignition, (d) turn off the ignition, (e) pull the handbrake, and (f) release the handbrake. To correctly turn on the car, the following

[1] Try a Google image search for "digital display" to see what I mean

steps should be taken in order: (1) press the clutch, (2) turn on the ignition, (3) release the handbrake, and (4) release the clutch. Add appropriate exceptions and throw these if the car is operated incorrectly.