Завдання до лабораторної роботи з теми «Виключні ситуації (exceptions)»

1. Напишіть програму, яка повинна прочитати з кожного з файлів `10.txt`, `11.txt`, `12.txt`, `13.txt`, `14.txt`, `15.txt`, `16.txt`, `17.txt`, `18.txt`, `19.txt`, `20.txt`, `21.txt`, `22.txt`, `23.txt`, `24.txt`, `25.txt`, `26.txt`, `27.txt`, `28.txt`, `29.txt` по два цілі числа (у звичайному десятковому записі, перше у першому рядку (при нумерації з 1), дру́ге у дру́гому), обчислити добутки цих чисел, і суму всіх цих добутків.

   Можливі які завгодно порушення: деяких з цих файлів може не бути; у деяких з файлів може бути неадекватна інформація (замість цілих чисел — дробові, або взагалі не числа), і так далі. Крім того, добутки потрібно рахувати в типі `int` (32-бітовому), і враховувати тільки ті з них, які поміщаються у цьому типі. (Перевірено, що у мові C# $\underline{\epsilon}$ засоби автоматичної генерації exception-а у такій ситуації.)

   Програма повинна видати у консоль знайдене середнє арифметичне (лише тих добутків, які вдалося обчислити), та створити/перезаписати файли:

   - `no_file.txt` — перелік відсутніх файлів
   - `bad_data.txt` — перелік файлів, з перших двох рядків яких яких неможливо прочитати два цілі числа у звичайному десятковому записі. Мова йде саме про перші два рядки: якщо саме вони не містять двох цілих чисел або містять щось зайве — вважати, що файл поганий, не намагаючись знайти їх десь далі; якщо після двох цілих чисел є ще щось — байдуже, воно може бути чим завгодно.
   - `overflow.txt` — перелік файлів, з яких можливо прочитати два цілі числа, які поміщаються у 32-бітовий тип `int`, але добуток цих прочитаних двох чисел не поміщається у 32-бітовий тип `int`.

   У випадку, якщо хоча б один з файлів `no_file.txt`, `bad_data.txt`, `overflow.txt` створити/оновити не виходить — програма повинна негайно обірватися, видавши у консоль (*своє!*) повідомлення про цей факт. Мається на увазі першопочаткове створення/оновлення; малоймовірну ситуацію, коли файл успішно створили, а потім по дорозі раптом стало неможливо в нього писати, дозволяється не обробляти.

   При всьому цьому, у програмі категорично заборонено мати хоча б один `if`: *усі* перевірки мають відбуватися засобами exception-ів. Цикл теж повинен бути один — той, який перебирає файли `10.txt`, `11.txt`, `12.txt`, `13.txt`, `14.txt`, `15.txt`, `16.txt`, `17.txt`, `18.txt`, `19.txt`, `20.txt`, `21.txt`, `22.txt`, `23.txt`, `24.txt`, `25.txt`, `26.txt`, `27.txt`, `28.txt`, `29.txt`.

   Тут може виникнути питання — «а як таке взагалі відлагодити, якщо я завжди можу писати у текстові файли?». Взагалі-то ситуацій, коли неможливо створити файл, не так уже й мало, а один з найпростіших способів створити таку ситуацію, не зв'язуючись ні з мережею, ні з правами доступу файлової системи — перед запуском програми створити *папку* `no_file.txt`, або `bad_data.txt`, або `overflow.txt`.

2. Напишіть програму, яка буде перебирати усі файли поточної папки і намагатися кожен з них розглянути як графічний (картинку) і виконати дзеркальне відбивання (по вертикалі).

   Список усіх файлів папки можна отримати як `Directory.GetFiles(...)`, де замість `...` можна або узяти поточну папку (`Directory.GetCurrentDirectory()`), або дописати, щоб конкретна папка вибиралася користувачем інтерактивно. Після цього конкретні файли можна перебрати просто як `foreach(String fileName in ...)`, де "`...`" — результат того самого `GetFiles`.

   Для роботи з графічними файлами пропонується використати `System.Drawing.Bitmap`. Цей клас вміє обробляти не лише формат `BMP`, а й деякі інші. Якщо і виявиться, що не всі

формати (на деякі картинки малопоширених форматів казатиме, що то не картинки) — вважайте це неприємним, але не істотним. Решту деталей знайдіть самостійно, користуючись MSDN та іншими інтернет-ресурсами.

Для кожного файла, який вдається успішно прочитати як картинку, треба створити новий. Незалежно від формату початкового файла, результат писати у форматі `gif`. Назвати новий файли треба так: узяти старе ім'я (відкинувши розширення, якщо воно було), дописати до імені "`-mirrored`" та замінити розширення на "`.gif`". Якщо файл не вдається прочитати як картинку (взнавати це слід за тим, чи виник exception у процесі читання картинки з файлу), поведінка повинна залежати від того, чи є його розширення одним із розширень графічних форматів: якщо не є — просто продовжити роботу, перейшовши до наступного файлу, якщо є — видати (через `MessageBox.Show(...)`) власне повідомлення про те, що файл не містить картинки, хоча, судячи з розширення, повинен. (Один зі способів перевірити розширення — перевірити відповідність власноруч написаному регулярному виразу. Наприклад, один раз перед циклом перебору файлів написати `Regex regexExtForImage = new Regex("^((bmp)|(gif)|(tiff?)|(jpe?g)|(png))$", RegexOptions.IgnoreCase);`, а у циклі перевіряти `if(regexExtForImage.IsMatch(Path.GetExtension(fileName)))`).

Насамкінець, воно на перший погляд здається складним (бо новим), а насправді все досить просто та вельми коротко: єдиний метод, порядка 30 рядків, з яких приблизно половина містять або саму лише фігурну дужку або щось дуже просте.

Оскільки програма повинна обробляти картинки, але не зобов'язана відображати їх, теоретично це може бути хоч програма з графічним інтерфейсом, хоч консольна програма (особливо якщо дозволити виводити повідомлення хоч `MessageBox`-ом,хоч у консоль). Практично при цьому виникає таке ускладеннння: у програмі з графічним інтерфейсом для підключення класів `Image` та `Bitmap` достатньо, щоб на початку тексту програми було написано `using System.Drawing;`, а для консольної програми, крім цього, треба ще вручну підключити бінарний файл відповідної бібліотеки. Наприклад, так: у Solution Explorer-і вибрати References, натиснути правою кнопкою миші, вибрати «Add Reference...», знайти потрібну бібліотеку (`System.Drawing` для `Image` та `Bitmap`, `System.Windows.Forms` для `MessageBox`, тощо) у списку «Framework», вибрати (ліворуч повинна з'явитися галочка (checkbox)), підтвердити цей вибір через «OK».