

El ayadi Ashraf

SOMO TOCHE Steve-Marley



Rapport de projet Tennis

jUNiA ISEN

Table des matières

Introduction :	3
Présentation du projet :	3
Organisation du projet et diagramme de classe :.....	4
Exemple de fonctionnement (notice) :	8
Difficultés rencontré :.....	10
Conclusion :	11

Introduction

Dans le cadre du module de Programmation Orientée Objet, nous avons développé une application Java permettant de simuler un match de tennis.

L'objectif du projet était d'appliquer l'ensemble des notions vues en cours tout en adoptant une démarche de développement claire, structurée et conforme aux bonnes pratiques.

Travailler en binôme nous a permis de répartir les tâches de manière cohérente et de confronter nos approches pour construire une architecture fiable. Le projet nous a donné l'occasion de mobiliser les concepts fondamentaux de la POO : encapsulation, héritage, énumérations, composition, gestion simple des exceptions, documentation Javadoc et respect des règles de nommage.

Nous avons choisi de modéliser le tennis car les règles du sport sont naturellement structurées : points, jeux, sets et match. Cette hiérarchie se traduit très bien en classes, ce qui nous a permis de concevoir un programme lisible et bien organisé, tout en restant fidèle aux principes enseignés.

Présentation du projet

Le but de notre projet était de concevoir une application Java capable de simuler de manière réaliste le fonctionnement d'un tournoi de tennis du Grand Chelem. L'objectif n'était pas de créer un jeu graphique, mais de mettre en pratique les principes clés de la programmation orientée objet à travers une structure propre, modulaire et réaliste.

Le tennis est un sport idéal pour ce type de projet car sa progression est naturellement hiérarchisée :

un échange fait avancer un jeu,
un jeu fait avancer un set,
et plusieurs sets déterminent le gagnant du match.

Cette logique correspond parfaitement à un découpage orienté objet.

Au fil du développement, l'application s'est étendue bien au-delà d'un simple match : elle permet désormais de gérer un tournoi complet avec :

- génération automatique de 128 joueurs hommes et 128 joueuses,
- création et gestion du tableau (1er tour, 2^e tour, 1/8, quarts, demis, finale),
- assignation d'arbitres et présence de spectateurs,
- lancement des matchs en mode manuel, automatique silencieux ou automatique détaillé,
- mise à jour automatique des statistiques des joueurs,
- navigation à travers une interface console structurée par menus,
- gestion des erreurs via try / catch,
- support complet des caractères accentués pour une lecture confortable en console.

Le projet met également en application des concepts importants vus en cours :

- héritage (Personne → Joueur, Arbitre, Spectateur)
- classes abstraites (Joueur)
- interfaces (Supporter, permettant à un joueur d'être aussi spectateur)
- encapsulation stricte (attributs privés et protected)
- exceptions personnalisées et sécurisation des données
- collections Java (ArrayList, listes imbriquées pour les tours)
- piles de statistiques (per-match et per-player)

La structure finale du projet est propre, organisée en plusieurs packages (joueurs, matchs, arbitres, tournoi, statistiques, interface console), ce qui rend l'ensemble cohérent, maintenable et extensible.

Organisation du projet et diagramme de classe

Le projet a été conçu en suivant une organisation structurée, logique et cohérente autour des principes de la programmation orientée objet (POO). L'objectif général était de simuler de manière réaliste la dynamique d'un tournoi de tennis, en représentant non seulement les joueurs, mais aussi les arbitres, les spectateurs, le déroulement des matchs et la progression globale du tournoi. Pour garantir une architecture claire, extensible et maintenable, le code a été réparti en différents packages, chacun correspondant à un domaine fonctionnel précis. Cette organisation favorise la lisibilité, facilite la maintenance et assure une séparation stricte des responsabilités.

Le noyau du projet repose sur la modélisation des personnes réelles. Le package des entités de base contient la classe Personne, qui regroupe les attributs communs : nom de naissance, nom courant, prénom, date de naissance, nationalité, genre, etc. L'utilisation de LocalDate garantit la validité des dates et permet de manipuler l'âge de manière fiable grâce à Period.between. Toutes les entités principales du projet (joueurs, arbitres, spectateurs) héritent de cette classe mère, ce qui permet une factorisation optimale de l'information et évite les répétitions.

L'héritage structure le modèle de manière claire. La classe abstraite Joueur étend Personne et ajoute les attributs et comportements propres à un joueur professionnel : main dominante (enum Main), classement automatique, sponsor, entraîneur et statistiques personnelles. Deux classes concrètes en héritent : JoueurHomme et Joueuse, respectant les contraintes du tournoi (un match Homme oppose deux hommes, et inversement). La classe Arbitre, elle aussi dérivée de Personne, représente l'arbitre responsable d'annoncer le score, de trancher les litiges et de superviser le match. Une spécialisation, ArbitreCentral, enrichit encore ce rôle avec des comportements adaptés.

Une particularité importante du projet concerne les spectateurs. La classe abstraite Spectateur hérite de Personne et implémente l'interface Supporter, qui modélise les réactions possibles dans les gradins (applaudir, crier, huér...). L'utilisation d'une interface permet de représenter plusieurs comportements sans recourir à l'héritage multiple, interdit en Java. De plus, un Joueur peut lui-même devenir supporter lorsqu'il n'est pas sur le terrain, ce qui illustre une application très concrète de ce mécanisme. Deux classes concrètes, SpectateurHomme et Spectatrice, complètent cette partie du modèle.

Le déroulement d'un match suit une hiérarchie fidèle aux règles du tennis. Grâce au principe de composition, un Match est constitué de plusieurs SetTennis, eux-mêmes composés de plusieurs Jeu, contenant chacun une série d'objets Echange. Ce découpage permet à chaque classe de gérer son propre niveau de logique : un échange donne un point, un ensemble de points forme un jeu, les jeux forment les sets, et les sets déterminent le vainqueur du match. Les enums CategorieMatch (simple homme / simple femme) et NiveauMatch (premier tour, huitième, quart, etc.) assurent la cohérence et limitent les erreurs.

Le projet inclut un module complet dédié aux statistiques. Deux classes ont été créées :

- StatistiquesJoueur, pour les données globales (carrière / saison),
 - StatistiquesMatch, pour les données spécifiques à un match (aces, doubles fautes, jeux gagnés...).
- Ce module est indépendant du reste de l'application et pourrait facilement être réutilisé ou étendu.

L'organisation générale du tournoi est gérée par la classe Tournoi, située au sommet du modèle. Elle représente un tournoi du Grand Chelem, défini par une ville, une année, et une surface déterminée automatiquement (gazon, terre battue...). Elle orchestre également la génération automatique de tous les participants : 128 joueurs hommes, 128 joueuses, 10 arbitres et au moins 100 spectateurs. Elle permet de créer le premier tour des tableaux puis de générer les tours suivants selon les vainqueurs. Par souci de clarté dans le diagramme UML, les relations directes les plus complexes ont été simplifiées, sans nuire à la logique.

D'un point de vue technique, plusieurs bonnes pratiques ont été appliquées.

- Utilisation systématique de == pour les types primitifs et equals() pour les objets.
- Factorisation de portions de code à l'aide du ternaire, améliorant la concision sans sacrifier la lisibilité.
- Respect strict de l'encapsulation : attributs privés ou protégés selon leur sensibilité.
- Utilisation fréquente de try/catch pour sécuriser la saisie utilisateur et empêcher tout plantage de l'application.
- Configuration explicite de l'encodage UTF-8, assurant l'affichage correct des caractères accentués dans la console.

Un point essentiel de l'organisation du projet a été la réalisation de tests réguliers au fur et à mesure du développement.

Chaque grande fonctionnalité (gestion des personnes, match, set, jeu, tournoi, statistiques, menu console...) a été testée dans des classes dédiées :

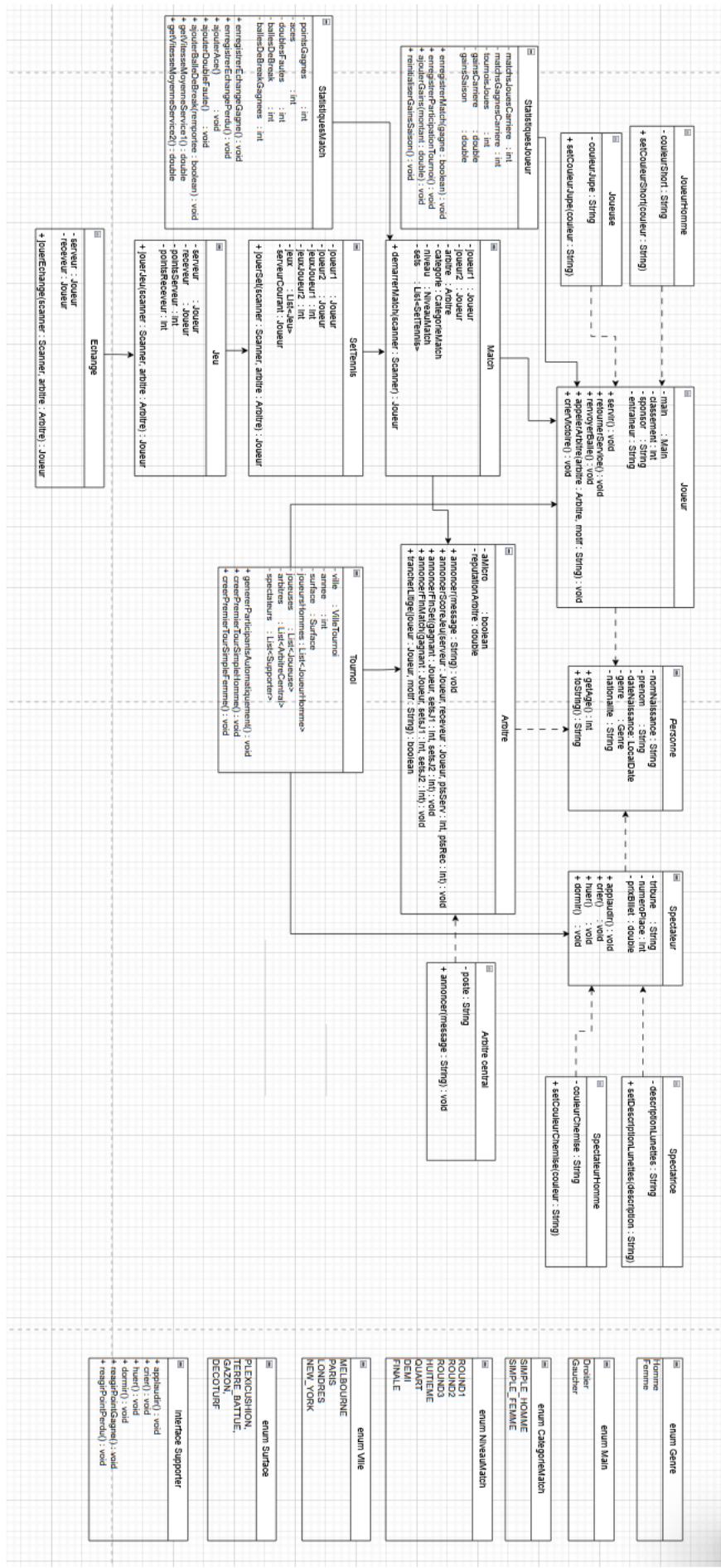
- TestPartie1 : validation des bases (Personne, Joueur, Arbitre, Supporter).
- TestPartie2 : tests complets sur Match, SetTennis, Jeu, arbitrage et logique du tennis.
- TestTournoi : test de la génération du tournoi, des participants et du premier tour.
- TennisApp : test final de l'application complète et de l'interface console.

Ces tests ont permis de :

- Valider progressivement chaque module,
- Corriger les erreurs avant l'intégration finale,
- Éviter les régressions,
- Garantir un fonctionnement robuste à chaque étape du projet.

L'utilisateur peut retrouver cette évolution dans les fichiers de test situés dans le package tennis.app, qui reflètent fidèlement la progression du développement et de la validation du logiciel.

Dans le diagramme de classes, les flèches pleines représentent les relations de composition, tandis que les flèches pointées indiquent les liens d'héritage entre une classe mère et ses sous-classes.



Exemple de fonctionnement (notice)

L'utilisation de l'application se fait entièrement via une interface console simple et intuitive. Dès le démarrage, l'utilisateur accède directement au Menu Principal, qui regroupe toutes les fonctionnalités du logiciel : création de tournoi, gestion des matchs, consultation des informations, etc.

1. Quel fichier faut-il lancer pour démarrer l'application ?

Pour exécuter le programme, il suffit de lancer la classe principale :

-tennis.app.TennisApp

C'est ce fichier qui :

- Configure l'encodage UTF-8 pour supporter les caractères accentués,
- Initialise le scanner,
- Affiche les menus,
- Orchestre toute la navigation dans l'application.

Dans NetBeans :

clic droit sur le projet → Set Configuration → Customize

Ensuite choisir tennis.app.TennisApp comme *Main Class*.

2. Création d'un tournoi

Après avoir lancé l'application via *TennisApp*, l'utilisateur peut choisir « *Créer un tournoi* ».

Il précise :

- La ville (Melbourne, Paris, Londres ou New York),
- L'année du tournoi.

L'application génère alors automatiquement :

- 128 joueurs hommes,
- 128 joueuses,
- 10 arbitres,
- Au moins 100 spectateurs.

Chaque participant reçoit des attributs variés et cohérents (taille, poids, main dominante...).

Un résumé clair du tournoi est ensuite affiché.

3. Création du premier tour

Depuis le menu de gestion du tournoi, l'utilisateur peut créer :

- Le premier tour Simple Homme,
- Le premier tour Simple Femme.

Le programme mélange les joueurs de façon aléatoire et crée les 64 matchs du premier tour.

4. Lancement d'un match

Pour chaque match du premier tour, l'utilisateur peut choisir le mode de déroulement :

- Mode manuel : le match avance étape par étape (set par set).
- Mode automatique silencieux : simulation instantanée sans détails.
- Mode automatique détaillé : simulation complète avec annonces set par set.

La logique du tennis est appliquée automatiquement :

- Alternance du service,
- Victoire d'un jeu,
- Victoire d'un set,
- Victoire du match,
- Annonces par l'arbitre,
- Mise à jour des statistiques.

5. Résultats et statistiques

À la fin du match, le programme affiche :

- Le vainqueur,
- Le score complet en sets,
- Les statistiques mises à jour.

L'utilisateur peut :

- Lancer un autre match,
- Consulter les informations du tournoi,
- Ou revenir au menu principal.

6. Robustesse et gestion des erreurs

L'application est sécurisée :

si l'utilisateur entre une valeur incorrecte, elle :

- N'interrompt jamais l'exécution,
- Affiche un message explicite,
- Redemande une entrée valide.

Grâce au bloc try/catch placé à chaque étape critique, l'expérience reste stable et fluide.

Difficultés rencontrer

La première difficulté est apparue dès la conception de l'architecture générale du projet. Même avec de bonnes connaissances théoriques en POO, il n'était pas évident de déterminer comment organiser l'ensemble des classes de manière cohérente. Le projet impliquait de nombreux acteurs (joueurs, arbitres, spectateurs), ainsi que plusieurs niveaux de logique (échange → jeu → set → match → tournoi). Il a fallu plusieurs essais et schémas avant d'obtenir une structure claire et extensible.

La gestion de l'encapsulation a également demandé beaucoup de réflexion. Le choix entre private, protected et public avait des conséquences directes sur la capacité des classes à collaborer entre elles. Dans certains cas, des attributs protégés bloquaient la remontée d'informations dans les niveaux supérieurs, ce qui nous a obligés à revoir les visibilités ou à ajouter des getters tout en respectant la règle d'encapsulation imposée par le professeur.

L'interdépendance entre les classes Match, SetTennis et Jeu a aussi généré plusieurs bugs. Comme chaque niveau dépend du précédent, une seule condition mal formulée pouvait bloquer tout l'enchaînement (par exemple, passer d'un jeu à un autre ou d'un set à la fin du match). Nous avons dû corriger et ajuster les transitions pour obtenir un déroulement fluide et conforme aux règles du tennis.

Une autre difficulté a concerné la mise en place de l'interface Supporter, imposée pour répondre à la règle « un joueur peut aussi être spectateur ». Il a fallu comprendre comment justifier l'utilisation d'une interface, éviter l'héritage multiple, et structurer les classes pour qu'un joueur puisse réellement réagir dans les gradins sans perturber le reste du modèle.

La gestion du tournoi complet a elle aussi apporté des défis : génération automatique de 256 joueurs, attribution cohérente des arbitres, mélange aléatoire, création dynamique des 64 matchs du premier tour, puis construction des tours suivants à partir des vainqueurs. Chaque étape devait être robuste et garantir la cohérence des catégories (match homme vs match femme).

L'interface console a demandé du travail supplémentaire. Il fallait créer des menus clairs, empêcher l'application de planter malgré des saisies incorrectes, et gérer des scénarios utilisateurs variés. L'emploi systématique de try/catch a permis de sécuriser l'exécution, mais a nécessité un effort constant pour anticiper les erreurs possibles.

Enfin, un dernier point technique a été la gestion de l'encodage UTF-8. Sans une configuration explicite de la sortie console, les caractères accentués apparaissaient mal, ce qui posait problème pour une application en français. La correction a nécessité de reconfigurer System.out avec un PrintStream en UTF-8.

Ces obstacles successifs nous ont permis de mieux comprendre l'importance de la conception avant l'implémentation, l'utilité de l'encapsulation, la puissance des interfaces, et surtout la nécessité de tester régulièrement chaque module avant de passer au suivant.

Conclusion

Ce projet nous a permis de mettre en pratique les différentes notions de programmation orientée objet vues pendant le cours. Même si l'organisation n'était pas évidente au début, travailler sur une application complète nous a aidés à mieux comprendre comment structurer plusieurs classes qui interagissent entre elles.

La simulation d'un match de tennis était un bon choix, car les règles sont simples à comprendre mais demandent tout de même une réflexion sérieuse pour les traduire correctement en code. On a dû réfléchir à la façon d'organiser l'architecture, aux relations entre les classes, aux visibilités, et à la manière de garder un code propre et cohérent.

Malgré les bugs et les ajustements nécessaires, on a réussi à obtenir un programme qui fonctionne et qui reste lisible. Ce travail nous a vraiment aidés à progresser, autant sur la technique que sur la méthode, et nous a donné une meilleure maîtrise de la POO.

Ce projet nous a aussi appris à mieux travailler en binôme, à communiquer sur nos choix et à garder une cohérence dans le développement. C'est une expérience qui nous servira clairement pour nos futurs travaux en Java ou dans d'autres langages orientés objet.