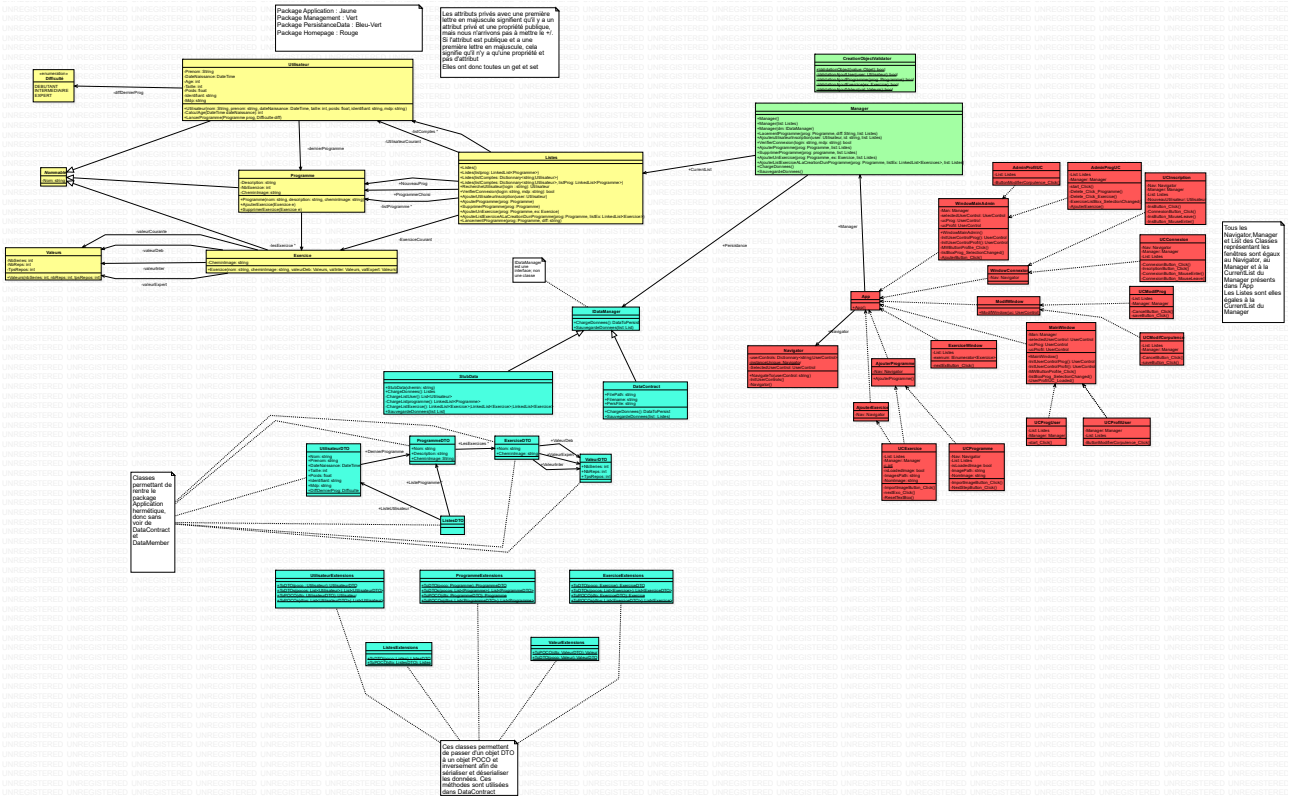
DOCUMENTATION PARTIE CONCEPTION ET PROGRAMMATION ORIENTÉE

◆ Diagramme de classe



Rappel : Les attributs privés avec une première lettre en majuscule signifient qu'il y a un attribut privé et une propriété publique, mais nous n'arrivons pas à mettre le +/.
Si l'attribut est publique et a une première lettre en majuscule, cela signifie qu'il n'y a qu'une propriété et pas d'attribut. Elles ont donc toutes un get et set

Notes sur les classes :

- La classe Nommable est une classe abstraite qui permet de ne pas avoir à insérer un nom dans chaque classe qui peut en avoir un. Elle contient donc une propriété Nom.

- Un Utilisateur a plusieurs propriétés qui seront affichées sur la page de profil (nom, prénom, âge calculé grâce à la méthode CalculAge qui reçoit la date de naissance de l'utilisateur en paramètre, la taille et le poids). Lorsque l'utilisateur lance un Programme avec une difficulté avec la méthode LancerProgramme, celui-ci devient

son DernierProgramme représentant donc le dernier programme effectué et la difficulté devient sa DiffDernierProg.
A l'inscription, l'utilisateur doit choisir un identifiant unique ainsi qu'un mot de passe.

- La classe Programme représente un Programme sportif. Celui-ci possède un nom, une description ainsi qu'une liste d'Exercice qui seront affichés. Le nombre d'exercice d'un Programme peut-être défini de 2 manières différentes : soit le Count() de la liste, soit une valeur passée en paramètres dans un Setter. On peut Ajouter et Supprimer un exercice dans la liste depuis la vue par l'utilisateur ayant l'identifiant « admin », donc par un Administrateur.

- La classe Exercice possède 4 valeurs : valeurDeb pour la difficulté DEBUTANT, valeurInter pour la difficulté INTERMEDIAIRE, valeurExpert pour la difficulté EXPERT et valeurCourante qui est la Valeur prise pour l'exercice en fonction de la difficulté qu'un utilisateur a choisi au lancement d'un programme.
  Son image ainsi que son nom seront affichés lorsqu'on affiche un programme, ou lorsqu'on le lance.

- La classe Valeur contient 3 entiers : un nombre de séries, un nombre de répétitions et un temps de repos qui varieront en fonction de la difficulté.

Il était plus simple de créer une nouvelle classe Valeur et d'affecter ses 4 valeurs à la classe Exercice plutôt que d'ajouter plus de 9 propriétés de type int à la classe Exercice.

- La classe Listes est divisée dans le code en deux parties ce qu'il fait qu'elle est partielle. La classe Listes tout court contient un Dictionnaire listCompte prenant en clé un identifiant et en valeur un utilisateur. Celui-ci nous permettra à la connexion de rechercher si le login rentré correspond bien à un Utilisateur avec la méthode RechercherUtilisateur, puis de vérifier si le mot de passe rentré correspond bien au login via la méthode vérifierConnexion.
  Elle contient aussi la LinkedList de tous les programmes listProgramme.
  La classe Listes.DonneesCourantes contient l'UtilisateurCourant, l'ExerciceCourant qui est l'exercice qui doit être effectué par l'Utilisateur ainsi que le ProgrammeChoisi pour faciliter le DataBinding. Sa méthode LancementProgramme met à jour la valeurCourante des exercices contenus dans la liste des exercices du programme passé en paramètre en fonction de la difficulté passé en paramètre et appelera la méthode LancerProgramme d'un Utilisateur.

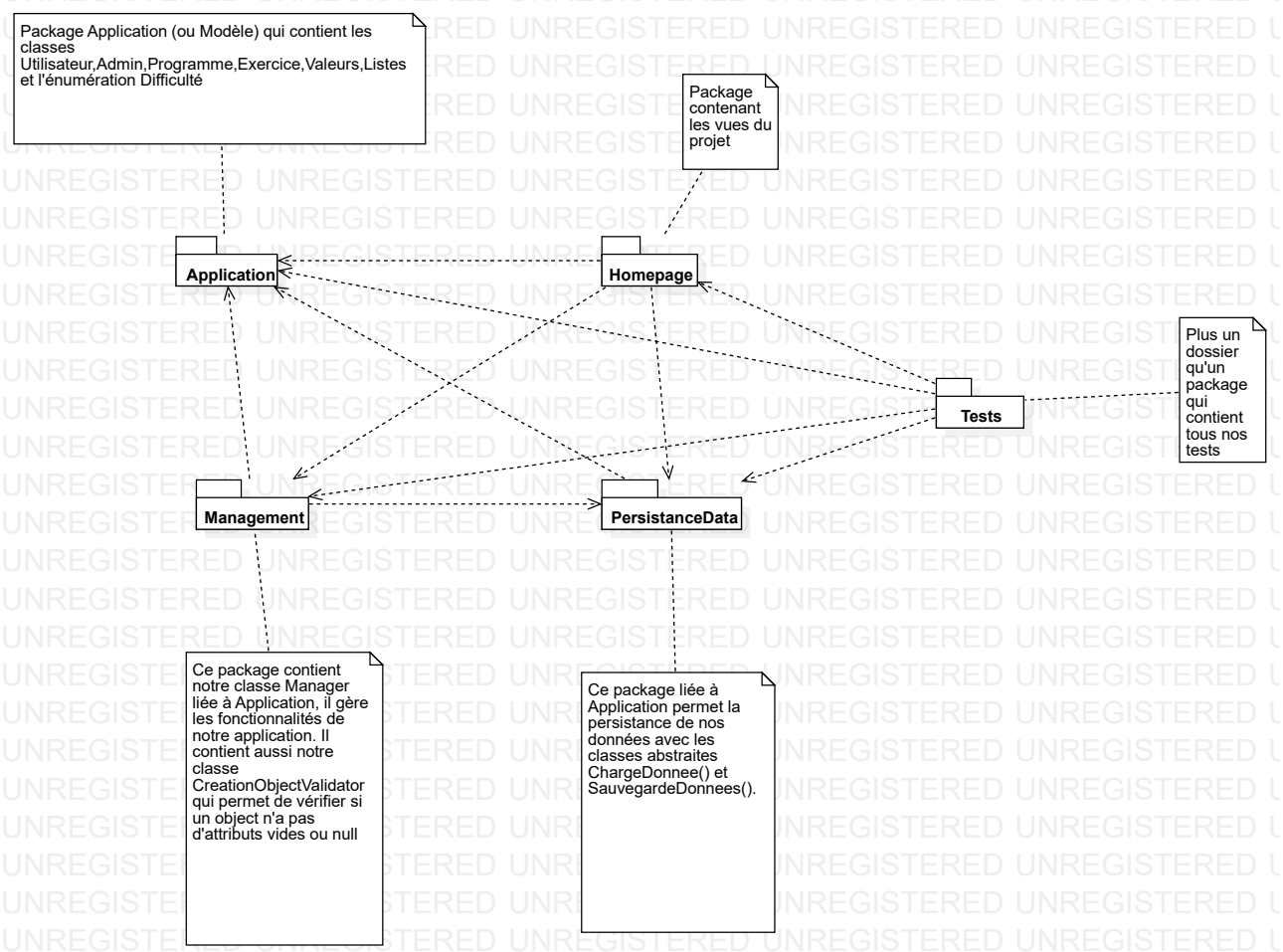
Liste la classe principale pour la sérialisation. Elle fait le lien entre le projet Application, Management et Persistance (cf. documentation projet tuteuré). De plus, une instance de la classe Listes est généralement notre DataContext pour la vue (objets courants et listes d'objets).

- La classe Manager est divisée en 3 classe partielle pour que chaque classe ait sa fonctionnalité (Manager.Persistance, Manager.Programme, Manager.Utilisateur). Elle contient la Liste courante. Cette classe permet d'appeler les différentes méthodes de

la classe Listes en effectuant auparavant un contrôle. Elle permet de plus de gérer la persistance (cf. documentation projet tuteuré). Ces méthodes sont celles qui vérifient et valide les objets de Classe que nous avons créés et appelle les méthodes de la classe Listes qui portent le même nom qu'elles.

- La classe CreationObjectValidator ne contient que des méthodes static qui renvoient un bool. Elles vérifient si les attributs des objets passé en paramètres sont correctes (pas vides pour les string ou pas trop élevé/bas pour les nombres). Elle est essentielle pour vérifier que les valeurs rentrées sont correctes avant d'ajouter un programme ou un exercice.

◆ Diagramme de paquetage



◆ Architecture du projet

- Description écrite de l’architecture

Nous allons décrire notre application en plusieurs étapes, nous allons tout d’abord parler de son architecture puis des patrons de conceptions et enfin des différentes dépendances entre les projets de l’application.

Pour commencer, notre application se divise en quatre gros projets distincts a qui s’ajoutent des projets de tests.

Les gros projets sont respectivement « Homepage» qui contient tous les éléments visuels (nous changerons le nom dans le futur). En effet, le projet contient différents user-control qui sont répartis dans l’espace de notre fenêtre principale. Il contient aussi une classe Navigator qui permet une navigation entre nos différents user-control afin de fluidifier la navigation. Nous avons aussi ajouté certains éléments qui dépendent du statut de connexion de l’utilisateur de l’application.

Il y a ensuite le projet « Application» qui contient les différentes classes qui sont à la base de l’application. Il y a les classes Utilisateur, Programme, Admin, Exercice , Valeurs ainsi qu’une énumération Difficulté.

Il y a le projet Management qui est le gestionnaire de notre application. Il contient une classe Manager qui contient la plupart de notre nos fonctionnalités telles que la création et suppression d’un Programme. La création d’un compte et la connexion à un compte, etc… Ce projet contient aussi une classe avec des méthodes statiques qui contrôle la création d’un utilisateur, d’un programme,d’un exercice et d’une valeur.

Les différents tests vérifient que les méthodes des classes et les interactions entre les différentes classes soient fonctionnels et ne présentent aucun bug.

- Description patrons de conceptions

Nous utilisons un patron de conception. C’est est un singleton, il est utilisé dans notre classe Navigator dans le projet Homepage. Il fait en sorte qu’une seule instance du Navigator soit instanciée dans notre code-behind. Il permet d’avoir accès à cette même instance à l’aide de la méthode GetInstance() ; cela nous permet de contrôler qu’un seul Navigator soit instancié afin d’avoir une navigation totalement fonctionnelle.

- Description des dépendances

Enfin, nous allons parler des dépendances entre nos différents projets.

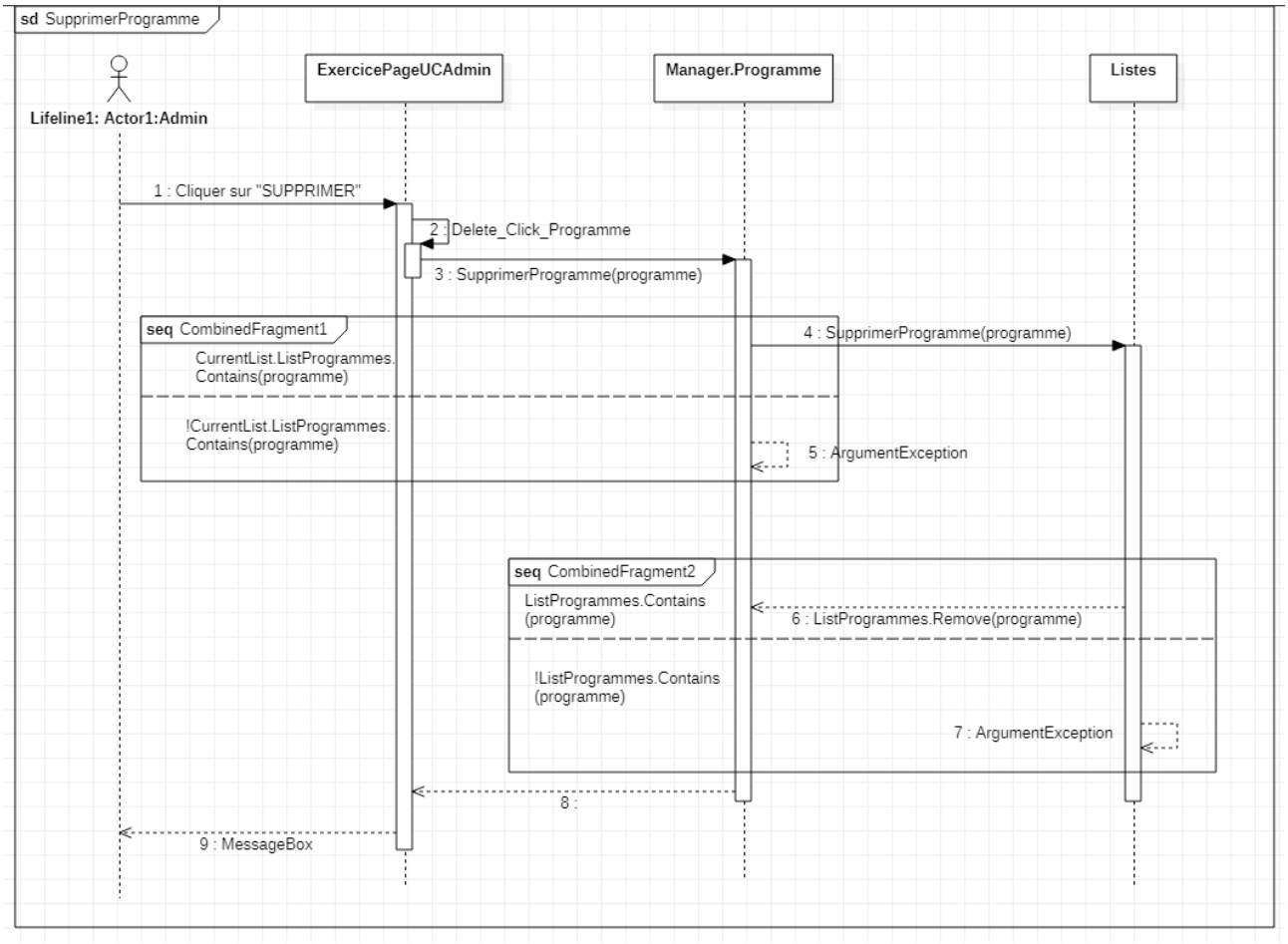
La vue Homepage est liée à la Persistance pour charger un Stub ou des données serialisées, elle est liée au Modèle (nommé Application) pour instancier des objets auxiliaires lors de la création de programme ou utilisateurs ainsi qu’à Management pour instancier un Manager qui va permettre d’utiliser toutes les fonctionnalités prévues pour notre application.

Management possède des dépendances envers Application car il contient des listes de programmes et d’utilisateurs ainsi qu’à Persistance car il récupère un IDataManager.

Application n’a aucune référence vers d’autres projet car il est la racine de l’application et ne dépend de rien pour exister.

Les Test ont des dépendances envers tous les projets excepté la vue afin de tester toutes nos fonctionnalités.

- **Diagramme de séquence**



Voici le fonctionnement lorsqu'un admin clique sur le bouton supprimer d'un programme :
Le bouton possède un évènement Delete_Click_Programme qui va appeler SupprimerProgramme de Manager.Programme avec en paramètre programme qui est le programme courant/sélectionné. Si la liste courante contient le programme courant, on appelle SupprimerProgramme de Listes sinon on retourne une argument exception.
Dans Listes, si ListProgrammes contient le programme, on fait appel à la méthode .Remove sinon on retourne une argument exception. On retourne enfin une messageBox pour l'admin. On vérifie ensuite si le nombre de programme dans la liste de programme est égale à 0, si oui, on set le ContentControl sur la MainWindowAdmin. Enfin on sauvegarde les changements via la méthode Manager.SauvegardeDonnees().

Bonus : Pourquoi chocolatine et pas pain au chocolat ?

Le terme «Pain au chocolat» a commencé à être utilisé au XIXème siècle en France pour désigner des pâtisseries d'inspiration viennoise. En effet, à cette période les échanges culturels entre l'Autriche et la France

sont assez importants. La première "Boulangerie Viennoise" est installée en France dans les années 1830 au 92 rue Richelieu, à Paris et est dirigée par un autrichien : Auguste Zang. C'est lui qui va véritablement apporter à Paris la mode des viennoiseries.

L'hypothèse la plus probable de l'origine du nom "Chocolatine" viendrait donc de cet autrichien. Car en l'entendant vendre des "Schokoladencroissant" avec son accent autrichien, les français auraient progressivement transformé le mot en "Chocolatine croissant", puis simplement "Chocolatine".
Il est donc probable que le premier terme pour désigner une viennoiserie fourrée au chocolat ait été "Chocolatine", à cause de cette déformation linguistique. Et c'est d'ailleurs plutôt logique puisque la particularité de cette viennoiserie est surtout d'être au chocolat, et qu'elle a rapidement perdu sa forme de croissant.
Quand au terme "Pain au chocolat", il serait plus récent. D'après Nicolas Berger, auteur d'une encyclopédie du chocolat ("Chocolat, mots et gestes") publié en 2013 aux éditions Alain Ducasse, le mot "Pain au chocolat" désignait à l'origine un morceau de pain dans lequel on fourrait un bout de chocolat pour le goûter des écoliers.
Lorsque les viennoiseries ont été reprises et réinterprétée par les pâtissiers français au début du XXème siècle, en utilisant notamment de la pâte levée feuilletée, certains auraient repris ce terme. Progressivement, l'utilisation du mot d'origine "Chocolatine" n'a été conservée que dans le Sud-Ouest, probablement à cause de sa proximité avec des mots occitans.
Mais l'original était bien "Chocolatine" !