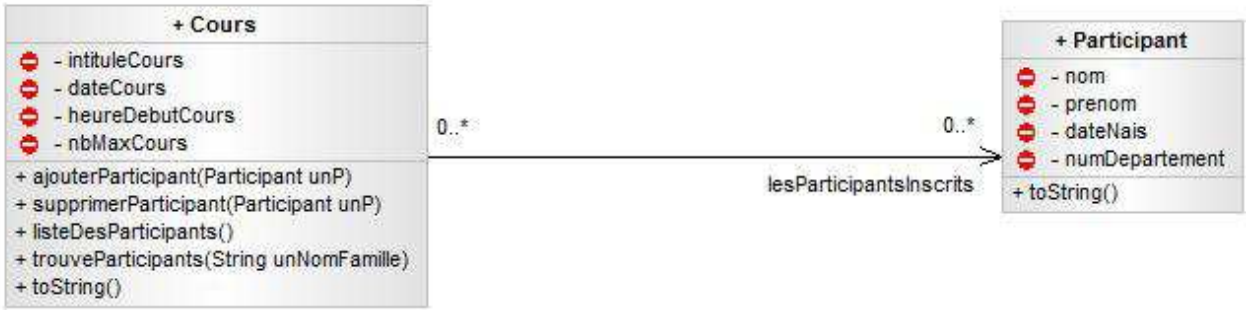
TP02 : LES COLLECTIONS

Objectif du TP :

- Créer une collection en JAVA
- Tester cette collection avec le Framework Junit.

CONTEXTE :

Dans la petite station de ski de « LansLeVillage », le responsable souhaite actuellement un programme afin de gérer les cours de ski des enfants débutants jusqu'à 15 ans.
Il souhaite avoir la liste des cours pour toute la saison d'hiver. Ces cours d'une heure sont caractérisés par la date, l'heure de début, le nombre maximum de participants et la liste des enfants y participant.
Il souhaite donc avoir des informations sur les enfants (nom, prénom, date de naissance et codePostal)



Utiliser Git pour gérer vos différentes versions :
Ajouter ce nouveau code dans votre application du TP1 (ayez ***commité*** votre TP1 un fois validé)

TRAVAIL à faire : Quelle nouvelle classe faut-il créer ?

COMPLETER LA CLASSE COURS

1) Les variables d'instance

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

- L'intitulé du cours
- Sa date et Son heure
- Et le nombre maximum de participant

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

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 les accesseurs SET :

- pour pouvoir modifier le nombre maximum de participant (si celui-ci est supérieur au nombre maximum initial)
- pour modifier la date et l'heure du cours.

3) La Collection (ArrayList)

Créer la collection dans la classe cours qui va permettre pour un cours de connaître tous les participants inscrits. La nommer « ***lesParticipantsInscrits*** »
Cette collection sera une propriété privée comme les autres variables.

Bien choisir le type adéquat pour cette collection typée : collection de quoi ?

Faut-il créer des accesseurs GET et SET pour cette collection ?

```
/*Créer l'ACCESSEUR GET uniquement ➔ cela nous servira lors des tests*/

public ArrayList<Participant> getLesParticipantsInscrits(){
    return lesParticipantsInscrits;
}
```

4) Créer le constructeur afin de prendre ne compte toutes les variables y compris la collection.

Créer le constructeur paramétré.

5) Les méthodes pour pouvoir remplir ou vider cette collection.

```
public boolean ajouterParticipant(Participant unP){
    *vérifier si'il reste de la place dans ce cours
    * renvoyer vrai si l'inscription a pu avoir lieu
    * faux sinon*/

}
public void supprimerParticipant(Participant unP){
    /*Traiter le cas où il ne trouve pas cet objet dans la liste*/

}
```

6) Faire la méthode qui permet de lister tous les participants du cours.

Cette méthode va parcourir toute la liste et afficher chaque participant.

Bien regarder comment on avait affiché les participants.

Compléter cette méthode `listeDesParticipants()` qui permet de retourner une chaine composée de tous les participants inscrits à ce cours.

7) Faire la méthode qui permet de rechercher tous les participants dans ce cours grâce à leur nom de famille.

Cette méthode va parcourir toute la liste des participants inscrits et comparer chaque participant avec le nom de famille donné en paramètre.
On alimentera une nouvelle liste de résultat qui contiendra tous les participants ayant ce nom (cas d'une fratrie)

Compléter la méthode `TrouveParticipants(String unNomFamille)` qui permet de **retourner une nouvelle liste** contenant tous les participants ayant ce même nom de famille.

8) Faire la méthode `toString()` qui permet d'afficher les inforamtions du cours uniquement. Changer la méthode `listeDesParticipants` pour en tenir compte.

TESTER LA CLASSE COURS (CF TEST DE LA CLASSE PARTICIPANT)

1) Création de notre TEST JUnit pour la Classe Cours

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

Faire Clic droit, puis new JavaClass et taper CoursTest

Cette classe hérite de TestCase (donc rajouter **EXTENDS** TestCase)

Mettre le import afin d'importer le framework Junit TestCase (***ALT+entrée*** si besoin)

Créer le **constructeur** de cette classe de **Test**

```
public CoursTest(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 !

2) TEST de l'ajout d'un participant à un cours

a) Création de la méthode test… et utilisation de la méthode AssertEquals

```
public void testAjoutParticipant() throws Exception {

/*Créer un nouveau participant nommé TAURAND*/

/*créer un nouveau Cours intitulé Ski Débutant du 15/12/2015 à 13 heures
avec un nombre maximum de 10*/

/*AJOUTER ce participant à ce cours*/

/*récupérer TOUTE la liste grâce à l'accesseur getLesParticipantsInscrits*/

/*Vérifier que dans la liste à l'indice 0, le nom du participant correspond
bien à TAURAND*/
assertEquals("Insertion non effectuee",nomDuParticipant_récupéré_dans_la
liste_indice_0,"TAURAND");
}
```

b) Lancement du TEST !

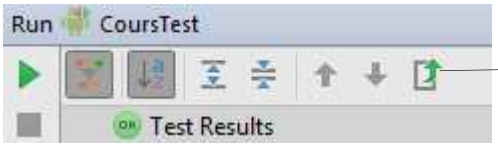
Pour lancer votre test, clic droit puis **RUN CoursTest** (Ctrl+Maj+F10)

- ⇨ 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 (utiliser le simulateur de machine Android (GennyMotion))

BRAVO, votre test a bien fonctionné !

EXPORTER votre test pour en garder une trace :



Génération d'un fichier qui garde la trace de vos tests !
Bonne idée pour le Portefeuille de compétences

MAIS, Vous avez oublié de faire un autre TEST ?

c) Autre TEST à faire !

Pour être sûr que cela fonctionne, il faut vérifier aussi que **l'insertion ne se fait pas quand le nombre de participants est dépassé.**

A vous de voir comment faire ?

3) TEST de la suppression d'un participant à un cours


Sur le même principe, tester la suppression d'un participant à un cours.

4) TEST de la recherche d'un ou plusieurs participants avec son nom de famille


Sur le même principe, tester la recherche d'un participant à un cours :

- **Rechercher un participant qui a un nom unique**
- **Rechercher dans le cas où 2 participants ont le même nom de famille** (cas d'une fratrie par exemple)

COMMENT GERER L'INTEGRALITE DES COURS DE LA SAISON ?

REPRISE DU CONTEXTE :

Dans la petite station de ski de « LansLeVillage », le responsable souhaite actuellement un programme afin de gérer les cours de ski des enfants débutants jusqu'à 15 ans.
Il souhaite avoir la liste des cours pour toute la saison d'hiver.

1) Comment faire pour gérer la liste de tous les cours ?

Actuellement, les cours sont indépendants et ne sont pas regroupés en une seule entité. Donc difficile de faire des recherches etc…


Gérer la liste de tous les cours de la saison donnant la possibilité d'ajouter un cours dans cette liste, de supprimer un cours et de rechercher un cours suivant sa date de cours. Une autre recherche pourra être faite suivant son intitulé.

2) Faire les tests nécessaires pour vérifier la validité de toutes ces nouvelles méthodes.