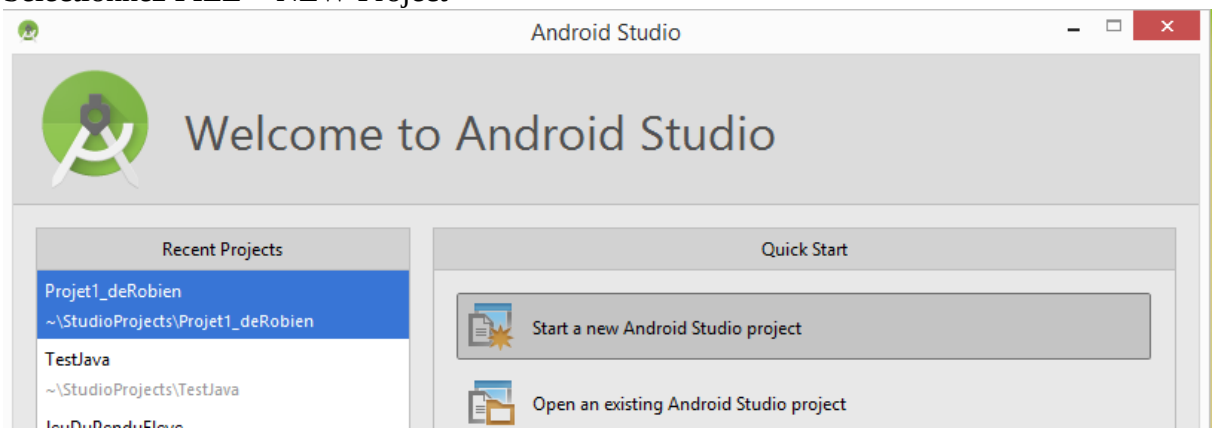
TP01 : INITIATION ANDROID STUDIO

Objectif du TP :

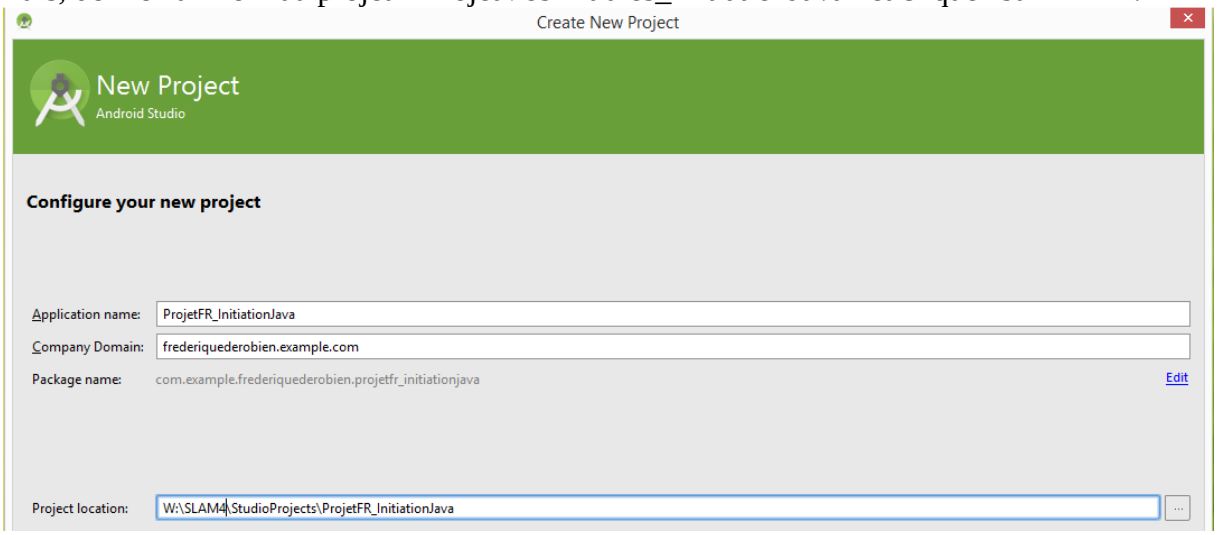
- Savoir créer un projet et prendre en main l'IDE
- Créer une classe JAVA
- Tester une classe JAVA avec le Framework Junit.

CREATION D'UN NOUVEAU PROJET

Sélectionnez FILE > NEW Project



Puis, donnez un nom au projet « ProjetVosInitiales_InitiationJava» et cliquez sur NEXT.



Vérifier bien l'endroit de sauvegarde de votre projet : **Project Location** : vous devez indiquer l'emplacement sur votre disque DISTANT ou LOCAL où vont se trouver toutes les sources du projet.

ⓘ un package correspond à un dossier dans votre projet.

Cliquez sur NEXT… et Choisir l'API minimum pour le SDK : Ici choisir ANDROID SDK version 4.0 (IceCreamSandwich) : environ 90% de propriétaire de téléphone Android

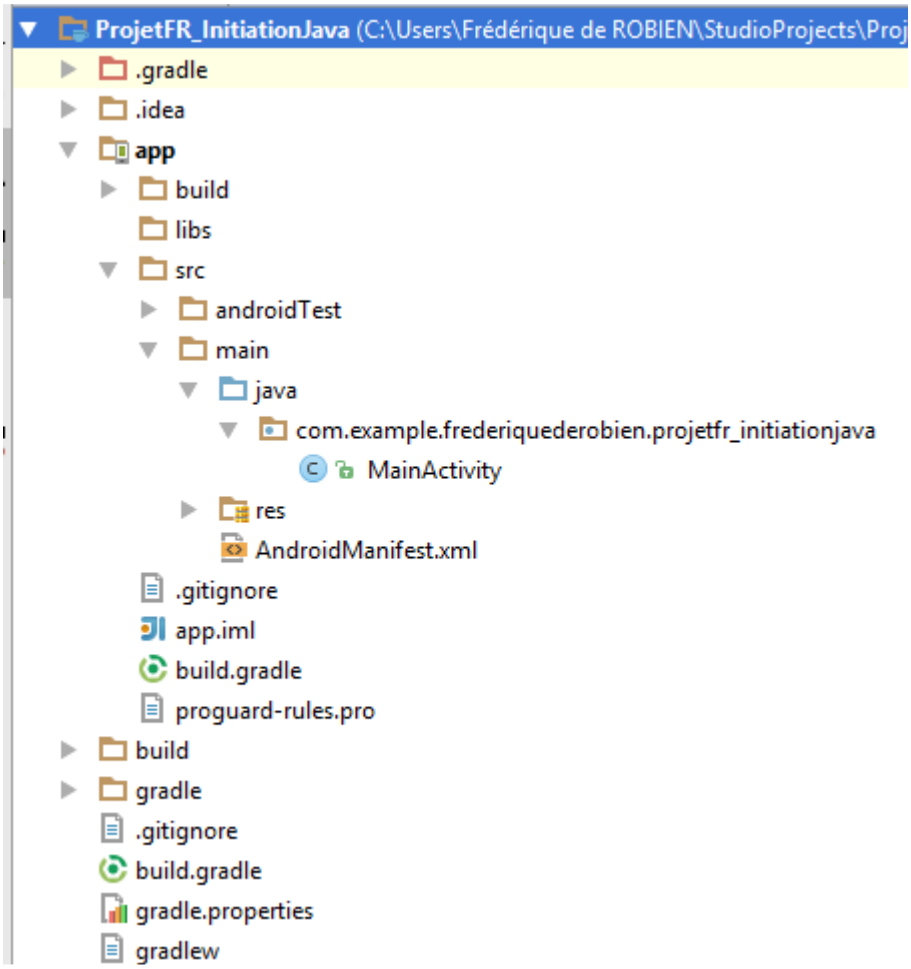
TRAVAIL à faire :

- Chercher la définition d'une API

- Chercher la définition du SDK

Choisir la manière dont l'utilisateur pourra visualiser les informations (VUE nommée ACTIVITY) : ici BlankActivity
Puis NEXT et laisser le nom de votre première activity (MainActivity)
Puis FINISH.

OBSERVATION DE L'ARBORESCENCE CREEE.



Vous constatez que la classe« **MainActivity** » a été créée dans le dossier **Main**.
C'est dans ce dossier que nous allons créer toutes les classes « métier » de notre projet.

TRAVAIL à faire :

- Qu'est qu'une classe métier ?

LE CODE – EDITION DES FICHIERS SOURCE

1) Règle syntaxique en JAVA

Le code est normalisé par des règles syntaxiques strictes.

- Le nom de la **classe** ont des **majuscules en première lettre**.
  Ex : `Voiture()`
- Le nom des variables n'ont jamais de majuscules en première lettre, ni d'espaces. Ex : `public String marqueVoiture ;`
- Création d'un get et d'un set pour chaque variable (phénomène d'encapsulation). Les noms de méthodes n'ont jamais de majuscules en première lettre, ni d'espaces. Pour des raisons de lisibilité, on met des majuscules dans le nom lui-même. Ex :`getMarqueVoiture() ;`
- Pas d'accent en java (pas conseillé).
- Les constantes sont toujours en majuscules
  Ex `Public final static int CONST = 1;`

💣 Java ne reconnaît pas les objets s'ils ne sont pas écrits exactement de la même manière. Cela est souvent source d'erreur. Alors, Soyez vigilant et rigoureux.

2) Visualisation de la structure d'une classe

Une classe doit toujours comporter des items OBLIGATOIRES comme

- **Package** : dossier où la classe se trouve physiquement
- **Import** : lien vers des dossiers des classes qui vont servir à la classe actuelle (ces classes ne sont pas dans le même package donc on a besoin de savoir où elles se trouvent !)
- **Public class N**omClasse : début de la définition de la classe nommée **N**omClasse

3) Etre efficace dans l'outil Android Studio…

a) *Afficher les numéros de ligne pour le débuggage*

Vous pouvez afficher les numéros de ligne dans l'Editeur de Source en faisant clic-droit sur la marge de gauche et sélectionnez Show Line Numbers.

b) *Utiliser la complétion de code*

La complétion de code est le fait que l'IDE propose les différentes possibilités dans les propriétés ou méthodes dont vous avez accès. Souvent après un point, la boîte de complétion s'ouvre automatiquement. Si la boîte de complétion de code ne s'ouvre pas, pressez **Ctrl-Espace**, pour l'ouvrir manuellement.

c) *reformater le **code (menu CODE)***

Reformater le code :

Souvent votre code est mal indenté, il faut le reformater.
Pressez Ctrl+Alt+L pour reformater l'ensemble du fichier.
Si des lignes sont sélectionnées, seules ces lignes sont reformatées.

d) Commenter le code (menu CODE) et utiliser le formalisme de la JAVADOC

Commentaire :

```
//commentaire
/*commen
      taire*/
```

Formalisme des commentaires pour générer de la **documentation automatiquement** garce à l'outil JAVADOC

```
/**
 * methode permettant de ….
 *
* @param param1 de type String
*                represente le nom de la personne ....
* @throws FormatException lance une erreur si les conversions entre type
sont invalides.
 * @author Frederique de Robien
 * @version : 1.0
 */
public void methode(String param1) throws FormatException {
```

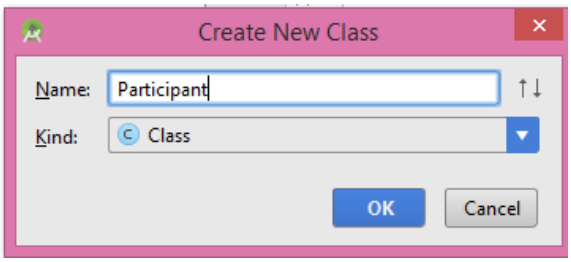
1) Créer une nouvelle classe

a) Nouvelle classe JAVA

Se positionner dans l'arborescence à l'endroit où l'on veut créer cette classe (ici même dossier (package) que la classe MainActivity.java

Clic droit sur le package, puis NEW
et JAVA CLASS
et donner le nom de cette classe (ici Participant)
Et OK

Vérifier le package de cette classe !



COMPLETER LA CLASSE PARTICIPANT

1) Les variables d'instance et le constructeur

Créer les **4 variables d'instance** pour cette classe à savoir :

- Le nom du participant
- Son prénom
- Sa date de naissance
- Et le numéro du département (ex : 49) /*utile pour faire des statistiques*/

Bien choisir le type adéquat pour chaque variable (type primitif ou type issu d'une classe) !

TRAVAIL à faire :

- Rechercher les différents types primitifs que l'on peut utiliser en JAVA

- Quels sont les autres types issus d'une classe : donner des exemples.

Créer les **2 constructeurs** (celui par défaut et celui paramétré)
Puis Reformater votre code : Ctrl+Alt+L

2) Les accesseurs SET et GET

Créer tous les accesseurs GET afin de pouvoir récupérer **toutes** les données encapsulées (principe de l'encapsulation avec private)

Créer 1 seul accesseur SET pour pouvoir modifier le département (s'il déménage)

3) La méthode toString() avec utilisation de la classe StringBuilder

Nous allons créer la méthode toString() afin qu'elle nous renvoie une chaîne de caractère.
Cette méthode a pour particularité d'exister dans toutes les classes, nous allons donc la redéfinir (Override)
➔ principe de la redéfinition !

TRAVAIL à faire :

- Rechercher quand est-ce que l'on applique la **redéfinition** (@Override)en POO ?

Pour cela, nous allons utiliser la classe StringBuilder qui nous permet de créer une chaine au fur et à mesure (principe de la concaténation).

Compréhension et Utilisation du StringBuilder...

```
@Override
public String toString(){

    StringBuilder sb = new StringBuilder("Chaîne à stocker");
    sb.append("chaîne à concatener à la précédente");
    sb.append(nom).append("\n ");/* \n signifie retour ligne*/
    sb.append("autre chaîne") ;

    return sb.toString(); /*retour de la chaîne

}
```

Compléter la méthode toString() qui permet de retourner une chaine composé de tous les éléments d'un participant.

Nous venons de créer une classe, comment tester puisque nous n'avons pas encore d'interface graphique (VUE) pour visualiser les informations ?

TESTER LA CLASSE PARTICIPANT

1) Le Framework Junit

En Java, il existe un Framework complètement dédié à la création des tests. Il s'appelle Junit pour Java Unit (création de **tests Unitaires en JAVA**)

Le **développeur responsable** de la classe Participant doit écrire des **tests unitaires** des différentes méthodes qu'il a codées. Ces tests ont pour but de vérifier que les méthodes de la classe se comportent comme prévu.

Pour cela le développeur utilise une classe dédiée aux tests, la classe ***Assert***.

Un test unitaire est considéré comme conforme si durant son exécution aucune assertion (affirmation) n'est violée. L'***annexe*** décrit plus précisément l'utilisation de la classe Assert.

Les cas de tests utilisent des affirmations (assertion en anglais) sous la forme de méthodes nommées assertXXX() proposées par le framework. Il existe de nombreuses méthodes de ce type qui sont héritées de la classe junit.framework.Assert :

Méthode	Rôle
assertEquals()	**Vérifier l'égalité de deux valeurs de type primitif ou objet (en utilisant la méthode equals()). Il existe de nombreuses surcharges de cette méthode pour chaque type primitif, pour un objet de type Object et pour un objet de type String**
assertFalse()	Vérifier que la valeur fournie en paramètre est fausse
assertNull()	**Vérifier que l'objet fourni en paramètre soit null**
assertNotNull()	Vérifier que l'objet fourni en paramètre ne soit pas null
assertSame()	Vérifier que les deux objets fournis en paramètre font référence à la même entité Exemples identiques : assertSame("Les deux objets sont identiques", obj1, obj2); assertTrue("Les deux objets sont identiques ", obj1 == obj2);
assertNotSame()	Vérifier que les deux objets fournis en paramètre ne font pas référence à la même entité
assertTrue()	**Vérifier que la valeur fournie en paramètre est vraie**

2) Principe des TEST avec JUnit

Pour tester que la classe participant est correcte, nous allons créer une nouvelle classe de TEST héritant de la classe junit.framework.TestCase

Le nom de cette classe est le nom de la classe à tester suivi par "Test"

```
01.
02. import junit.framework.TestCase;

03.public class MaClasseTest extends TestCase{
04.
05.public MaClasseTest(String testMethodName) {
06.   super(testMethodName);
07.}
08.
```

```
09.public void testMethode() throws Exception {
10.         fail("Cas de test a ecrire");
11.}
12.}
```

Dans cette classe, il faut écrire une méthode dont le nom commence par "test" en minuscule suivi du nom du cas de test (généralement le nom de la méthode à tester). Chacune de ces méthodes doit avoir les caractéristiques suivantes :

- elle doit être déclarée public
- elle ne doit renvoyer aucune valeur (void)
- elle ne doit pas posséder de paramètres.

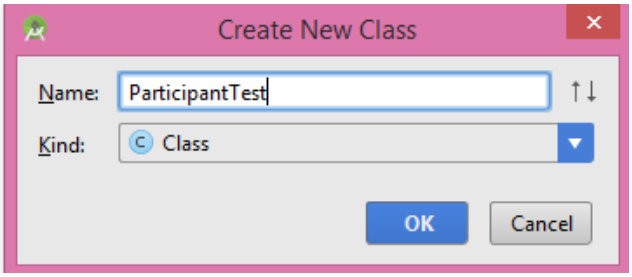
💣Par introspection, JUnit va automatiquement rechercher toutes les méthodes qui respectent cette convention. Le respect de ces règles est donc important pour une bonne exécution des tests par JUnit.

3) Création de notre premier TEST JUnit pour la Classe Participant

Se positionner dans le dossier app/src/androidTest/java/votrePackage.

Faire Clic droit, puis new JavaClass.

Cette classe hérite de TestCase (donc rajouter EXTENDS TestCase)



Mettre le import afin d'importer le framework Junit TestCase.

Créer le constructeur de cette classe de Test

```
public ParticipantTest(String testMethodName) {
    super(testMethodName);
}
```

Créer votre méthode qui doit toujours commencer par **test** en minuscules. Cette méthode lance une Exception lorsque votre test n'est pas concluant !

```
public void testGetNom() throws Exception {

        /*instancier la classe participant en utilisant le constructeur paraméte*/

//création d'un objet sdf qui est du format dd/MM/yyyy pour pouvoir entrer des dates dans ce format
SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy ");
Participant p = new Participant("de Robien", "Frédérique",
sdf.parse("01/01/2001"),(byte)49) ;
//on force que l'entier 49 soit transformer en byte (sinon il est trop grand): Principe du CAST

        /*appel de la méthode getNom() puis récupération du nom retourné*/
String n = p.getNom();
```

```
        /*appel de la méthode assert pour vérifier si cela est correct*/

assertEquals("Mauvais Nom de Participant",n,"de Robien");
}
```

Bonnes pratiques pour les tests qui doivent être indépendants !

Chacune de ces méthodes contient généralement des traitements en trois étapes :

- **Instanciation** des objets requis
- Invocation des traitements sur les objets : **appel de la méthode à tester**
- Vérification des résultats des traitements **avec la méthode assert**

4) Lancement du TEST !

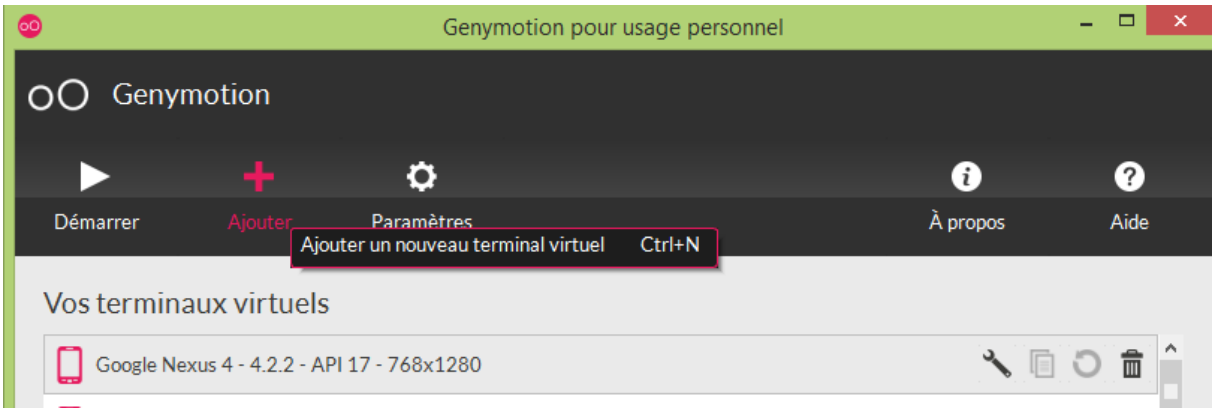
Pour lancer votre test, clic droit puis **RUN ParticipantTest** (Ctrl+Maj+F10)

- ⇨ Il va tester toutes les méthodes présentes dans cette classe.
- ⇨ Il va envoyer une ERREUR si votre test n'est pas correct

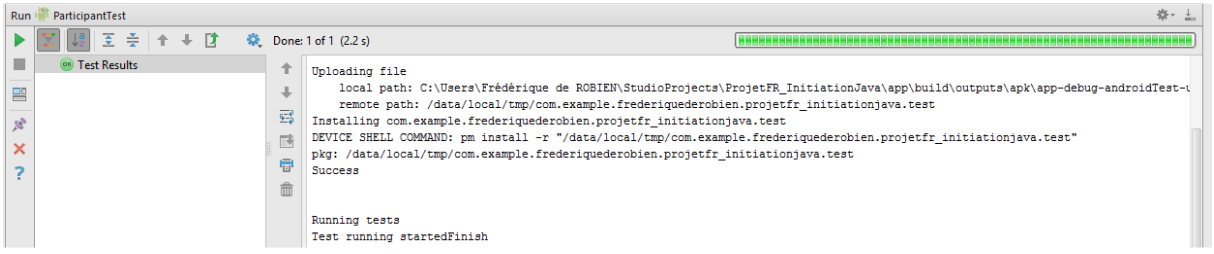
Il vous demande un Android DEVICE (c'est à dire un support Androïd pour lancer l'interface graphique du projet)

a) Utilisation d'un simulateur de machine Android (GennyMotion)

Lancer GennyMotion et configurer une machine Android



Vous obtenez normalement dans la Console RUN en bas de votre écran :



BRAVO, votre test a bien fonctionné !

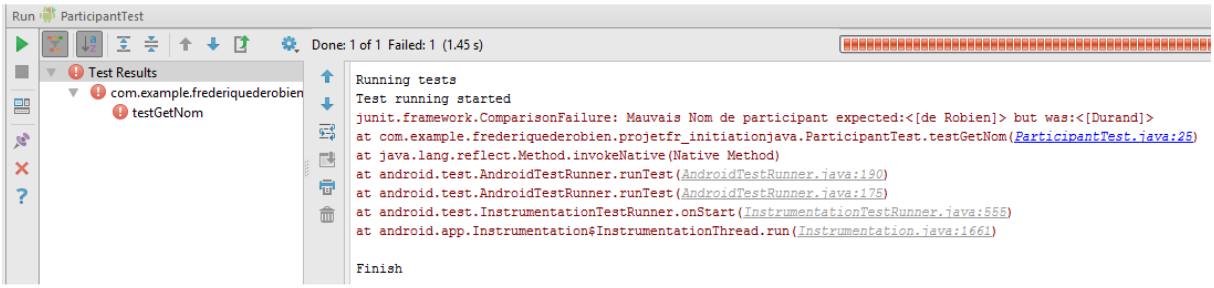
Modifier le ASSERT de la façon suivante :

```
assertEquals("Mauvais Nom de Participant",n,"Dupont");
```

et lancer votre test !

C'est NORMAL, il vous « lance » l'erreur par l'intermédiaire de l'exception (throws Exception !

Vous obtenez normalement dans la Console RUN en bas de votre écran :



Le message indique qu'il attend « de Robien » mais il ne trouve que « Durand » donc il envoie le message d'erreur !

5) Amélioration du test

TEST 1 :

```
Participant p = null;
p= new Participant("de Robien", "Frederique",
sdf.parse("01/01/2001"),(byte)49) ;
assertNotNull("objet non instancié",p);
```

TEST 2 :

```
Participant p = null;
assertNotNull("objet non instancié",p);
p= new Participant("de Robien", "Frederique",
sdf.parse("01/01/2001"),(byte)49) ;
```

Dans quel cas, il y a une erreur ????? Pourquoi ?

Faire une nouvelle méthode de test dans la classe ParticipantTest afin de vérifier la méthode toString(). (Cf aide pour la gestion des dates)

A VOUS de JOUER !

AIDE :

LA GESTION DES DATES ....

La prise en charge des dates est particulière en JAVA. On veut changer la date de Naissance et la Date du cours du format String (chaîne) au format date (plus approprié)

private Date dateNais ; //issu de la Classe Date

Astuce : Ne pas oublier d'importer le package où se situe la classe `SimpleDateFormat`

```
import java.util.Date;
```

Pour entrer une date :

```
SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
            //création d'un objet sdf qui est du format dd/MM/yyyy

      try {
            unCampeur.setdateNais(sdf.parse("13/10/1970"));
            //parse transforme ma chaîne "13/10/1970" en format date
      } catch (ParseException ex) {
            ex.printStackTrace(); //Si j'attrape l'exception alors affichage
            dans la console de cette exception
            }
```

Pour afficher une date :

```
sdf.format(unCampeur.getdateNais);
//format transforme ma date qui est de type Date dans un format chaîne de type "
dd/MM/yyyy "
```

LES TESTS ....

Une des nombreuses ressources pour les tests : http://www.jmdoudoux.fr/java/dej/chap-junit.htm#junit-1

LA JAVADOC ....

https://openclassrooms.com/courses/presentation-de-la-javadoc