Travaux dirigés

TD 0 : Révisions

1. Créez un programme java prenant N paramètres et affichant sur la sortie standard une concaténation de tous ses arguments. Exemple :
   `java monprogramme Ceci va etre affiche sur la sortie standard`
   Affichera :
   `Ceci va etre affiche sur la sortie standard`
2. Modifiez ce programme pour afficher les arguments dans le sens inverse de l'entrée, et séparés par des tirets. Exemple de sortie avec la même entrée :
   `standard-sortie-la-sur-affiche-etre-va-Ceci`
3. Créez un programme dont la classe principale contiendra une méthode statique « `factorielle` » qui prendra en **paramètre** un nombre et **retournera** la factorielle de ce nombre. Celui-ci pourra ensuite être **affiché** dans la fonction `main`.

TD 1 : Questionnaire

- Dans un nouveau projet nous allons créer les fonctions nécessaires à la création d'un questionnaire.
- Commencez par créer une **classe abstraite** `Question` ayant les prototypes de méthodes suivantes :
  - `Question(String text) ; // Constructeur, prenant en paramètre la question à poser`
  - `void prompt() ; // Méthode abstraite. Permet de demander la réponse à l'utilisateur (en affichant un prompt, comme "> " par exemple).`
  - `void print() ; // Méthode abstraite. Permet d'afficher la question.`
  - `String getResponse() ; // Méthode abstraite. Permet de retourner la réponse fournie par l'utilisateur.`
  - `boolean isOk() ; // Méthode abstraite. Permet de savoir si la réponse actuelle est valide ou non.`

- Vous créerez ensuite deux autres **classes** `QuestionText` et `QuestionQCM` implémentant les méthodes abstraites de la classe `Question`.
    - Pour la classe QuestionQCM, il faudra avoir une méthode supplémentaire `setChoices(String[] choices, int goodResponse) ;` permettant de définir les réponses possibles à la question, ainsi que l'indice de la bonne réponse. (Notez que vous pouvez aussi ajouter ces paramètres au constructeur, afin de ne pas avoir à appeler cette méthode).
    - Afin de convertir la chaîne lue dans `QuestionQCM` en entier, vous pourrez utiliser la fonction `Integer.parseInt() ;` qui prend un `String` en paramètre, et retourne un `int`.
- Enfin, vous créerez un type générique `Questionnaire` :
    - La forme de cette classe sera :
      `public class Questionnaire<Q extends Question> { /* … */ }`
      Ceci va permettre de s'assurer que le type `Q` fourni implémente la classe abstraite `Question` et va ainsi nous permettre d'utiliser les méthodes abstraite de celle-ci.
    - Elle devra avoir les méthodes suivantes :
    - `void addQuestion(Q q) ; // Permet d'ajouter une question dans le questionnaire.`
    - `void run() ; // Permet de lancer le questionnaire, poser les questions à l'utilisateur, et attendre les réponses.`
    - `int mark() ; // Permet de retourner la note actuelle.`
    - `int getQuestionsCount() ; // Permet de retourner le nombre de questions enregistrées.`
- Testez votre code en :
    - Ayant plusieurs types de question dans votre questionnaire.
    - Ayant qu'un seul type de question dans votre questionnaire.
    - N'ayant aucune question enregistrée.
- Citez les avantages et inconvénients d'utiliser une **classe absitraite** pour `Question` au lieu d'une **interface**.

Voilà quelques informations supplémentaires pour vous aider dans la progression de ce TD :

- Pour écrire du texte dans un terminal : `System.out.println();` ou

```
System.out.print();
```

- Pour lire du texte d'un terminal :

```
import java.util.Scanner;
/* ... */
Scanner scanner = new Scanner(System.in);
String answer = scanner.nextLine();
```

- Pour stocker les éléments dans votre classe `Questionnaire`, vous pourrez utiliser la classe `Vector`, qui s'utilise ainsi (Sa documentation complète est disponible ici : https://docs.oracle.com/javase/7/docs/api/java/util/Vector.html) :

```
import java.util.Vector;
/* … */
Vector<MonType> v = new Vector<MonType>();
v.add(uneInstanceDeMonType); // Ajoute un élément à la fin
v.size(); // Retourne le nombre d'éléments en mémoire
v.elementAt(i); // Retourne l'élément à l'indice i
```

TD 2 : UI Android

Un compteur

- En utilisant Android Studio, créez une application avec une activité vide par défaut (*Empty Activity*).
- Dans le layout principal, ajoutez un bouton et une zone de texte.
- Dans l'activité utilisant ce layout, ajoutez trois attributs :

```
private Button button;
private TextView text;
private int counter;
```

- Dans la méthode **onCreate**, récupérez les éléments button et text via les méthodes **findViewById** de votre activité (Vous devrez caster le résultat de cette méthode!). Pour récupérer votre bouton, vous appellerez donc **findViewById(R.id.mon_bouton)**.
- Créez une méthode **click()** permettant d'incrémenter counter et d'afficher sa valeur dans la zone de texte de l'activité, via la méthode **setText**.
- Capturez l'évènement **onClick** via la méthode **setOnClickListener** du bouton en utilisant une

classe anonyme, et depuis cette classe vous venez de créer, appelez la méthode **click** de votre activité.

Un menu

- À votre application, ajoutez un fichier ressource, du type menu. Mettez-y le contenu suivant :

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:title="Change direction" android:id="@+id/change_direction
" />
    <item android:title="Reset" android:id="@+id/reset" />
</menu>
```

- Dans le fichier de votre activité, ajoutez la ligne `import android.view.Menu;`
- Intégrez ce menu à votre application en surchargeant la méthode `public boolean onCreateOptionsMenu(Menu menu)` de votre activité.
- Afin de capturer l'événement de sélection, surchargez aussi la méthode `public boolean onOptionsItemSelected(MenuItem item)` de votre activité.
  - Le menu **reset** réinitialisera la valeur du compteur à zéro.
  - Le menu **change_direction** changera le sens de votre compteur (croissant / décroissant).
- Pour ce deuxième menu, vous adapterez bien évidemment votre classe.
- Testez votre code.

Animation de texte

- Fermez le projet actuel, et choisissez *Import an Android code sample*.
  - Choisissez le projet **TextSwitcher**
- Testez cette application et analysez comment fonctionnent les animations (Le code principal se trouve dans le fichier **MainActivity**). Vous pourrez par exemple :
  - Changer les paramètres transmis à `setGravity`.
  - Expérimenter différents types d'animations, comme `android.R.anim.slide_in_left` ou `android.R.anim.slide_out_right`.
- **Note :** Vous disposez d'autres classes que **TextSwitcher**, telles que **ViewSwitcher** ou **ImageSwitcher**, qui permettent d'animer simplement des images ou des vues.

Géolocalisation

- En vous servant de la documentation sur http://developer.android.com/guide/topics/location/strategies.html, utilisez l'API Android pour obtenir la géolocalisation du téléphone et afficher la latitude et longitude à l'écran (dans un ou plusieurs `TextView`), ainsi que dans les logs (via l'objet **Log**, dont l'utilisation est détaillée dans les annexes du cours).

TD 3 : APIs Android

Un client HTTP

- Dans une nouvelle application, créez une interface ayant trois éléments :
  1. Un champ de formulaire pour du texte (`EditText`).
  2. Un bouton (`Button`).
  3. Un affichage de texte (`TextView`).
- Un clic sur le bouton (2) va récupérer l'URL inscrite dans le champ (1), récupérer le contenu de l'URL spécifiée en HTTP, et l'afficher dans la zone de texte (3).
- Vous utiliserez la classe AsyncTask pour gérer le téléchargement en arrière plan.
- Vous pourrez utiliser les classes `BufferedReader` et `StringBuilder` pour faciliter la lecture du flux de données.
- En guise d'amélioration, vous pourrez aussi ajouter une barre de progression affichant l'avancement du téléchargement.

Un capteur d'accélération

- Dans une nouvelle application, vous créerez un capteur d'accélération en utilisant l'accéléromètre du téléphone. Pour cela, vous devrez utiliser la classe `SensorManager`, celle-ci permettant de récupérer une instance de `Sensor` pour le capteur qui vous intéresse.
- Dans la méthode `onSensorChanged`, les trois premiers éléments de l'attribut value de l'événement fourni en paramètre sont l'accélération sur l'axe `X`, `Y` et `Z`.
- Avec ces valeurs, affichez l'accélération globale du téléphone, via la formule bien connue de Pythagore appliquée en 3 dimensions :
  $Acc = \sqrt{(X^2 + Y^2 + Z^2)}$
- Faites en sorte d'afficher cette valeur une fois par seconde seulement (Le timestamp actuel sur le téléphone est fourni via la méthode `System.currentTimeMillis()`.
- En plus de cette valeur, vous afficherez aussi la valeur minimale et maximale obtenue jusqu'à maintenant.

Une minuterie

- Dans une nouvelle application, vous allez créer une minuterie.
- Celle-ci possédera les caractéristiques suivantes :
  - Le timer sera par défaut de 60 secondes et ne sera pas configurable initialement.
  - L'application affichera le timer en secondes, avec en dessous deux boutons, permettant de

démarrer, mettre en pause et arrêter le timer.
    - L'application n'utilisera pas de services dans un premier temps, ce qui signifie que si l'application est mise en pause ou fermée, le timer sera coupé.
    - Vous animerez l'interface à chaque changement de secondes, comme vu au TD précédent.
- Vous utiliserez la classe **java.util.Timer** pour gérer l'intervalle entre les secondes.
    - Vous préciserez au constructeur de cette classe d'être exécuté dans un autre thread (démon).
    - Étant dans un autre thread, vous ne pourrez pas mettre à jour l'UI directement, il vous faudra utiliser la classe **android.os.Handler** pour lui transmettre des éléments, de cette façon :

```
Handler h=new Handler();
h.post(new Runnable() {
    public void run() {
        /* Votre code à lancer dans l'UI */
    }
});
```

    - L'instance **Handler** devra être créée depuis le thread de l'UI !
- En terme d'alerte, vous utiliserez le package **android.os.Vibrator**.
    - Pour pouvoir l'utiliser, vous devrez inclure la permission : `<uses-permission android:name="android.permission.VIBRATE"/>` à votre fichier Manifest