TP 1

# Travaux Pratiques : Introduction à la Programmation Orientée Objet en Java - Création d’un RPG

## Introduction

Dans le cadre de ce projet de travaux pratiques, nous allons explorer les bases de la programmation orientée objet (POO) en Java en développant un jeu de rôle (RPG) simple. L’objectif est d’introduire progressivement les concepts fondamentaux de la POO tout en créant un programme que vous pouvez vous approrier pour le rendre plus ou moins interessant.

## Objectifs

* Comprendre les principes de base de la programmation orientée objet.
* Apprendre à déclarer des variables en Java.
* Utiliser les entrées utilisateur pour interagir avec le programme.
* Afficher des données à l’écran.
* Progresser étape par étape dans la conception et l’implémentation d’un RPG en Java.

## Instructions

### **Étape 1 : Déclaration de variables et saisie du nom du personnage**

Dans cette première étape, vous allez créer un programme Java simple qui permet à l’utilisateur de saisir le nom de son personnage pour un jeu de rôle (RPG) et d’afficher ce nom à l’écran.

**Instructions :**

1. Démarrez votre environnement de développement Java.
2. Créez une nouvelle classe Java appelée “PersonnageRPG”.
3. Dans la classe “PersonnageRPG”, déclarez une variable de type String pour stocker le nom du personnage.
4. À l’intérieur de la méthode principale (main), utilisez un objet Scanner pour permettre à l’utilisateur de saisir le nom de son personnage.
5. Affichez ensuite le nom du personnage à l’écran.

**Exemple de code :**

import java.util.Scanner;  
   
public class PersonnageRPG {  
 public static void main(String[] args) {  
 // Déclaration de la variable pour stocker le nom du personnage  
 String nomPersonnage;  
   
 // Création d'un objet Scanner pour la saisie utilisateur  
 Scanner scanner = new Scanner(System.in);  
   
 // Demande à l'utilisateur de saisir le nom du personnage  
 System.out.print("Entrez le nom de votre personnage : ");  
 nomPersonnage = scanner.nextLine();  
   
 // Affiche le nom du personnage à l'écran  
 System.out.println("Le nom de votre personnage est : " + nomPersonnage);  
   
 // Ferme le scanner  
 scanner.close();  
 }  
}

N’hésitez pas à vous familiariser avec l’auto-completion et appréhender votre IDE Le Java est un langage assez verbeux, de plus nous allons *à outrance* faires de choses répétitives pour aborder les concepts, si vous décrouvrez comment tirer profit de l’outil au maximum et que vous comprennez ce qu’il se passe, vous pourriez gagner un temps précieux

**Conseils** - Familiarisez vous avec les points d’arrêt si vous les découvrez [Run and Debug Java in Visual Studio Code](https://code.visualstudio.com/docs/java/java-debugging)

1. Faites en sorte que l’utilisateur soit invité à saisir un nom à son personnage tant que la longueur de la variable nom est inférieure à 3

### **Étape 2 : Création de nouvelles variables pour les caractéristiques du personnage**

Dans cette étape, vous allez créer de nouvelles variables pour représenter les caractéristiques principales de votre personnage dans un RPG, telles que la force, l’agilité et l’intelligence. Vous utiliserez un switch case pour choisir quelles caractéristiques augmenter à l’aide des points de compétence.

**Instructions :**

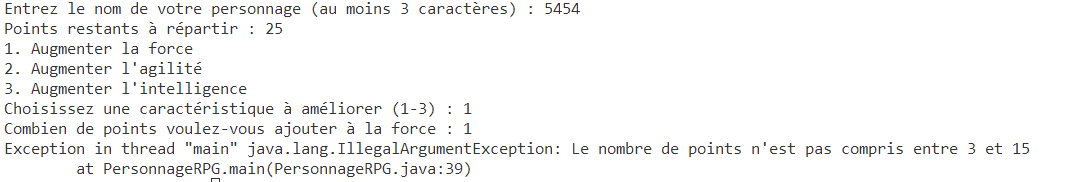
1. Ajoutez trois nouvelles variables de type int dans votre classe “PersonnageRPG” pour représenter la force, l’agilité et l’intelligence du personnage.
2. Initialisez ces variables à zéro au début du programme.
3. Demandez à l’utilisateur de répartir 25 points de compétence entre ces caractéristiques en utilisant un switch case pour choisir quelle caractéristique augmenter.
4. Assurez-vous que le total des points attribués ne dépasse pas 25.
5. Si le total des points attribués est négatif ou dépasse 25, lancez une exception pour indiquer une erreur.

**Conseils :**

* Assurez-vous de comprendre comment fonctionne le switch case pour gérer différents choix de l’utilisateur.
* Utilisez des conditions (if/else) pour vérifier si les points attribués sont valides et pour lancer des exceptions si nécessaire.
* Testez votre programme avec différentes répartitions de points pour vous assurer qu’il fonctionne correctement.

1. Équilibrez votre jeu, on veut éviter que le personnage soit une brute avec 25 de force et 0 en agilité et intelligence On veut un minimum de 3 points dans chaque compétence et pas plus de 15 points par compétence Vous avez deux solutions, soit vous initialisez les variables à 3 et vous considérez qu’il n’y a plus que 16 points à attribuer, soit vous faites en sorte que l’utilisateur choisisse un chiffre compris entre 3 et 15 en lui affichant une message en cas de saisie incorrecte On préfèrera la deuxième option car elle vous apprends à lever une exception : throw new IllegalArgumentException("L'agilité doit être entre 3 et 15.");
2. Votre code ressemble à ça au moment d’attribuer les points :

...  
 // Répartition des points de compétence  
 int pointsRestants = 25;  
 while (pointsRestants > 0) {  
 System.out.println("Points restants à répartir : " + pointsRestants);  
 System.out.println("1. Augmenter la force");  
 System.out.println("2. Augmenter l'agilité");  
 System.out.println("3. Augmenter l'intelligence");  
 System.out.print("Choisissez une caractéristique à améliorer (1-3) : ");  
 int choix = scanner.nextInt();  
   
 switch (choix) {  
 case 1:  
 System.out.print("Combien de points voulez-vous ajouter à la force : ");  
 int pointsForceAAttribuer = scanner.nextInt();  
 if(pointsForceAAttribuer > pointsRestants){  
 throw new IllegalArgumentException("Le nombre de points attribués est supérieur au nombre de points restants");  
 }  
 int nouveauSoldePointForce = force + pointsForceAAttribuer;  
 if (nouveauSoldePointForce < 3 || nouveauSoldePointForce > 15) {  
 throw new IllegalArgumentException("Le nombre de points n'est pas compris entre 3 et 15");  
 }  
 force = nouveauSoldePointForce;  
 pointsRestants -= pointsForceAAttribuer;  
 break;  
   
 ...  
 }  
 ...  
 }

Lancez le programme et essayez d’attribuer 20 points d’agilité vous aurez l’erreur suivante : 

1. Adaptez votre code pour introduire try{} catch{} Exemple :

while(pointsRestants > 0){  
 try{  
 System.out.println("Points restants à répartir : " + pointsRestants);  
 System.out.println("1. Augmenter la force");  
 System.out.println("2. Augmenter l'agilité");  
 System.out.println("3. Augmenter l'intelligence");  
 System.out.print("Choisissez une caractéristique à améliorer (1-3) : ");  
 int choix = scanner.nextInt();  
   
 switch (choix) {  
 case 1:  
 // Reste du code  
 throw new IllegalArgumentException("Le nombre de points n'est pas compris entre 3 et 15");  
 break;  
 }  
 } catch (Exception e) {  
 System.err.println("\033[0;31m" +e.getMessage()+"\033[0m");  
 }  
}

Ne pretez pas trop attention aux caractères bizarres dans le catch Cela permet de changer la couleur de la console pour afficher l’erreur en rouge

Ainsi lorsque vous ferez une mauvaise saisie, le message d’erreur s’affichera et vous devrez recommencer jusqu’à avoir correctement attribué tous vos points

**Vous remarquez que le code commence à devenir répétitif pour 4 pauvres variables On va alors essayer de transformer ce code en programme orienté objet**

### **Étape 3 : Création de la classe Personnage**

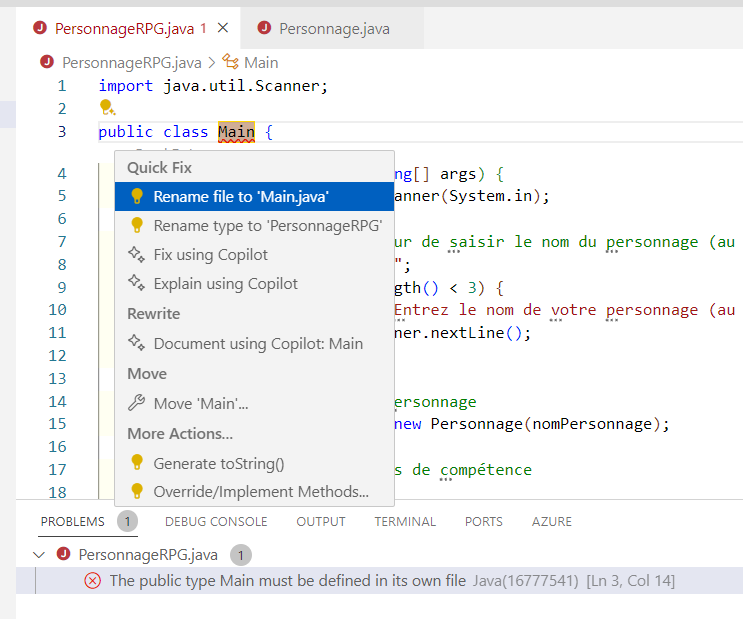
1. Créez une nouvelle classe Personnage dans le fichier Personnage.java qui contiendra les attributs et les méthodes pour représenter un personnage dans le RPG.

public class Personnage {  
 private String nom;  
 private int force;  
 private int agilite;  
 private int intelligence;  
   
 // Constructeur  
 public Personnage(String nom) {  
 this.nom = nom;  
 this.force = 0;  
 this.agilite = 0;  
 this.intelligence = 0;  
 }  
   
 // Getters  
 public String getNom() {  
 return nom;  
 }  
   
 public int getForce() {  
 return force;  
 }  
   
 public int getAgilite() {  
 return agilite;  
 }  
   
 public int getIntelligence() {  
 return intelligence;  
 }  
   
 // Setters avec vérifications  
   
 public void augmenterIntelligence(int intelligenceAAjouter) {  
 int nouvelleIntelligence = intelligenceAAjouter + intelligence;  
 if (nouvelleIntelligence < 3 || nouvelleIntelligence > 15) {  
 throw new IllegalArgumentException("L'intelligence doit être entre 3 et 15.");  
 }  
 this.intelligence = nouvelleIntelligence;  
 }  
 // ... Faites pareil pour Agilité et Force  
}

#### **Étape 3.1 : Utilisation de la classe Personnage dans le programme principal**

Dans votre programme principal, utilisez la classe Personnage pour créer un personnage et utilisez ses méthodes pour accéder et modifier ses caractéristiques.

import java.util.Scanner;  
   
public class Main {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.in);  
   
 // Demande à l'utilisateur de saisir le nom du personnage (au moins 3 caractères)  
 String nomPersonnage = "";  
 while (nomPersonnage.length() < 3) {  
 System.out.print("Entrez le nom de votre personnage (au moins 3 caractères) : ");  
 nomPersonnage = scanner.nextLine();  
 }  
   
 // Création d'un objet Personnage  
 Personnage personnage = new Personnage(nomPersonnage);  
   
 // Répartition des points de compétence  
 int pointsRestants = 25;  
 while (pointsRestants > 0) {  
 try{  
 System.out.println("Points restants à répartir : " + pointsRestants);  
 System.out.println("1. Augmenter la force");  
 System.out.println("2. Augmenter l'agilité");  
 System.out.println("3. Augmenter l'intelligence");  
 System.out.print("Choisissez une caractéristique à améliorer (1-3) : ");  
 int choix = scanner.nextInt();  
   
 switch (choix) {  
 case 1:  
 System.out.print("Combien de points voulez-vous ajouter à la force (entre 3 et 15) : ");  
 int pointsForceAAttribuer = scanner.nextInt();  
 personnage.augmenterForce(pointsForceAAttribuer);  
 pointsRestants -= pointsForceAAttribuer;  
 break;  
 case 2:  
 System.out.print("Combien de points voulez-vous ajouter à l'agilité (entre 3 et 15) : ");  
 int pointsAgiliteAAttribuer = scanner.nextInt();  
 personnage.augmenterAgilite(pointsAgiliteAAttribuer);  
 pointsRestants -= pointsAgiliteAAttribuer;  
 break;  
 case 3:  
 System.out.print("Combien de points voulez-vous ajouter à l'intelligence (entre 3 et 15) : ");  
 int pointsIntelligenceAAttribuer = scanner.nextInt();  
 personnage.augmenterIntelligence(pointsIntelligenceAAttribuer);  
 pointsRestants -= pointsIntelligenceAAttribuer;  
 break;  
 default:  
 System.out.println("Choix invalide, veuillez choisir une option valide !");  
 }  
 } catch (Exception e) {  
 System.err.println("\033[0;31m" +e.getMessage()+"\033[0m");  
 }  
 }  
   
 // Affichage des caractéristiques du personnage  
 System.out.println("Nom du personnage : " + personnage.getNom());  
 System.out.println("Force : " + personnage.getForce());  
 System.out.println("Agilité : " + personnage.getAgilite());  
 System.out.println("Intelligence : " + personnage.getIntelligence());  
   
 scanner.close();  
 }  
}

Vous remarquez une erreur en compilant votre code  En Java le nom du fichier doit respecter le nom de la classe qu’il contient et il ne doit y avoir qu’une seule classe par fichier Vous remarquez que la petite ampoule 💡 vous propose de corriger ça à votre place

**Observations :**

* La classe Personnage encapsule maintenant les données et les opérations liées à un personnage, améliorant ainsi la lisibilité et la maintenance du code.
* Les vérifications des valeurs des caractéristiques sont désormais effectuées dans les setters de la classe Personnage, ce qui renforce le principe d’encapsulation et garantit l’intégrité des données.

### **Étape 4 : Ajout de compétences au personnage**

Dans cette étape, vous allez étendre les caractéristiques du personnage en ajoutant des compétences telles que la perception, l’endurance, le charisme et la chance. Au lieu de créer des getters et des setters pour chaque compétence, vous allez stocker ces informations dans un dictionnaire pour simplifier votre code. > En effet vous avez vu qu’il était assez pénible d’écrire 3 fois la même chose, si vous le faites 7 fois il y a de grandes chances que vous finissiez par introduire une erreur

**Instructions :**

1. Ajoutez les compétences suivantes : Perception, Endurance, Charisme, et Chance, en plus des caractéristiques déjà existantes. Pour cela créez un fichier Competence.java et ajoutez le code suivant :

enum Competence {  
 FORCE, PERCEPTION, ENDURANCE, CHARISME, INTELLIGENCE, AGILITE, CHANCE;  
}

1. Nous allons introduire une nouvelle variable que vous ne connaissez pas : EnumMap EnumMap est un dictionnaire dont les clés font partie d’une énum En gros vous pouvez l’imaginer comme un tableau, la première colonne contient les noms de compétences et la deuxième ce que vous voulez Toutefois vous ne voulez pas qu’on puisse ecrire n’importe quoi dans ce tableau Par exemple enumMap.put(Competences.FORCE, 1); c’est OK mais on ne pourra pas écrire enumMap.put("toto",1)
2. Dans la classe Personnage, créez une méthode augmenterCompetence pour augmenter le niveau d’une compétence spécifique en vérifiant que la nouvelle valeur est comprise entre 3 et 15.
3. Utilisez des exceptions pour signaler une erreur si l’utilisateur entre une valeur incorrecte lors de l’augmentation d’une compétence.
4. Vous pouvez vous débarraser de augmenterForce, augmenterAgilite et augmenterIntelligence

A ce stade, votre classe Personnage ressemble à ceci : (Il n’y a pas les getter/setter sur nom mais vous devriez les avoir implémentés)

import java.util.EnumMap;  
import java.util.Map;  
   
public class Personnage {  
 private String nom;  
 private Map<Competence, Integer> competences;  
   
 // Constructeur  
 public Personnage(String nom) {  
 this.nom = nom;  
 this.competences = new EnumMap<>(Competence.class);  
 // Initialisation des compétences avec des valeurs par défaut  
 initialiserCompetences();  
 }  
   
 // Méthode privée pour initialiser les compétences avec des valeurs par défaut  
 private void initialiserCompetences() {  
 for (Competence competence : Competence.values()) {  
 competences.put(competence, 0);  
 }  
 }  
   
 // Méthode pour augmenter le niveau d'une compétence  
 public void augmenterCompetence(Competence competence, int points) {  
 /// Effectuez les vérifications nécessaires  
   
 competences.put(competence, nouveauNiveau);  
 }  
   
 public int getNiveauCompetence(Competence competence) {  
 return competences.get(competence);  
 }  
   
 // Méthode pour afficher les compétences du personnage  
 public void afficherCompetences() {  
 System.out.println("Compétences de " + nom + " :");  
 for (Competence competence : Competence.values()) {  
 System.out.println(String.format("%-10s \t niveau : %d",competence, getNiveauCompetence(competence)));  
 }  
 }  
   
}

Dans votre code principal :

// Répartition des points de compétence  
 int pointsRestants = 25;  
 while (pointsRestants > 0) {  
 System.out.println("Points restants à répartir : " + pointsRestants);  
 System.out.println("Choisissez une compétence à améliorer : ");  
 for (int index = 0; index < Competence.values().length; index++) {  
 System.out.println(String.format("%d. %s", index + 1, Competence.values()[index]));  
 }  
   
 Integer choixCompetence = scanner.nextInt();  
 if (choixCompetence < 1 || choixCompetence > Competence.values().length) {  
 throw new IllegalArgumentException("Veuillez choisir un nombre entre 1 et " + Competence.values().length);  
 }  
   
 Competence competenceAAugmenter = Competence.values()[choixCompetence - 1];  
 System.out.print("Combien de points voulez-vous ajouter à " + competenceAAugmenter + " (entre 3 et 15) : ");  
 int pointsCompetence = scanner.nextInt();  
   
 try {  
 personnage.augmenterCompetence(competenceAAugmenter, pointsCompetence);  
 pointsRestants -= pointsCompetence;  
 } catch (IllegalArgumentException e) {  
 System.err.println("\033[0;31m" +e.getMessage()+"\033[0m");  
 }  
 }

**Conseils :**

* Utilisez un dictionnaire pour stocker les compétences afin de simplifier la gestion des données et de rendre votre code plus flexible.
* Utilisez des exceptions pour gérer les erreurs lors de l’augmentation des compétences, en veillant à ce que les valeurs soient comprises entre 3 et 15.

**Illustration de l’avantage de l’encapsulation :**

L’encapsulation permet de regrouper les données (les compétences du personnage) et les méthodes qui agissent sur ces données (augmenter le niveau d’une compétence) dans une seule entité, la classe Personnage. Cela rend le code plus modulaire, plus facile à comprendre et à maintenir. De plus, en encapsulant les données, nous pouvons garantir leur intégrité et empêcher l’accès direct et non autorisé à ces données de l’extérieur de la classe. Dans notre exemple, si votre personnage gagne un niveau vous n’aurez qu’à appeler la méthode augmenterCompetence et vous n’aurez pas à refaire toutes les vérifications, aussi, il est impossible sans modifier la classe Personnage de changer la limite de 15 points de compétences. C’est un aspect fondamental en termes de sécurité. Imaginez que votre seule mission est de créer une librairie avec la classe Personnage et que d’autres personnes peuvent créer un jeu, vous êtes assurés qu’il ne pourront pas la modifier et que le niveau maximum sera toujours de 15.

A ce stade là, déposez votre code sur e-campus