



Android TP 5 : Les exceptions et les intentions.

Plusieurs notions conceptuelles dans ce TP liées au multitâches.

Nous mettrons d'abord en œuvre le procédé, relativement simple, d'exception, vue, en principe, dans le cours sur JAVA.

Nous essaierons ensuite de comprendre comment les applications et le système ANDROID échangent des messages via les intentions (« intent »).

On rappelle les 2 raccourcis clavier bien utiles : CTRL espace pour l'auto-complétion¹ et CTRL p pour voir les paramètres pour une fonction.

→ Le texte du TP est émaillé de questions à rédiger. Elles sont précédées d'une *. Ceci sera fait sur un document à transmettre à l'enseignant à la fin de la séance en précisant bien le paragraphe concerné.

→ Jeter un œil aux annexes de ce texte avant de commencer.

→ Pour placer les composants on utilisera les modèles étudiés lors des TP précédents.

1 Les exceptions

Lors de l'exécution d'un programme, quel que soit le langage ou la plateforme utilisés, certaines erreurs conduisent à l'arrêt brutal du programme. En langage courant, on dit qu'il y a « plantage ». Ceci se produit, par exemple, lors de la division d'un nombre par 0. Dans une telle situation, le microprocesseur génère une « exception » pouvant être traitée. Le langage JAVA dispose d'outils pour intercepter les exceptions et les traiter. Le but de ce paragraphe est d'étudier ce mécanisme.

Avant cela, nous devons mettre en place un projet répondant au cahier des charges suivant : **l'interface permet de saisir deux nombres entiers et d'afficher le résultat de la division entre ces 2 nombres. Ce résultat s'affiche quand on appuie sur le bouton « OK » (Figure 1).**

1.1 Création du projet de test

→ Créer le projet TP5_1, situé dans le dossier TP5_1 du dossier de travail.

→ Créer l'interface utilisateur comme sur la figure 1. Bien la comprendre et vérifier qu'elle correspond au cahier des charges.

→ Ecrire le programme comme ci-dessous (en l'analysant et en le comprenant bien entendu). On remarquera le traitement assez lourd des chaînes de caractères. Tout élément sur l'IU est une chaîne de caractères et doit être converti pour être traité comme un nombre.

→ Compiler et tester le programme. Fournir une copie d'écran du résultat.

```
public class MainActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        // recuperation des IDs  
        EditText dividende = (EditText) findViewById(R.id.dividende);
```



```
EditText diviseur = (EditText) findViewById(R.id.diviseur);  
TextView resultat = (TextView) findViewById(R.id.resultat);  
Button boutonOK = (Button) findViewById(R.id.ValideResultat);
```

//Calcul et affichage de l'opération de division suite à l'appui sur le bouton "OK"

```
boutonOK.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        // variable pour récupérer la valeur du dividende  
        int valeur_dividende;  
        // variable pour récupérer la valeur du diviseur  
        int valeur_diviseur;  
  
        // recuperation des valeurs, conversion des chaînes de caractères en nombre  
        valeur_dividende=Integer.parseInt(dividende.getText().toString());  
        valeur_diviseur=Integer.parseInt(diviseur.getText().toString());  
  
        //Calcul et affichage du resultat, transformation du nombre en chaîne  
        int valeur_resultat;  
        valeur_resultat = valeur_dividende/valeur_diviseur;  
        resultat.setText(String.valueOf(valeur_resultat));  
    }  
});
```

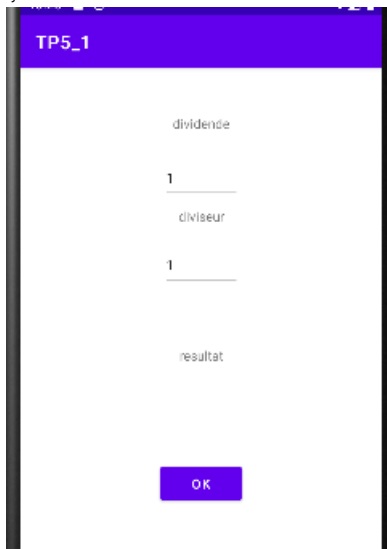


Figure 1 - Interface utilisateur TP5_1

On pourra utiliser le code XML ci-dessous :

```
<EditText  
    android:id="@+id/dividende"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:autofillHints=""  
    android:ems="5"  
    android:inputType="number"  
    android:minHeight="48dp"  
    android:text="1"  
    android:textSize="14sp"  
    app:layout_constraintBottom_toBottomOf="parent"  
    app:layout_constraintHorizontal_bias="0.5"  
    app:layout_constraintLeft_toLeftOf="parent"
```



```
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintVertical_bias="0.2"
tools:ignore="LabelFor,DuplicateSpeakableTextCheck" />
```

```
<TextView
    android:id="@+id/messageDividende"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="dividende"
    android:textSize="14sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintHorizontal_bias="0.5"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.1"
/>
```

```
<EditText
    android:id="@+id/diviseur"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:autofillHints=""
    android:ems="5"
    android:inputType="number"
    android:minHeight="48dp"
    android:text="1"
    android:textSize="14sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintHorizontal_bias="0.5"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.4"
    tools:ignore="LabelFor" />
```

```
<TextView
    android:id="@+id/messageDiviseur"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="diviseur"
    android:textSize="14sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintHorizontal_bias="0.5"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.3" />
```

```
<TextView
    android:id="@+id/resultatCalcul"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:minHeight="48dp"
    android:textSize="14sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintHorizontal_bias="0.5"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"
```



```
app:layout_constraintVertical_bias="0.7" />
```

```
<TextView
    android:id="@+id/resultat"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintHorizontal_bias="0.5"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.6"
    android:text="resultat"
    android:textSize="14sp" />
```

```
<Button
    android:id="@+id/ValideResultat"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="ok"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintHorizontal_bias="0.5"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintVertical_bias="0.9"
    app:layout_constraintTop_toTopOf="parent" />
```

1.2 Les exceptions

→* Qu'apporte l'attribut « `android:inputType="number"` » dans les deux « EditText » ? Est-il, par exemple, possible de saisir des lettres à la place des chiffres ?

→ Vérifier que le programme « se plante » quand le diviseur vaut 0. Fournir une copie d'écran.

Nous allons maintenant utiliser l'outil de traitement des exceptions disponible en JAVA :

→ Repérer les lignes de programme ci-dessous :

```
valeur_resultat = valeur_dividende /valeur_diviseur;
resultat.setText(String.valueOf(valeur_resultat));
```

→ Les remplacer par ceci et vérifier. Fournir une copie d'écran.

```
try {
    valeur_resultat = valeur_dividende/valeur_diviseur;
    resultat.setText(String.valueOf(valeur_resultat));
}
catch (Exception e) {
    resultat.setText("Division par 0");
}
```

L'outil logiciel `try{...} catch (exception {...})` permet d'anticiper un dysfonctionnement (appelé une exception) et de prendre en charge son traitement. Un certain nombre d'exception sont répertoriées.

Il est possible de visualiser dans la fenêtre « logcat » la nature de l'exception. Ceci sera très utile lors de la mise au point des programmes.



→ Ecrire maintenant comme ci-dessous le traitement de l'exception, compiler, exécuter et observer la fenêtre « logcat ». La nature de l'exception doit s'afficher. Fournir la copie d'écran dans le document à rendre.

```
catch (Exception e) {  
    resultat.setText("Division par 0");  
    Log.i(mon exception, "texte quelconque", e);  
}
```

Les exceptions sont hiérarchiques, et « Exception » est le parent de plus haut niveau existant. C'est à dire que « catch(Exception e) » permettra à notre code de gérer et attraper toutes les exceptions existantes de la même manière.

Si nous voulons par exemple juste attraper l'exception « division par 0 » sans s'occuper du reste, il conviendra d'attraper l'exception correspondante. En JAVA, il n'existe pas d'exception propre à « division par 0 », mais une exception globale pour les erreurs arithmétiques nommée « ArithmeticException ». Le code deviendra alors :

```
try {  
    valeur_resultat = valeur_dividende/valeur_diviseur;  
    resultat.setText(String.valueOf(valeur_resultat));  
}  
catch (ArithmeticException e) {  
    resultat.setText("Division par 0");  
    Log.i(mon exception, "texte quelconque", e);  
}
```

Il est également possible de gérer plusieurs types d'exception en chaînant les block « catch » de cette manière :

```
try {  
    valeur_resultat = valeur_dividende/valeur_diviseur;  
    resultat.setText(String.valueOf(valeur_resultat));  
}  
catch (ArithmeticException e) {  
    resultat.setText("Division par 0");  
}  
catch (Exception e) {  
    resultat.setText("Erreur inconnue");  
    Log.i(mon exception, "texte quelconque", e);  
}
```

Les objets JAVA et également ceux d'Android génèrent toutes sortes d'exceptions en permanence afin de permettre au programmeur de faire de la gestion d'erreur, en particulier liées à la saisie de l'utilisateur.

Par exemple, la méthode « parseInt » de la classe « Integer » que nous utilisons dans notre code pour convertir en entier la chaîne saisie par l'utilisateur, renvoie l'exception « NumberFormatException » lorsque qu'il est impossible de convertir la chaîne de caractères en entier (par exemple, lorsqu'elle contient des lettres).

En langage JAVA, la classe Throwable permet de déclarer des objets qui seront traités par la machine virtuelle JAVA comme des erreurs ou des exceptions. Exceptions est une des deux sous-classes de Throwable. ANDROID STUDIO fournit des outils pour prendre en charge de nombreuses sources d'exception.

La liste des exceptions (environ 370) prises en charge par ANDROID STUDIO est disponible là :

<https://developer.android.com/reference/java/lang/Exception.html>

→ *Retrouver dans la liste l'exception arithmétique. Faire une copie d'écran et la joindre au document à rendre.

→ Lire l'annexe 1, qui liste les opérations provoquant une exception arithmétique



1.3 Conclusion

→ *Ecrire, dans le document à rendre, une phrase précisant le rôle de la partie de code écrite entre les accolades de la partie « try » et de la partie « catch » de l'exception.

→ *A-t-on intérêt à utiliser dès que possible cet outil ? Est-ce difficile ? Que faut-il vérifier ?

2 Atelier – mise en œuvre de l'exception « débordement de tableau »

Nous allons, dans cet atelier, utiliser le projet TP5_1 pour provoquer une exception « débordement de tableau » et la gérer.

→ *Rechercher dans la documentation la signification des exceptions « `ArrayIndexOutOfBoundsException` » et expliquer son fonctionnement

→ Déclarer dans le programme un tableau de 3 nombres :

```
int tab[]={1,2,3};
```

→ Lorsque l'utilisateur appuie sur le bouton « OK », le code ci-dessous doit s'exécuter :

```
for (int i=0; i<FIN; i++) tab[i]=0;
```

La constante FIN prendra pour valeur 3 puis 4. Ce dernier cas va provoquer un débordement de tableau (accès à un quatrième élément pour lequel la place dans la mémoire n'a pas été réservée).

→ Lorsqu'il n'y a pas d'exception, le message « ça marche » doit s'afficher dans un « Textview » placé en dessous de celui nommé « resultat ». Dans le cas contraire le message « exception » s'affiche. On affichera aussi la nature de l'exception dans la fenêtre logcat.

→ Faire vérifier le résultat par le professeur.

3 Un paragraphe de cours : les intentions (« intents »)

Une des problématiques de la programmation ANDROID est **l'échange de données** entre applications, applications et services, applications et matériel, application et système ANDROID.

Le problème se pose, par exemple, pour récupérer les données issues d'un capteur (gyroscope), de la batterie, d'une communication BLUETOOTH.

Le mécanisme mis en œuvre est celui des « intents » (intentions en français)

3.1 Qu'est-ce qu'un « intent » ?

Un « intent » est un objet (faisant donc partie d'une classe) contenant les champs montrés figure 4. Ces champs sont **des chaînes de caractères**. N'oublions pas que ces objets permettent **l'échange de données**.

Les champs principaux sont l'action et les données. Le champ « Action » désigne ce que l'on veut que le destinataire des données fasse. Les actions sont classées en « catégories ».

Les données sont accessibles via des EXTRAS. On utilise le terme **BUNDLE** pour désigner l'ensemble des EXTRAS disponibles pour un intent.



3.2 Les filtres d' « intent »

Les intentions auxquelles l'activité peut répondre se déclarent dans le fichier manifest.xml de l'application entre les bornes `<intent-filter>`.

Par exemple, pour une application qui se lance par l'IHM principale d'Android, on ajoute (Figure 2) :

```
<activity android:name=".MainActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Figure 2 - filtres d'intent pour l'activité principale

Un autre exemple (Figure 3) : Ci-dessous, l'action EDIT, en association avec le « data mimetype » « contact », permettrait ici à notre application de modifier les contacts du téléphone, à condition que l'utilisateur en donne l'autorisation.

```
<intent-filter android:label="Edit Contact">
    <action android:name="android.intent.action.EDIT" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:mimeType="vnd.android.cursor.item/contact" android:host="com.android.contacts" />
</intent-filter>
```

Figure 3 - filtre d'intent – ACTION EDIT

De la même manière, l'action VIEW permettrait à l'application de lire les contacts, à condition que l'utilisateur de la machine donne son autorisation (Figure 4)

```
<intent-filter android:label="View Contact">
    <action android:name="android.intent.action.VIEW" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:mimeType="vnd.android.cursor.item/contact" android:host="com.android.contacts" />
</intent-filter>
```

Figure 4 - filtre d'intent - action VIEW

Un filtre d'intent peut aussi être déclaré de façon dynamique dans le code JAVA de l'application. Nous verrons cela dans le prochain atelier.

3.3 Méthodes disponibles pour manipuler les « intent »

ANDROID STUDIO offre de nombreux constructeurs et méthodes pour déclarer et manipuler les intentions.

Tout est spécifié ici : <https://developer.android.com/reference/android/content/Intent.html> Mais ce n'est pas très simple.



4 Mise en œuvre des intentions : supervision des caractéristiques de la batterie de la machine ANDROID

Dans ce TP, nous allons « capturer » les valeurs retournées par l' « intent » [Intent.ACTION_BATTERY_CHANGED](#) qui concernent l'état de la batterie de la machine utilisée.

Les données concernées sont la tension de la batterie, la valeur maximale de celle-ci, le pourcentage de la charge de la batterie, sa température. On peut savoir aussi si la charge se fait via un câble USB, via le réseau AC, si la batterie est en charge, si la charge est complète...

Chaque fois que l'une de ces données change, un « intent » est « envoyé » par le système ANDROID. La programmation consiste à détecter cet intent et à lire les données associées.

Il est aussi possible de provoquer « à la main » l'émission d'un « intent » ce que nous ferons en appuyant sur un bouton.

La classe « BatteryManager » permet d'accéder aux données retournées par l'intent [Intent.ACTION_BATTERY_CHANGED](#)

La documentation est ici :

<https://developer.android.com/reference/android/os/BatteryManager>

- Créer le projet TP5_Batterie dans le dossier TP5_Batterie du dossier de travail.
- Placer sur l'IU 2 boutons avec le texte respectivement « GO » et « STOP ». Les placer symétriquement en haut de l'écran.
- Placer un TextView en bas et au milieu de l'écran, avec pour texte initial « Affichage arrêté »
- Placer un autre TextView au milieu de l'écran, sans texte initial, et de type « nombre ».
- Déclarer les objets associés aux 4 « view » dans la classe « MainActivity » (voir ci-dessous)

```
public class MainActivity extends AppCompatActivity {  
  
    Button boutonOK; // associé au bouton « Lance Affichage »  
    Button boutonStop; // associé au bouton « Stoppe Affichage »  
    TextView niveau; // au milieu de l'écran, affichage tension de la batterie  
    TextView resultat; // en bas de l'écran, état des boutons
```

- Dans la méthode « onCreate », inhiber l'accès au bouton stop dès le début du code (le bouton ne sera plus « clickable, méthode « setEnabled ») et associer les « ID » à chaque objet.
- Compiler et charger l'application pour vérifier que tout est en place, en particulier que le bouton « Stoppe Affichage » n'est pas clickable.
- Préparer les 2 « listeners » permettant de coder ce que doit faire l'application en cas d'appui sur l'un des deux boutons. On écrira les 2 méthodes, sans code entre les deux accolades.

Aide :

```
boutonOK.setOnClickListener etc...  
boutonStop.setOnClickListener etc...
```

- Dans le listener du bouton « boutonOK », ajouter le code permettant d'afficher dans le TextView « resultat » le message « Affichage OK », d'inhiber le bouton « boutonOK » et d'activer le bouton « BoutonStop ».



→ Dans le listener du bouton « boutonStop », ajouter le code permettant d'afficher dans le TextView « resultat » le message « Affichage arrêté », d'inhiber le bouton « boutonStop » et d'activer le bouton « BoutonOK ».

→ Compiler et charger l'application pour vérifier que cela fonctionne.

Nous allons maintenant récupérer les messages (ou plutôt les intents) envoyés par le système d'exploitation ANDROID et qui concernent des mesures sur la batterie. Pour cela, nous devons définir une classe qui hérite de la classe `BroadcastReceiver`. Cette classe n'est pas spécifique à la batterie mais utilisée quelque soit la source de l'intent.

Les objets de cette classe sont ainsi conçus pour recevoir des intents et appliquer des comportements spécifiques au code. La classe ne possède qu'une seule méthode `onReceive()` que notre classe doit implémenter.

→ Insérer le code ci-dessous avant la méthode `OnCreate` (Cette nouvelle classe fait partie de `MainActivity`) :

```
public class recepateurEtatBatterie extends BroadcastReceiver {  
    @Override  
    public void onReceive(Context context, Intent intent) {  
        Log.i("Batterie", "Intent reçu") ;  
    }  
}
```

→ Déclarer ensuite un objet de la classe `recepateurEtatBatterie` dans le code de l'activité principale (avant les listeners) :

```
recepateurEtatBatterie monRecepteur = new recepateurEtatBatterie();
```

Il nous faut maintenant ajouter le filtre d'intentions correspondant aux propriétés de la batterie, de sorte que notre `BroadcastReceiver` soit exécuté automatiquement par le système ANDROID.

D'après le paragraphe 6, il nous faut ajouter dans le manifeste de l'application les lignes ci-dessous (après `</activity>`)

```
<receiver android:name=".MainActivity$recepateurEtatBatterie">  
    <intent-filter>  
        <action android:name="android.intent.action.BATTERY_CHANGED"/>  
    </intent-filter>  
</receiver>
```

De plus, ici, on va ajouter le déclenchement de la production de « l'intent » pour pouvoir lire les données lors de l'appui sur le bouton « GO ». Il faut donc implanter la réception de l'intent de façon dynamique.

→ On écrit ainsi

```
registerReceiver (monRecepteur, new IntentFilter(Intent.ACTION_BATTERY_CHANGED) ;  
ceci dans le listener du bouton « boutonOK ».
```

→ Et pour que ce soit plus amusant, nous allons désactiver le filtre dans le listener du bouton « boutonStop » :

```
unregisterReceiver (monRecepteur) ;
```

→ Compiler et charger l'application pour vérifier que cela fonctionne. A quel moment « l'intent est-il reçu ? » (voir la fenêtre Log).

Il n'y a plus maintenant qu'à lire les « extras » de l'intent pour récupérer les informations de la batterie :



→ Déclarer les données ci-dessous dans la classe « `recepteurEtatBatterie` » :

```
int valeurCharge;  
int valeurMax;  
int pctCharge;  
double temperature;  
int tension;
```


→ et ajouter le code ci-dessous dans la méthode « `onReceive` » :

```
valeurCharge = intent.getIntExtra(BatteryManager.EXTRA_LEVEL, -1);  
valeurMax= intent.getIntExtra(BatteryManager.EXTRA_SCALE,-1);  
pctCharge = (valeurCharge * 100)/valeurMax;  
tension = intent.getIntExtra(BatteryManager.EXTRA_VOLTAGE, -1);  
temperature = intent.getIntExtra(BatteryManager.EXTRA_TEMPERATURE, 0)/10.0;  
// Affichage  
niveau.setText("Tension batterie = "+ Integer.toString(tension) + "mV");
```

→ Compiler et charger l'application pour vérifier que cela fonctionne. La tension de la batterie est donnée en mV, la température en dixièmes de degrés.

→ Tester le programme avec une cible réelle. Observer la fenêtre du Logcat, vérifier que « l'intent » est envoyé sans avoir appuyé sur le bouton « GO » quand la tension de la batterie change. Débrancher le câble et vérifier que si le bouton « Stop » est activé, il n'y a pas de rafraichissement de l'affichage. Expliquer pourquoi.

5 Atelier

Afficher dans l'IU un tableau de bord de l'état de la batterie, avec la tension en Volt, la température en degrés, le pourcentage de charge sous forme d'une barre de progression. Indiquer aussi si la batterie est en charge ou non. Si elle est en charge, l'icône  doit s'afficher. Elle est disponible dans les ressources.

La documentation du widget « barre de progression » est disponible là :

<https://developer.android.com/reference/android/widget/ProgressBar.html>

Aide :

- La barre progresse de 0 à 100.
- On déclare un objet de type `ProgressBar` et on récupère son ID.
- On peut alors utiliser les méthodes
 - o `void incrementProgressBy (int diff)` qui augmente (nombre positif) ou diminue (nombre négatif) la progression de la barre
 - o `void setProgress (int progress)` qui donne à la progression de la barre la valeur « progress ». `int getProgress()` retourne la position actuelle de la barre

Trouver également un moyen, via les intents, de récupérer l'état de branchement du téléphone / de la tablette. La documentation de la classe `BatteryManager` peut aider.



6 Annexes

Annexe1 : liste des opérations qui génèrent une exception arithmétique (Figure 7)

ArithmeticException is the base class for the following exceptions:

- **DivideByZeroException**, which is thrown in integer division when the divisor is 0. For example, attempting to divide 10 by 0 throws a **DivideByZeroException** exception.
- **NotFiniteNumberException**, which is thrown when an operation is performed on or returns **Double.NaN**, **Double.NegativeInfinity**, **Double.PositiveInfinity**, **Single.NaN**, **Single.NegativeInfinity**, **Single.PositiveInfinity**, and the programming language used does not support those values.
- **OverflowException**, which is thrown when the result of an operation is outside the bounds of the target data type. That is, it is less than a number's **MinValue** property or greater than its **MaxValue** property. For example, attempting to assign $200 + 200$ to a **Byte** value throws an **OverflowException** exception, since 400 greater than 256, the upper bound of the **Byte** data type.

Figure 5 - Exceptions arithmétiques

Annexe2 : rappel de quelques concepts et mots clés du langage JAVA et pas seulement

1- Le mot clé « super » :

Ce mot est utilisé dans la définition d'une classe fille lorsque l'on veut que le constructeur de la classe mère soit exécuté. Ceci est souvent l'occasion de passer un paramètre à ce constructeur.

2- Le mot clé « @override »

Ce mot est utilisé pour définir une méthode qui est héritée de la classe parente. Si la méthode n'existe pas dans la classe parente, le compilateur génèrera une erreur. Ce mot clé est donc un « pense-bête » pour vérifier que la méthode parente existe bien. Il n'est donc pas obligatoire.

3- La classe « Integer » :

Classe permettant de définir un nombre de type int. Plusieurs méthodes permettent des conversions, par exemple en chaîne de caractères.

4- La classe « Double » :

Classe permettant de définir un nombre décimal. Plusieurs méthodes permettent des conversions, par exemple en chaîne de caractères.

5- Le mot clé « URI » (Uniform Resource Identifier):

Chaîne de caractères identifiant une ressource sur un réseau (par exemple une ressource Web) et dont la syntaxe respecte une norme d'Internet mise en place pour le World Wide Web, par exemple, www.iut-cachan.fr est un URI (sous famille URL). Un URI doit permettre d'identifier une ressource de manière permanente, même si la ressource est déplacée ou supprimée.