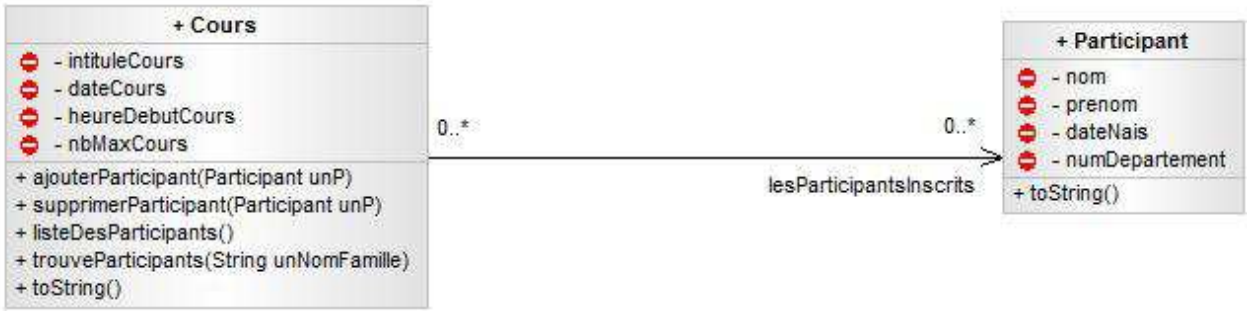
TP04 : L'HERITAGE

Objectif du TP :

- Principe de l'héritage
- Principe de la redéfinition (constructeur et méthode)
- Notion de polymorphisme
- Utilisation d'une classe abstraite

CONTEXTE ACTUEL :

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)



EVOLUTION DU CONTEXTE:

La Station de LansLeVillage souhaite proposer des cours collectifs et individuels aux participants.
Tous les cours ont un intitulé, une date et une heure pour le début du course et un tarif horaire. La durée du cours peut varier de 1 heure jusqu'à 2h30.
Les cours sont soit collectifs soit individuels, il n'y a pas d'autre type de cours.
Pour les cours collectifs, le nombre maximum de participants est obligatoire.
Pour les cours individuels, le nom et prénom du Moniteur sera indiqué ainsi que le nom et le prénom du participant.

La liste de tous les cours individuels et/ou collectifs de la saison de ski devront être consultés. Nous pourrons rajouter des cours et supprimer un cours.

ANALYSER LE NOUVEAU CONTEXTE

TRAVAIL à faire : **SUR PAPIER ou WINDESIGN**

1) Déterminer les nouvelles classes nécessaires pour gérer ces cours individuels/collectifs

Compléter le diagramme de classe avec la classe nommée `CoursCollectifs` et l'autre classe nommée `CoursIndividuels`

2) Faites le lien d'héritage adéquat

Préciser le lien d'héritage avec ces 2 nouvelles classes

3) Où stocker la nouvelle liste de tous les cours de la station ?

Créer la collection dans une nouvelle classe nommée `PlanningSaisonHiver` qui va permettre de gérer tous le cours qu'ils soient individuels ou collectifs.

CODER LE NOUVEAU CONTEXTE : PRINCIPE DE L'HERITAGE

TRAVAIL à faire : **avec l'IDE Android Studio**

1) Créer la classe CoursCollectifs

Copier votre classe Cours actuelle en `CoursCollectifs` (clic droit sur la classe Cours, Refactor puis Copy)

2) Modifier la classe Cours.

La classe Cours va être à présent une classe abstraite (`abstract`) afin de spécifier juste les propriétés et méthodes ***communes*** à TOUS les cours.

Changer cette classe `Cours` et passer tout en `Abstract`.

- Passer les propriétés communes en **`protected`**
- Passer la classe en abstract : **`public abstract class`** `Cours {`
- Passer chaque méthode en abstract :
  `public abstract boolean` `ajouterParticipant(Participant unP);`

3) Modifier la classe Courscollectifs

La classe `Courscollectifs` va donc **hériter** de la classe `abstraite Cours`.

Changer cette classe `Courscollectifs`

- Garder juste les propriétés spécifiques à cette classe en **`private`**
- Modifier le constructeur afin de prendre en compte le constructeur de la classe mère.
- Que faut-il changer autrement ?

Changer le TEST de la classe `Cours`

- Regarder la classe `CoursTest`, des erreurs apparaîssent…
- Renommer la classe `CoursTest` en `CoursCollectifsTest`
- Corriger les erreurs afin que cette classe de Tests soit encore valide.

4) Créer la classe CoursIndividuels

La classe `CoursIndividuels` va aussi **hériter** de la classe `abstraite Cours`.

Créer cette nouvelle classe `CoursIndividuels`

- créer les 2 propriétés spécifiques à cette classe en **`private`** afin de gérer le *participant* et le *moniteur*
- Modifier le constructeur afin de prendre en compte le constructeur de la classe mère *(juste le moniteur sera connu lors de la création du cours individuel)*
- Redéfinir la méthode **`public boolean`** `ajouterParticipant(Participant unP){`
  `vérifier si le cours n'est pas encore affecté à un Participant`
  `* renvoyer vrai si l'inscription a pu avoir lieu`
  `* faux sinon*/`
- Redéfinir la méthode **`public void`** `supprimerParticipant(Participant unP){`
  `vérifier si le participant passé en paramètre est bien celui inscrit au cours si oui, le supprimer c'est à dire que le cours devient à nouveau libre !`
- Redéfinir la méthode **`public String`** `toString()){`
  `Afficher le cours, le moniteur et le participant s'il existe.`
- Redéfinir la méthode **`public String`** `listeDesParticipants(){`
  `Affiche le participant s'il existe sinon retourne le message` **`"aucun participant affecté à ce cours"`**

PASSER le TEST ***fourni*** de la classe `CoursIndividuels`

- **Récupérer** sur Partage la classe `CoursIndividuelsTest`
- Passer cette classe de Tests tant que vous avez des errreurs.
- Corriger vos erreurs sur la classe `CoursIndividuels` si besoin.

COMPRENDRE LE PRINCIPE DU POLYMORPHISME

rappel : Le ploymorphisme (ploy : plusieurs, morpho : forme). Donc ici, le cours (classe mère) peut prendre plusieurs formes soit être individuel soit être collectif. (ses 2 classes filles)

1) Création d'une nouvelle Classe `PlanningSaison`

- Créer les 2 nouvelles propriétés pour gérer le planning de la station de ski (date début et date de fin de la saison)
- Gérer le fait que ce planning comporte la planification des cours de la saison

2) Création des méthodes :

```
public ajouterUnCours(Cours c) {
   /*cette méthode doit ajouter un cours en vérifiant que la date du cours est bien
   compris dans les dates du planning
   De plus, un cours peut être soit collectif soit individuel => quelles conséquences
   sur cette méthode ?*/

// A compléter
}
public supprimerUnCours(Cours c) {
   /*cette méthode doit supprimer un cours. Attention, la suppression du cours entraîne
   la suppression des participants qui le compose */
// A compléter
}
public listeDesCoursCollectifs() {
   /*cette méthode doit lister JUSTE les cours collectifs. On n'affichera pas ici les
   participants inscrits pour chaque cours*/
// A compléter
}
public listeDesCoursIndividuels() {
   /*cette méthode doit lister JUSTE les cours individuels. On n'affichera pas ici les
   participants inscrits pour chaque cours*/

// A compléter
}
public listeDesCours() {
   /*cette méthode doit lister tous les cours qu'ils soient collectifs ou individuels
   en UTILISANT les 2 méthodes ci-dessus*/

// A compléter
}
```

3) Etude de la classe de Test

Soit la classe de Test suivante à étudier :

- Etude de la méthode `setUp` :
  - Quels sont les 4 différents tests que je vérifie ?
  - Identifier le **polymorphisme** dans cette classe de test
  - **Créer un cours c4 afin de tester une date comprise dans la saison ! gérer le assert correspondant**
- Etude de la méthode `testAjouterCours` : bien comprendre les assert !
- Etude de la méthode `testSupprimerCours` : bien comprendre les assert !

```
public class PlaningSaisonTest extends TestCase {

    private static final String DATE_PATTERN = "dd/MM/yyyy";
    private SimpleDateFormat sdf =null;
    private Participant m = null;
    private Cours c,c1,c2,c3 = null;
    private PlaningSaison ps = null;

    public PlaningSaisonTest(String testMethodName) {
        super(testMethodName);
    }

    protected  void setUp(){
        sdf = new SimpleDateFormat(DATE_PATTERN);
        try {
            ps= new PlaningSaison(sdf.parse("01/12/2015"),sdf.parse("30/03/2016"));

            c = new CoursCollectifs("Ski debutant", sdf.parse("01/12/2015"), 13, (byte)
10);
            m = new Participant("CHIOGGIO", "Roberto", sdf.parse("11/06/1966"), (byte)
0);
            c1= new CoursIndividuels("rando itineraire du
Loup",sdf.parse("30/03/2016"),12,m);
            c2 = new CoursCollectifs("Ski debutant adulte", sdf.parse("11/11/2015"),
13, (byte) 20);
            c3 = new CoursCollectifs("Ski debutant2 adulte", sdf.parse("31/03/2016"),
13, (byte) 20);

        } catch (ParseException e) {
            e.printStackTrace();
        }
        assertFalse("Date du cours anterieure à la date de DEBUT de la saison",
ps.ajouterCours(c2));
        assertFalse("Date du cours superieure à la date de FIN de la saison",
ps.ajouterCours(c3));

        assertTrue("COURS collectif non ajoute",ps.ajouterCours(c));
        assertTrue("COURS individuel non ajoute", ps.ajouterCours(c1));
    }

    protected  void tearDown(){
        sdf = null;            m = null;            c1 = null;        c = null;
        c2 =null;              c3= null;            ps = null ;
    }

    public void testAjouterCours() throws Exception {
        assertNotNull("cours collectif non cree", c);
        assertNotNull("cours individuel non cree",c1);
        /*test de l'ajout d'un cours COLLECTIF dans le planing de saison*/
        assertEquals("Insertion du cours collectif non effectuee",
ps.getLesCoursSaison().get(0).toString(), c.toString());
        /*test de l'ajout d'un cours INDIVIDUEL dans le planing de saison*/
        assertEquals("Insertion du cours individuel non effectuee",
ps.getLesCoursSaison().get(1).toString(), c1.toString());
    }
    public void testSupprimerCours() throws Exception {
        testAjouterCours();
        /*test de la suppression du cours COLLECTIF dans le planing de saison*/
        ps.supprimerCours(c);
        //ici on supprime le cours collectif, donc decalage
        assertEquals("Suppression du cours collectif non effectuee",
ps.getLesCoursSaison().get(0).toString(), c1.toString());
    }
}
```

REALISER UNE SUITE DE TEST !

```
import junit.framework.Test;
import junit.framework.TestSuite;

public class TestSuitePlanning {
    public static Test suite() {
        TestSuite suite = new TestSuite();
        suite.addTestSuite(<classe de Tests>.class); /*ligne à modifier et à rajouter*/
        return suite;
    }
}
```

A partir de l'exemple ci-dessus, réaliser la suite de Tests permettant d'obtenir le compte-rendu de tests ci-dessous :

TestSuitePlanning: 11 total, 11 passed 55 ms

Collapse | Expand

com.example.frederiquederobien.projetfr_initiationjava.ParticipantTest		0 ms
testGetNom	passed	0 ms
testToString	passed	0 ms
com.example.frederiquederobien.projetfr_initiationjava.CoursCollectifsTest		2 ms
testAjouterParticipant	passed	1 ms
testSupprimerParticipant	passed	0 ms
testTrouveParticipants	passed	1 ms
com.example.frederiquederobien.projetfr_initiationjava.CoursIndividuelsTest		27 ms
testAjouterParticipant	passed	26 ms
testChangerMoniteur	passed	0 ms
testSupprimerParticipant	passed	1 ms
testVerifMoniteur	passed	0 ms
com.example.frederiquederobien.projetfr_initiationjava.PlaningSaisonTest		26 ms
testAjouterCours	passed	0 ms
testSupprimerCours	passed	26 ms

Generated by Android Studio on 24/09/15 11:54