TP4 – Gestion des erreurs avec les exceptions

PARTIE 1 : La classe Temps

1 – Classe Temps

Créez une classe appelée **Temps.**
Cette classe est destinée à vous permettre de créer des objets représentant du temps, mesuré en heures, minutes et secondes.

- Ajoutez à cette classe, 3 variables d'instance, qui seront 3 entiers destinés à contenir les heures, les minutes et les secondes. Ces variables doivent être déclarées **private**.
- Dotez la classe des méthodes "getters" et "setters" nécessaires pour accéder à ces variables.
- N'ajoutez pas de constructeur à la classe pour le moment.
- Dotez la classe d'une méthode **toString()** qui retourne une chaîne de caractères au format "HH:MM:SS". Par exemple, pour un objet **Temps** qui représente 1h 6mn 4s, **toString()** retournera "01:06:04".

2 – Classe principale main

Créez une autre classe, considérée comme la classe principale car elle contient la méthode **main()**.
Puis, écrivez le code permettant d'effectuer les actions suivantes :

- Déclarer 2 variables locales à la méthode **main()**, appelées **t1** et **t2**, destinées à représenter deux temps.
- Créer 2 instances de la classe **Temps** et les affecter aux variables **t1** et **t2**.
- Affecter des valeurs pour que **t1** représente l'heure limite jusqu'à laquelle un élève est admis en cours à la première séance de l'après-midi sans être marqué en retard (autrement dit : 13h30 et 45 secondes). **t2** représentera la même chose, mais pour la 2ème séance de l'après-midi (autrement dit : 15h15 et 45 secondes).
- Afficher ces 2 valeurs au format HH:MM:SS.

3 – Constructeur avec paramètres

On envisage de faire la même chose avec une variable **t3** pour représenter l'horaire correspondant à la 3ème séance de l'après-midi. Mais, le côté fastidieux du codage nous incite à mieux utiliser les ressources de la programmation orientée objet.

- Ajoutez à la classe **Temps**, un constructeur avec 3 paramètres : les heures, les minutes et les secondes.
  (Rappel : dans Eclipse, vous pouvez utiliser le menu "Generate Constructor using Fields…".)

- Certainement, des erreurs s'afficheront dans la classe principale. Cherchez ce qu'il faut rajouter pour ne plus avoir ces erreurs sans pour autant changer la classe principale.

- Enfin, ajoutez dans la méthode **main()**, les lignes qui permettent d'initialiser et d'afficher un objet **t3** correspondant à la 3ème séance de l'après-midi.

4 – Exception

Notre classe **Temps** a un défaut. En effet, rien n'empêche le programmeur de la classe principale d'indiquer un temps du type 15h 98mn 2750s.
Or, nous souhaitons que les nombres de minutes et de secondes soient obligatoirement compris entre 0 et 59 et que les nombres d'heures soient compris entre 0 et 23.

Vous allez mettre en place des mécanismes pour garantir que ces règles sont respectées en toutes circonstances.

- Dans les méthodes "setters", contrôlez la valeur passée en paramètre. Si elle ne correspond pas aux critères de validité, envoyez une exception du type **IllegalArgumentException**.
  Cette exception doit véhiculer un message du type "Nombre de minutes incorrect : 78" (en supposant que l'on a passé le nombre 78 à la méthode **setMinutes()**).
- Testez l'efficacité du mécanisme, en utilisant des valeurs incorrectes dans certaines instructions de la méthode **main()**.

Ce n'est pas encore parfait, car si vous avez modifié uniquement les setters, vous pouvez encore écrire dans la méthode **main()**, des instructions telles que :
t2 = new Temps(150, 250, 550);

Pour y remédier de façon élégante (sans ré-écrire des conditions avec des if), faites appel aux setters à l'intérieur du code du constructeur.

- Vérifiez que cette fois-ci, il est impossible d'introduire des valeurs incorrectes dans les variables d'un objet **Temps**.

5 – Formulaire de saisie

Comme vous l'avez remarqué, l'apparition d'une exception provoque l'arrêt immédiat du programme. C'est adapté, si on est en phase de débogage, mais ça ne l'est pas, s'il s'agit d'une erreur de saisie d'un utilisateur. L'action adéquate consisterait à lui demander de recommencer sa saisie.

Pour voir comment gérer cette situation, nous allons modifier notre programme pour permettre à l'utilisateur de saisir un temps au format HH:MM:SS. Voici le code que vous pouvez écrire :

```
String message = "Entrez un temps au format HH:MM:SS"; String reponse = "";
while( true ) {
	reponse = javax.swing.JOptionPane.showInputDialog( message, reponse );
	String[] valeurs = reponse.split(":");
	t1.setHeures( Integer.parseInt( valeurs[0] ) );
	t1.setMinutes( Integer.parseInt( valeurs[1] ) );
	t1.setSecondes( Integer.parseInt( valeurs[2] ) );
	break;
}
System.out.println(t1);
```

Pour le moment, la boucle **while** ne sert à rien car elle se termine par une instruction **break**. Elle n'est donc exécutée qu'une seule fois.

La méthode **split()** permet de découper la réponse au niveau des caractères "deux points" et retourne un tableau contenant les 3 sous-chaînes obtenues par ce découpage.

La méthode **Integer.parseInt()** permet de transformer une chaîne de caractères en entier.

Ce programme peut générer de nombreuses exceptions.
Testez-le en faisant une saisie correcte, puis des saisies qui génèrent des exceptions :
- 25:00:00 → heures incorrectes
- 12:99:00 → minutes incorrectes
- 12:30:xx → xx ne peut pas être converti en nombre
- 12:30 → le tableau **valeurs** ne comporte que 2 éléments

6 – Gestion des exceptions – 1ere expérimentation

L'objectif est de faire que, si une exception apparaît, on revient au début de la boucle. Par contre, s'il n'y a pas d'anomalie, on sort de la boucle et on affiche **t1**.

- Dans Eclipse, sélectionnez les 6 lignes de code contenues dans la boucle **while**. Puis, clic-droit et "Surround With > Try/catch Block".

Comme vous le voyez, un bloc **try/catch** a été généré. Mais, il ne traite qu'un seul type d'exceptions : **NumberFormatException**, c'est-à-dire celles qui sont générées par la méthode **Integer.paresInt()**.
Pour s'en rendre compte, il suffit de cliquer sur le mot **NumberFormatException**. Celui-ci passe en fond gris et les appels de la méthode **parseInt()** également. Ce qui vous permet de voir d'où proviennent les exceptions qui sont susceptibles d'être détectées.

Testez les valeurs suivantes :
- 20:00:xx
- 20:yy:xx
- 25:yy:xx

Vous constatez qu'à chaque fois, une seule anomalie est traitée : la première qui est détectée.

- Si c'est une anomalie de type **NumberFormatException**, on revient au début de la boucle, car :
  - L'exception est gérée par le bloc **catch**. Elle ne provoque donc plus l'arrêt du programme.
  - On ne passe plus par l'instruction **break**. On ne sort donc pas de la boucle.
- Si c'est une exception d'un autre type, comme elle n'est pas gérée, elle provoque l'arrêt immédiat du programme.

Dans les 2 cas, il y a affichage de lignes rouges dans la vue "Console" d'Eclipse. Ces affichages sont similaires, mais pas du tout identiques.

- Dans le cas de l'exception gérée, l'affichage est produit par l'instruction : **e.printStackTrace();**

Il suffirait de supprimer cette instruction pour que les lignes rouges ne soient plus affichées.
Cet affichage commence directement par le nom de l'exception écrit en bleu.

- Dans le cas d'une exception non gérée par un bloc **catch**, le programme s'arrête brutalement et génère des lignes rouges dont la première commence par
  Exception in thread "main"
  Il n'est pas possible d'empêcher cet affichage en cas d'une exception non gérée. Ou plutôt, si on veut l'empêcher, il faut gérer l'exception en ajoutant un bloc **catch** supplémentaire.

Faites-le en dupliquant les 2 lignes du bloc **catch** de façon à obtenir le résultat suivant :

```
} catch ( NumberFormatException e ) {
      e.printStackTrace();
} catch ( IllegalArgumentException e ) {
      e.printStackTrace();
}
```

Constatez que, à présent, la saisie d'une valeur telle que "25:yy:xx" ne provoque plus l'arrêt immédiat du programme.

7 – Gestion des exceptions – 2ème expérimentation

Pourquoi Eclipse a-t-il généré tout seul le bloc catch pour l'exception de type **NumberFormatException** et pas pour celle de type **IllegalArgumentException** ?

D'ailleurs, si vous cliquez sur le mot **IllegalArgumentException**, les appels des méthodes **setHeures()**, **setMinutes()** et **setSecondes()** ne sont pas mis sur fond gris. Ce sont les appels de la méthode **paresInt()** qui sont sur fond gris.

Ce dernier point s'explique par le fait que les exceptions du type **NumberFormatException** sont une sous-catégorie au sein des exceptions de type **IllegalArgumentException**.

- Supprimez les deux lignes
  } catch (NumberFormatException e) {
  e.printStackTrace();
- Vérifiez que la saisie d'une valeur telle que "12:yy:xx", ne provoque pas l'arrêt du programme.

En effet, comme une **NumberFormatException** est une sorte de **IllegalArgumentException**, son apparition est à présent gérée par le bloc **catch** restant.
Vous pouvez vérifier dans les lignes rouges que c'est bien une exception de type **NumberFormatException** qui a été détectée.

Le principe est que, s'il y a plusieurs blocs **catch**, c'est le premier qui correspond à une catégorie compatible avec l'exception qui est exécuté. Les autres blocs **catch** sont ignorés. Si aucun bloc **catch** n'est compatible avec l'exception, celle-ci est envoyée au niveau supérieur, c'est-à-dire à l'appelant de la méthode en cours d'exécution. Comme nous sommes dans la méthode **main()**, c'est la machine virtuelle Java qui la reçoit, ce qui provoque l'arrêt du programme.

Il existe une catégorie d'exceptions encore plus large : celle qui regroupe tous les types d'exceptions possibles. Elle s'appelle **Exception** (tout court).

- Dans le bloc **catch**, remplacez **IllegalArgumentException** par **Exception**.

A présent, toutes les exceptions possibles sont gérées, y compris celle qui est envoyée lorsqu'on appuie sur le bouton "Annuler" (Nous avons oublié de tester si la réponse vaut **null** pour gérer le cas du bouton "Annuler"). On ne peut donc pas sortir du programme. Arrêtez-le en appuyant sur le bouton carré rouge.

8 – Gestion des exceptions – 3ème expérimentation

Ça peut être pratique, dans certains cas, d'utiliser la catégorie **Exception**. Mais, la plupart du temps, on souhaite programmer des comportements différents en réaction à chaque type d'exceptions. Dans ce cas, il est indispensable de définir plusieurs blocs **catch**.

Revenons à notre question : « Pourquoi Eclipse a-t-il généré tout seul le bloc **catch** pour l'exception de type **NumberFormatException** et pas pour celle de type **IllegalArgumentException** ? »

Pour y répondre effectuez les manipulations suivantes :

- Dans la boucle **while**, supprimez la ligne où se trouve le mot **try** et les 3 lignes du bloc **catch**. Ainsi vous êtes revenus à la situation de départ.
- Allez dans le classe **Temps** et observez le code de la méthode **setHeures()**. Celle-ci envoie une exception en cas de valeur incorrecte.
  Mais le code d'une méthode est considéré comme "privé". Il n'est pas visible depuis l'extérieur de la classe. La seule chose qui soit publique, c'est la déclaration de la méthode – c'est-à-dire sa première ligne – à condition qu'elle commence par la mot **public**.
  Donc si on veut rendre publique l'information que cette méthode est capable d'envoyer une exception, il faut l'indiquer à cet endroit-là.
- Modifiez la déclaration de la méthode **setHerues()** de façon à ce qu'elle devienne :
  public void setHeures(int heures) throws IllegalArgumentException { (Il ne faut pas modifier le reste de la méthode.)
  Cela permet à l'environnement extérieur de savoir que cette méthode peut envoyer des exceptions du type **IllegalArgumentException**.
- Modifiez de la même façon, les méthodes **setMinutes()** et **setSecondes()**.
- Enregistrez les modifications de la classe **Temps**, puis revenez à la classe principale.
- Comme précédemment, sélectionnez les 6 lignes de code contenues dans la boucle **while**. Puis, clic-droit et "Surround With > Try/catch Block".

Cette fois-ci, vous devez avoir 2 blocs **catch**. Si, dans le 2ème, vous cliquez sur le mot **IllegalArgumentException**, les appels des méthodes **getHeures()**, **getMinutes()** et **getSecondes()** doivent être mis sur fond gris.

9 – Message d'anomalie

A présent, la saisie de valeurs incorrectes ne provoque plus l'arrêt brutal du programme. Cependant, l'affichage des lignes rouges n'est pas très convivial pour un utilisateur non informaticien.

- Faites que les lignes rouges ne soient plus affichées en cas d'exceptions gérées par un bloc **catch**.
- A la place, faites que la variable **message** contienne un texte compréhensible par l'utilisateur.
  - Dans le cas de **IllegalArgumentException**, vous pouvez récupérer le message que véhicule l'exception puisque c'est vous qui l'avez généré et qu'il est en français.
  - Pour **NumberFormatException**, le message transporté par l'exception est en anglais. Mettez plutôt dans la variable **message** un texte du type "Valeur non numérique : xxxxxxxx" (où xxxxxxxx représente la chaîne saisie par l'utilisateur).
- Vérifiez le fonctionnement du programme.

10 – Gestion du débordement de tableau

Il reste un cas d'erreur à traiter : c'est celui où l'utilisateur saisit une chaîne incomplète du type "15:25" où les secondes sont absentes.

En effet, dans ce cas, la méthode **String.split()** crée un tableau avec 2 éléments seulement et lorsqu'on exécute la ligne

```
t1.setSecondes( Integer.parseInt( valeurs[2] ) );
```

cela provoque une erreur car l'expression **valeurs[2]** recherche un 3ème élément dans un tableau qui n'en contient que 2.

- Provoquez l'erreur et regardez dans les lignes rouges qu'elle est l'exception qui est envoyée dans ce cas.
- Ajoutez un bloc **catch** pour gérer ce type d'exception. Faites que l'utilisateur reçoive un message du type "Format obligatoire HH:MM:SS".
- Vérifiez que le programme ne plante plus en cas de saisie incomplète.

11 – Ordre des blocs catch

Normalement, à la fin de la section **try**, vous avez 3 blocs **catch**.
L'ordre dans lequel sont positionnés les blocs **catch**, a une importance.

- Pour se rendre compte, modifiez cet ordre de telle façon que le bloc correspondant à **IllegalArgumentException** soit le premier des 3.
- Enfin, dupliquez l'un des blocs **catch**, de façon à ce qu'il y ait deux blocs **catch** correspondant au même type d'exception.

PARTIE 2 : calculs sur les temps

On souhaite utiliser nos objets **Temps** pour faire des calculs, comme par exemple additionner 2 durées ("Combien vaut 3h 45mn 59s + 4h 57mn 42s ?").

Adaptation de la classe Temps

Vous allez ramener 2 modifications principales à la classe Temps.

Décodage d'une chaîne "HH:MM:SS"

- Ajoutez à la classe **Temps**, un nouveau constructeur qui prenne en paramètre une chaîne de caractères. Cette chaîne est censée contenir la valeur d'un temps au format HH:MM:SS.
- Le constructeur doit donc décoder cette chaîne pour en extraire la valeur des heures, des minutes et des secondes.
- Vous pouvez vous inspirer de ce qui a été fait au cours de la PARTIE 1, mais il y a une différence importante. Dans l'exercice, on gérait l'affichage d'un message pour informer l'utilisateur. Ici, dans un constructeur, on n'est pas censé faire d'affichage. En cas d'anomalie, le comportement adéquat consiste à envoyer une exception au programme appelant. (voir la contrainte définie au point suivant).
- Il vous est demandé de ne pas envoyer directement les exceptions **NumberFormatException**, **IllegalArgumentException** ou **IndexOutOfBoundsException**.
  En effet, leur envoi dévoilerait la façon dont est programmé le constructeur. Or, cela ne regarde pas l'environnement extérieur, car vous souhaitez être libre de pouvoir changer ce code ultérieurement sans que l'extérieur en soit informé.
- A la place, il vous est demandé d'envoyer une exception de type **Exception** (tout court) en lui faisant transporter un message qui pourra être affiché à l'intention de l'utilisateur par le programme principal.

Nombre total de secondes

Une solution pratique, permettant de faire toutes sortes d'opérations, consiste à convertir un temps en un nombre entier de secondes (1 heure est représentée par 3600 secondes, etc.). Ensuite, on peut effectuer des calculs sur ce nombre entier de secondes, puis reconvertir le résultat en heures, minutes et secondes.

- Ajoutez à la classe **Temps**, une méthode **getTotalSecondes()** qui retourne la valeur du temps exprimée sous forme d'un nombre entier de secondes.
- Vérifiez que les valeurs calculées par cette méthode sont exactes, en l'appelant à partir d'une autre classe et en faisant afficher son résultat pour des valeurs connues. Exemples :
  1h 23mn 20s = 5 000 secondes
  15h 25mn 55s = 55 555 secondes

- Ensuite, ajoutez un nouveau constructeur, prenant en paramètre un entier, qui sera interprété comme la valeur du temps exprimée en nombre total de secondes. Cette valeur sera convertie en heures, minutes et secondes de façon à initialiser les variables membres de l'objet.

Astuce : pour calculer la valeur de la variable **minutes** à partir du nombre total de secondes, on peut utiliser la formule :

```
minutes = ( totalSecondes % 3600 ) / 60 ;
```

A vous de trouver les formules pour obtenir les valeurs des variables **heures** et **secondes**.

- Pensez à gérer les cas d'anomalies. En effet, si on écrit

```
Temps t = new Temps( 99999 ) ;
```

cela correspond à un temps supérieur à 24 h, ce qui est incorrect par rapport à la définition de notre classe.
Le constructeur doit donc envoyer une exception, si l'entier qui lui est passé en paramètre correspond à une durée supérieure ou égale à 24 heures.

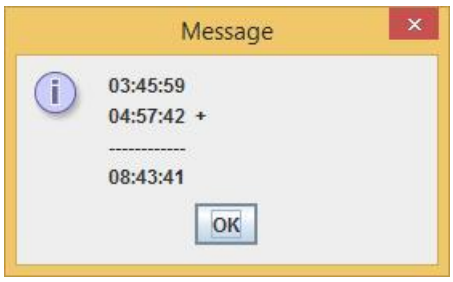
Calculatrice

Écrivez un programme ayant le fonctionnement suivant :

- Affichage d'une 1ère boîte de dialogue qui demande à l'utilisateur d'entrer un 1er temps au format HH:MM:SS.
  Attention ! vous devez utiliser le constructeur de la classe **Temps** pour décoder la réponse de l'utilisateur (contrairement à l'exercice de la PARTIE 1 où ce décodage était fait par la classe principale).
- En cas de saisie erronée, la boîte de dialogue est ré-affichée avec un message explicatif et ce, jusqu'à ce que la saisie soit correcte ;
- Ensuite, affichage d'une 2ème boîte de dialogue, identique à la première, pour demander à l'utilisateur de saisir un 2ème temps.
- Puis, les 2 temps sont convertis en nombre total de secondes de façon à pouvoir les additionner.
- Un 3ème objet **Temps** est créé à partir du résultat de cette addition.
- Une dernière boîte de dialogue est affichée. Elle contient le résultat de l'opération et devrait avoir une apparence similaire à celle présentée ci-contre.

Conseil : Pour afficher une boîte de message, vous pouvez utiliser la syntaxe suivante :

```
javax.swing.JOptionPane.showMessageDialog( null, message );
```



Contrôle qualité

Non redondance du code :

- Pour la saisie des deux temps à additionner, le programme utilise deux boîtes de dialogue identiques. Bien entendu, il faut absolument éviter d'avoir du code redondant.
  Si nécessaire, créez une méthode statique pour éviter d'avoir du code en double.

Ergonomie :

- Pensez à gérer le cas où l'addition va produire un temps supérieur ou égal à 24 h. Évitez que cela se traduise par le plantage du programme et l'affichage de lignes rouges.
  A la place, faite afficher une boîte de dialogue similaire à celle présentée ci-contre.
- Pensez à gérer le cas où l'utilisateur appuie sur le bouton "Annuler" dans l'une des deux boîtes de saisie. Dans ce cas, le programme doit s'arrêter immédiatement sans afficher de message d'anomalie.

Bonnes pratiques :

N'oubliez pas d'appliquer également les règles suivantes :
- Nommage des variables et des méthodes (majuscules / minuscules, ...).
- Lisibilité du code (indentation = Ctrl+I, ...) et des commentaires.