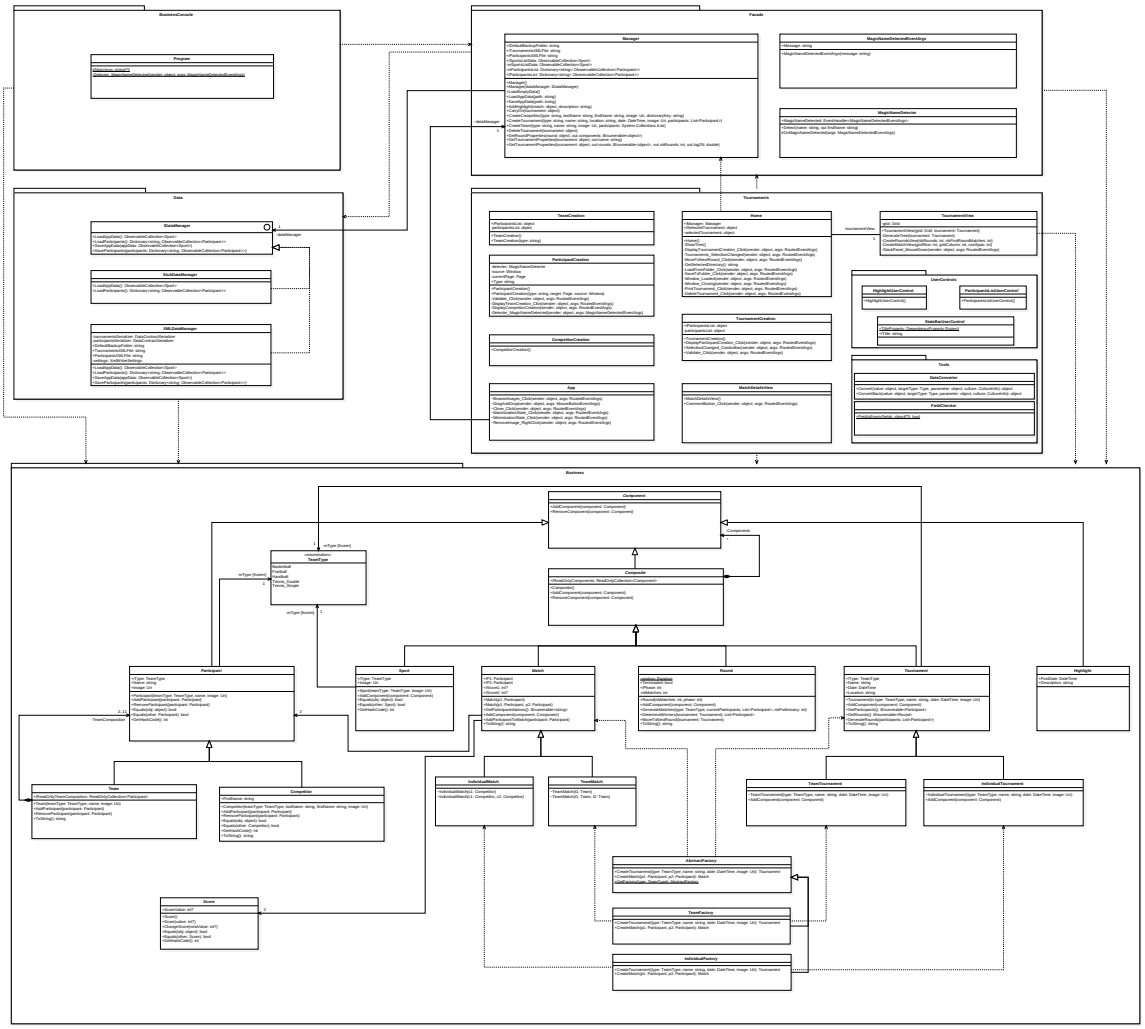
Tournaments

Diagramme de classes



Le diagramme original est aussi disponible en PDF dans le dossier intitulé *Diagrammes UML*.

Description du diagramme de classes

La solution se divise en différents projets qui ont chacun une responsabilité unique. Cette division des responsabilités permet une meilleure maintenabilité de l'application.

Tournaments est notre projet WPF. Il gère la vue et les évènements. Il est également décomposé en dossiers qui regroupent les éléments par finalité (la création d'un participant, la création d'un tournoi...) pour une meilleure visibilité. Le dossier UserControls contient les *contrôles utilisateur* qui permettent de n'avoir qu'un composant et éviter la duplication de code XAML.

Business est une bibliothèque de classes qui représente le métier de notre application. Nous avons implémenté plusieurs patrons de conception, notamment une *fabrique abstraite* qui permet de créer des environnements d'objets : un *IndividualTournament* (resp. *TeamTournament*) ne peut contenir que des *Round* contenant des *IndividualMatch* (resp. *TeamMatch*). Il y a également deux patrons *composite*. Dans l'ensemble, la façade mentionnée plus bas ne connait que les classes abstraites (sauf pour *Participant* et ses filles), ce qui l'oblige à passer par la fabrique abstraite pour créer les tournois et matches. Un sport contient une liste de tournois, qui se composent à la fois de *Round* et de *Participant*. Ces derniers sont nécessaires pour calculer le nombre de tours et de matches que l'arbre va contenir. Un Round se compose de matches et ces derniers contiennent deux participants, ainsi que deux scores. L'implémentation du composite nous a donc permis de n'avoir qu'une seule collection à gérer à travers la classe *Composite*, qui contient une collection de *Component*. L'implémentation du composite sur participant permet d'avoir une possibilité d'évolution de la composition d'une *Team*, qui se compose pour l'instant de *Competitor*. Il y a une couche d'abstraction qui oblige l'entité qui utilise le métier à passer par la fabrique abstraite pour les instancier, les constructeurs des classes filles étant internes. Cela permet de contrôler la fabrication des objets et ne pas créer des tournois individuels avec des matches par équipe. Cette encapsulation n'est pas possible avec *Participant* et ses filles car il n'a pas été possible de les intégrer à la fabrique abstraite, certains sports se composant à la fois de *Competitor* et de *Team*.

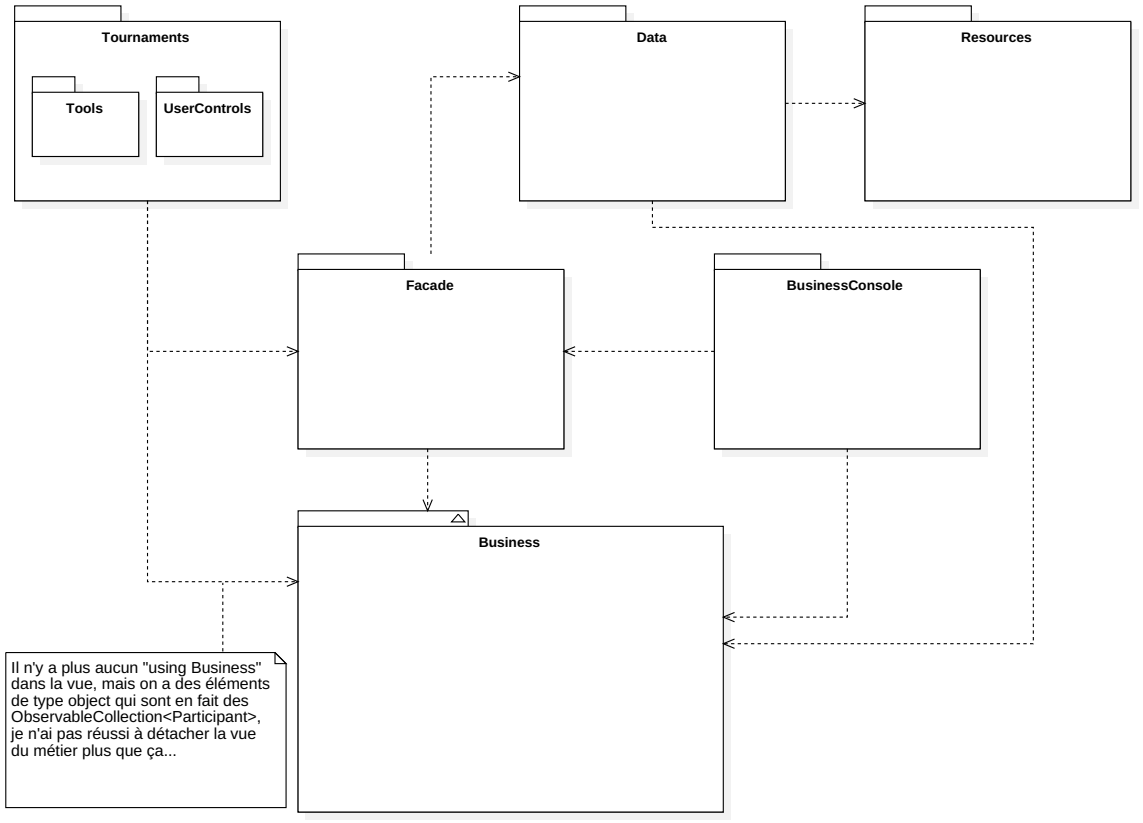
BusinessConsole est une application console qui permet de tester le métier rapidement avant de l'utiliser dans le projet WPF. C'est un gain de temps puisque cela nous évite tous les problèmes liés au binding.

Resources est une bibliothèque de classes qui regroupe les images qui sont ainsi dans une dll à part. Cela est pratique pour la maintenance : si une refonte graphique a lieu, par exemple, il suffit de déployer la dll uniquement.

Data est une bibliothèque de classes qui représente nos données. Elle regroupe principalement nos données de test pour le moment sous la forme d'un *stub*. Plus tard, nous nous en servirons pour charger et sauvegarder les données sérialisées.

Facade est la façade de notre projet. L'utilisateur passe par une façade pour créer ou charger les objets dont il a besoin sans avoir l'implémentation des classes et méthodes, ce qui limite le risque de dysfonctionnements liés à une mauvaise utilisation du métier. C'est le but de la façade, qui permet de donner à l'utilisateur un point de passage unique à un sous-système complexe. Le client passera donc par la façade pour effectuer les cas d'utilisation listés dans notre diagramme de cas d'utilisation.

Diagramme de paquetages



Description du diagramme de paquetages

Pour isoler le métier du reste de l'application, nous avons décidé de mettre en place une façade dans un package propre à lui. La vue ne dépend du métier que pour l'Enum et les listes de Participant, autrement elle passe par la façade pour réaliser les actions métier et récupérer les données qui sont gérées par *Data*. Data dépend du métier pour pouvoir créer les bonnes instances à fournir à l'application et des ressources qui sont les images par défaut de l'application. L'application console dépend, comme la vue, de la façade et du métier pour les mêmes raisons. Il est pour l'instant compliqué de complètement séparer le métier de la vue et de l'application console en raison de l'Enum, qui est indispensable pour qu'un participant pour un sport donné ne se retrouve pas à participer à des tournois d'autres sports.