# TP JAVA n°4
# Exceptions

## Exercice 1 : Division par zéro

1) Repartir de l'exemple traité en cours de division par zéro. Dans la classe **Main**, On saisira 2 entiers x et y au clavier, puis calculera z = x / y, en définissant y à 0. N'effectuer aucun traitement d'exception ici et constater le résultat.
2) Reprendre le traitement du 1) dans un bloc **try**, et y adjoindre un bloc **catch** traitant un objet de type **ArithmeticException**. Avant la sortie du **main()**, afficher un message montrant que vous avez traité votre exception sans rendre la main à la machine virtuelle Java.
3) Ecrire une classe **maDivParZero** héritant de **ArithmeticException**, ayant un **constructeur** redéfinissant le message d'erreur proposé par la classe **ArithmeticException** en le personnalisant. Dans le bloc try, lancer une exception de type **maDivParZero** si le dénominateur y vaut 0. Définir ensuite 2 blocs catch selon 2 séquences :

- Dans la première séquence, le premier **catch** intercepte une erreur de type **maDivParZero** et le second une erreur de type **ArithmeticException**
- la seconde séquence inverse le placement des 2 blocs.

Constater à nouveau le résultat obtenu.

## Exercice 2 : Émission musicale

On revient ici sur l'exercice « Emission musicale » (TP_2). Dans cet exercice, lorsque l'on ajoutait une chanson à la playlist, on vérifiait si la durée totale de la playlist après insertion ne dépassait pas la durée de l'émission pour accepter l'insertion (en renvoyant un booléen). 1

1) Modifier ce cas particulier en lançant une exception définie par vos soins à la place. Les ajouts à la playlist devront se faire sous forme d'une boucle dont on ne sortira que si l'utilisateur le décide, et non si le cas particulier spécifié se produit.
2) *On peut ajouter autant de chanson que l'on veut dans la playliste mais c'est dans **passeTout()** qu'il ne faut pas dépasser la durée, attention la variable d'instance temps n'a plus le même sens. A faire avec exception.

# *Exercice 3 : Polygone convexe

**1 : Interface Polygone**

Créez une interface Polygone qui définira le contrat pour tous les types de polygones. Cette interface devra déclarer les méthodes suivantes :

- ArrayList<Point> getSommets(): Retourne la liste des sommets du polygone.
- double calculerPerimetre(): Calcule et retourne le périmètre du polygone.
- double calculerAire(): Calcule et retourne l'aire du polygone.

**2 : Classe abstraite PolygoneAbstrait**

Créez une classe abstraite PolygoneAbstrait qui implémente l'interface Polygone. Cette classe aura pour attribut protégé une ArrayList<Point>, une implémentation par défaut pour la méthode getSommets() et une implémentation abstraite pour la méthode calculerAire(). La méthode calculerPerimetre() sera implémentée ici, car le calcul du périmètre est le même pour tous les polygones.

**3 : Classe PolygoneConvexe héritant de PolygoneAbstrait**

Créez la classe PolygoneConvexe qui hérite de PolygoneAbstrait.

- **Constructeur et validation de la convexité:** Le constructeur de PolygoneConvexe prendra une ArrayList<Point> en paramètre. Au lieu de lever une exception directement, il appellera une méthode privée estConvexe() qui effectuera la vérification. Si estConvexe() retourne false, le constructeur lèvera l'exception.
- **Implémentation de estConvexe():** Un moyen simple est de vérifier qu'à chaque sommet on "tourne" dans la même direction. Cette méthode a ses limites, elle ne fonctionne pas avec les polygones étoilés par exemple. Bonus : trouvez un moyen plus robuste de vérifier qu'un polygone est convexe.
- **Implémentation de calculerAire():** Implémentez la méthode calculerAire() spécifique aux polygones convexes. Utiliser la méthode decomposerEnTriangles()
- **Méthode decomposerEnTriangles():** Ajoutez une méthode privée decomposerEnTriangles() qui décompose le polygone convexe en une liste de triangles (Qui sont des polygones convexes). Chaque triangle sera représenté par une liste de 3 Point. Cette méthode sera utile pour des calculs plus complexes ou des affichages graphiques. Pour une représentation graphique, voir ce lien.

**4 : Classe PolygoneRegulier héritant de PolygoneConvexe**

Créez une classe PolygoneRegulier qui hérite également de PolygoneConvexe. Un polygone régulier a tous ses côtés de même longueur et tous ses angles de même mesure.

- **Constructeur:** Le constructeur prendra le nombre de côtés et la longueur d'un côté en paramètres. Il devra ensuite générer la liste des sommets du polygone régulier.
- **Implémentation de calculerAire():** Implémentez la méthode calculerAire() spécifique aux polygones réguliers qui est plus simple est moins coûteuse que la méthode définie dans PolygoneConvexe.
- **Surcharge de toString():** Redéfinissez la méthode toString() pour afficher les informations spécifiques du polygone régulier (nombre de côtés, longueur d'un côté).

**Bonus:**
Réfléchissez à un moyen d’approximer pi en utilisant la classe PolygoneRegulier