## RODOsnych świąt

W tym roku wprowadzenie RODO przysporzyło wielu firmom kłopotów i konieczności zmiany systemów. Okazuje się, że ten problem dotknął również św. Mikołaja. Część dzieci (Mikołaj podejrzewa jednak, że to nadgorliwi rodzice, którzy nie rozumieją idei RODO) przysłała listy z zaszyfrowanym podpisem. Mikołaj nie wie przez to, do którego dziecka powinien trafić prezent opisany w liście. Wprawdzie Mikołaj posiada listę identyfikatorów dzieci i potrafi je zdekodować na ich imiona (metoda *DecodeChildName*), ale okazuje się, że sprytne dzieci/rodzice pomnożyli te identyfikatory przez jakąś dużą liczbę pierwszą przez co Mikołaj musi dokonać faktoryzacji (rozkładu na czynniki pierwsze), aby uzyskać faktyczny identyfikator dziecka. Na szczęście Mikołaj odkrył dwie zasady: a) identyfikator dziecka jest zawsze liczbą pierwszą, b) liczba pierwsza, przez którą pomnożony został identyfikator jest większa od identyfikatora. Jednak Mikołaj nadal obawia się, że nie będzie w stanie samemu podołać temu zadaniu, więc chce zlecić jego wykonanie elfom. Aby ułatwić sobie zadanie poprosił obiecującego programistę (Ciebie!) o pomoc. W zamian obiecał przynieść Ci w tym roku coś ekstra pod choinkę.

Oto jak Mikołaj wyobraża sobie działanie systemu:
W kolejnych dniach będą przychodzić listy (klasa *Letters*) z unikalnym numerem identyfikacyjnym listu (*id*), informacją, w którym dniu przyszedł (*receivedDay*) oraz podaną liczbą (*number*), która koduje podpis dziecka.
W każdym dniu Mikołaj bierze listy, które przyszły w tym dniu i wysyła je do fabryki (klasa *ChristmasFactory*) po czym idzie spać (metoda *GoSleep*). W tym samym czasie fabryka rozpoczyna pracę nad przesłaną paczką listów (metoda *ManageLetters)*. Dla każdego z nich równolegle w oddzielnym wątku szuka liczby najmniejszej liczby pierwszej, przez którą dzieli się *number* z listu (metoda *NumberFactorization*). Po przetworzeniu wszystkich listów z paczki fabryka zwraca wyniki Mikołajowi. Wyniki te następnie po odkodowaniu imienia dziecka są zapisywane do listy *decodedLetters*. 24 grudnia nowe listy już nie przychodzą. W tym dniu Mikołaj czeka na przeliczenie wszystkich listów po czym rusza w drogę z okrzykiem: „Ho! Ho! Ho!”.

Uwagi:
- praca fabryki nie może zakłócać snu Mikołaja oraz sen Mikołaja nie wpływa na pracę fabryki
- fabryka może równolegle pracować nad listami z kilku dni, tzn. fabryka rozpoczyna pracę nad paczką listów z kolejnego dnia niezależnie od tego czy wszystkie poprzednie zostały zakończone
- liczby mogą być zbyt duże, aby przechowywać je w zmiennych typu int – użyj long
- kolejność komunikatów może nieznacznie różnić się od tych w załączonym pliku
- Etapy II, III oraz IV możesz zrealizować w dowolnej kolejności

Zanim Twój system zostanie wdrożony produkcyjnie Mikołaj chciałby go przetestować. Przygotował więc paczkę 10 przykładowych listów, które mogłyby przyjść w dniach 19 – 23 grudnia. Oczekuje od Ciebie uzupełnienia metody *PrepareForChristmas*,. Aby przyśpieszyć testy systemu sen Mikołaja skracamy do 1 sekundy.

**Etap I (2 pkt.)**
Napisz metody:

- *NumberFactorization* w klasie *ChristmasFactory,* która przyjmie jedną liczbę typu long i zwróci najmniejszą liczbę pierwszą przez którą dzieli się ta liczba.
- *ManageLetters* w klasie *ChristmasFactory,* która przyjmie listę listów, dla każdego z nich równolegle rozpocznie obliczenia, a w momencie ich zakończenia zwróci wynik będący listą par: identyfikator listów, znaleziona liczba pierwsza.
- *SendLettersToFactory* w klasie *SantaClause*, która przyjmie listę listów i kolejno:

- wpisze "Wysłanie listów do fabryki z {day} grudnia"
- wywoła metodę *ChristmasFactory.ManageLetters*
- po otrzymaniu rezultatu wypisze "Przetworzono wszystkie listy z {day} grudnia"
- doda zdekodowane przy pomocy metody *DecodeChildName* wyniki do listy *decodedLetters*

Odkomentuj odpowiednie wywołania w metodzie *PrepareForChristmas*.

Uwaga: Dwie z powyższych metod powinny być oznaczone jako async (i oczywiście używać await).

**Etap II (1 pkt.)**
Okazało się, że czasami zakodowany identyfikator jest błędny. W tym celu poprosił o stworzenie następującego systemu obsługi wyjątków:
Zdefiniuj własny wyjątek *IncorrectFactorizationException*, który będzie zawierał enum z przyczyną wyjątku. Dodaj dwie możliwe przyczyny:
- *IsPrime* – występuje w przypadku, gdy przekazana liczba jest liczbą pierwszą (nie można dokonać jej dalszej faktoryzacji)
- *MoreThan2Factors* – występuje, gdy przekazana liczba jest iloczynem więcej niż dwóch liczb pierwszych (wskazówka: wynik dzielenia przez liczbę pierwszą nie daje liczby pierwszej)

W metodzie *NumberFactorization* dodaj mechanizm rzucania wyjątków z odpowiednim parametrem w wyżej wymienionych przypadkach.
Dodaj obsługę wyjątków (łapanie ich z klauzulą *when*). Dla *IsPrime* zwróć po prostu przekazaną do metody liczbę. W przypadku wystąpienia *MoreThan2Factors* zwróć 3.

**Etap III (1 pkt.)**
Mikołaj co jakiś czas chciałby wiedzieć na jakim etapie przetwarzania jest wskazany przez niego list. Napisz metodę *WhatAbout*, która przyjmie identyfikator listu (id) i wypisze jeden z 3 komunikatów:
"List {id} nie dotarł jeszcze do fabryki" – jeśli fabryka nie otrzymała jeszcze listu wskazanego dziecka.
"List {id} jest przetwarzany przez fabrykę" – jeśli obliczenia dla listu tego dziecka nadal trwają.
"List {id} został przetworzony" – jeśli obliczenia zostały zakończone i ich wynik znajduje się w liście *decodedLetters*.
Uwaga: zapis do listy w metodzie *SendLettersToFactory* oraz jego odczyt z niego w metodzie *WhatAbout* mogą nastąpić jednocześnie, więc aby zapobiec nieprzewidzianym sytuacjom z tym związanym użyj *lock*, aby zablokować listę na czas odczytu/zapisu.

Dodaj wywołanie tej metody z losowym identyfikatorem listu (spośród tych, które występują) po każdym śnie Mikołaja (po wywołaniu metody *GoSleep* w *PrepareForChristmas*)

**Etap IV (1 pkt.)**
Mikołaj wpadł na pomysł jak przyspieszyć obliczenia. Zamiast sytuacji, w której jeden elf sprawdza po kolei podzielność i pierwszość każdej liczby z wskazanego przedziału, Mikołaj zaproponował, aby kilka elfów na raz sprawdzało różne dzielniki.
Napisz metodę *NumberFactorizationParallel* będącą modyfikacją metody *NumberFactorization*, która korzysta z Parallel.For. Zwrócony wynik przez to polecenie zawiera LowestBreakIteration, które może okazać się przydatne. Wyniki zwracane przez tę metodą nie powinny różnić się od tych z *NumberFactorization.* Zamień wywołanie *NumberFactorization* na *NumberFactorizationParallel* i sprawdź to.