TP N° 6
Exception

Objectifs

– Réviser la notion d'exception, ainsi que la manière dont on l'appelle (throw-try-catch).

1. Contrôle de la saisie d'un nombre entier compris entre 1 et n

Un programme qui demande saisir une borne inférieure et une borne supérieure. Puis demande de saisir un nombre compris entre cette borne inférieure et cette borne supérieure.

2. Introduction

Ecrire la classe Erreur vue dans le cours et la tester

Ecrire un programme qui gère la saisie d'un entier avec des exceptions

Ecrire un programme qui gère la division par zéro avec des exceptions

3. Contrôle de division par zéro

Nous allons reprendre l'exercice sur les nombres rationnels. Maintenant que l'on sait comment les manipuler, il faut prévoir ce qui se passe au cas où on voudrait diviser un nombre rationnel par un autre nombre rationnel nul.

Modifier l'opérateur de division pour qu'il lève une exception en cas de division par zéro.
Essayer plusieurs types d'exceptions (type primitif, classe...).

Modifier le programme principal pour qu'il intercepte les exceptions générées par une mauvaise division.

4. Gestion de rationnels

Nous allons reprendre l'exercice sur les nombres rationnels. Maintenant que l'on sait comment les manipuler, il faut prévoir ce qui se passe au cas où on voudrait diviser un nombre rationnel par un autre nombre rationnel nul.

Modifier l'opérateur de division pour qu'il lève une exception en cas de division par zéro.
Essayer plusieurs types d'exceptions (type primitif, classe...).

Modifier le programme principal pour qu'il intercepte les exceptions générées par une mauvaise division.

5. Le coursier simple

Nous allons simuler le travail d'un coursier dans une grande entreprise. Ce coursier passe tous les matins dans les bureaux et récupère le courrier à affranchir. Ce courrier est de deux types, avec une différence au niveau du processus d'affranchissement:

– les lettres, qui peuvent être ordinaires (0,5 euro) ou urgentes (+0,3 euro). On convient que la lettre représente 1 unité de volume.
– les colis, dont l'affranchissement dépend du volume (0,5 euro / unité de volume).

Le coursier place les courriers dans un grand sac (de capacité fixée à sa création). De retour dans son bureau, il place le sac dans une machine a affranchir (qui affranchit automatiquement les courriers contenus dans le sac en fonction de leur type). On demande de modéliser la tournée du coursier en

– créant un sac et quelques courriers;
– plaçant les courriers dans le sac;
– affranchissant le sac;
– affichant le tarif des différents courriers.

6. Le coursier au sac dynamique

Nous allons reprendre l'exercice du coursier et ajouter une classe Coursier, afin de lui conférer un sac de volume toujours limité, mais de quantité cette fois illimitée. Le sac demeure donc un tableau de courriers, pour lequel on doit pouvoir être renseigné à tout moment sur la quantité et le volume de courriers qu'il contient. A la création, le sac est vide.

On doit pouvoir calculer l'affranchissement total du sac, afficher son contenu, retourner le courrier d'indice donné, et ajouter un courrier à la fin du sac, à condition qu'il y ait encore de la place. Pour l'avant-dernière méthode en particulier, on lèvera une exception au cas où l'entrée demandée est trop grande par rapport à la quantité courante du sac. Et pour la méthode d'ajout, on lèvera des exceptions au cas où un tel ajout dépasserait la capacité du sac (en quantité comme en volume).

Ecrire un programme principal qui permet de tester cette classe, et surtout d'intercepter les exceptions générées par une mauvaise demande de renseignement, ou un mauvais ajout.

7. Pile File

Ecrire deux classes génériques *Pile* et *File* qui permettent de créer et de manipuler respectivement des piles et des files d'éléments de type quelconque.

Une pile se manipule selon le principe “dernier arrivé, premier servi” (Last In First Out), à savoir que le sommet de la pile (le seul élément auquel on ait accès) est le dernier élément rentré. Empiler consiste à ajouter un élément en fin de pile (lequel deviendra le nouveau sommet). Dépiler consiste à supprimer le sommet (qui est remplacé par l'ex-avant-dernier élément).

Une file se manipule selon le principe “premier arrivé, premier servi” (First In First Out), à savoir que le sommet de la file (le seul élément auquel on ait accès) est le premier élément rentré. Enfiler consiste à ajouter un élément en fin de file (le sommet est donc inchangé).

Défiler consiste à supprimer le sommet (qui est remplacé par l'ex-deuxième élément).

Ecrire un programme principal permettant de créer, de remplir et de vider des piles et des files d'éléments divers (entiers, réels...).

8. Le problème de Josephus avec une file standard

Le problème à résoudre est inspiré d'une histoire attribuée à l'historien juif Flavius Josèphe (ou Josephus). Pendant la Première Guerre Judéo-Romaine (66-73 après J-C) à laquelle il a pris part activement, il a été pris au piège dans une caverne avec un groupe de 40 soldats cernés par des Romains. La légende dit que, préférant la mort à la capture, les soldats décidèrent de former un cercle et de tuer la troisième personne vivante rencontrée en suivant le parcours autour du cercle; ceci jusqu'à ce qu'il ne reste qu'une personne, celle-ci devant se suicider (d'autres versions affirment que chaque troisième personne du cercle devait se donner la mort sans “l'aide” de personne). Flavius Josèphe, pas très enthousiaste à l'idée de casser sa pipe, trouva rapidement la bonne place dans le cercle afin d'être le dernier survivant... et donc de sauver sa peau !

On demande de modéliser le “problème de Josephus” au moyen d'une file. Pour cela, on demande d'écrire une procédure qui prend en entrée les paramètres suivants :
– N : nombre de soldats dans le cercle.
– M : le nombre tel que chaque M-ième soldat vivant dans le parcours circulaire est tué.
La procédure commencera par créer une file contenant tous les soldats, chacun représenté par un numéro de 1 à N. Il faudra ensuite se servir de cette file pour afficher tous les soldats supprimés dans l'ordre où ils ont été supprimés, puis enfin le dernier soldat survivant (qui est la solution au problème).

Compléter le programme principal afin de résoudre le problème de Josephus pour plusieurs valeurs de M et N (dont les valeurs “historiques”).