### Projet LU2IN002 - 2020-2021

|  |
| --- |
| Numéro du groupe de TD/TME : 8 |

|  |  |  |
| --- | --- | --- |
| *Nom :Zeng* | *Nom :Li* | *Nom :Martin* |
| *Prénom :Fanxiang* | *Prénom :Junji* | *Prénom :Amaury* |
| *N° étudiant :28600693* | *N° étudiant :28610187* | *N° étudiant :28610127* |

|  |
| --- |
| *Thème choisi (en 2 lignes max.)* |
| Simulation du fonctionnement d’un club de foot |

|  |
| --- |
| *Description des classes et de leur rôle dans le programme (2 lignes max par classe)* |
| Class Personne : classe mère, qui possède les attributs de base salaire et nom  Class Joueur : classe fille de la classe personne, qui a comme attribut la capacité, âge et le contrat du joueur (avec des constants définies)  Class Remplaçant : classe fille de la classe joueur, qui lui possède des coefficients différent de celui des joueurs  Class Manager : classe fille de la classe personne, qui lui peut avoir un effet sur les bénéfices du club en fonction du salaire qu’on lui donne  Class Coach : classe fille de la classe personne, qui lui peut avoir un effet sur l’augmentation du capacité des joueurs  Class Match : un singleton qui a un constructeur private, qui simule un match en fonction du capacité moyenne des joueurs de notre club contre une équipe adverse de capacité généré aléatoirement  Class Déclinable : une interface implémenter par les joueurs , qui a une fonction qui modifie l’âge du joueur et diminue son capacité si le joueur a dépassé un âge seuil  Class Outil : une classe qui ne possède que des méthodes static pour simplifier la programmation et de rendre le code plus lisible  Class DepassementPlafondSalaireException : classe de l’exception qui throw une exception lorsque la somme des salaires des joueurs dépasse un certain plafond  Class Club : class composé des joueurs, des remplaçants, un coach , un manager, et un match, et l’unité de simulation est année, avec un bénéfice faite à chaque année. |

|  |
| --- |
| *Schéma UML des classes vision fournisseur (dessin “à la main” scanné ou photo acceptés)* |
| *PS : agrandissez avec ctrl + souris ou bien en bas à droite sur word pour voir plus clairement* |

|  |  |
| --- | --- |
| *Checklist des contraintes prises en compte:* | *Nom(s) des classe(s) correspondante(s)* |
| Classe contenant un tableau ou une ArrayList | public class Club |
| Classe avec membres et méthodes statiques | public class Outil |
| Classe abstraite et méthode abstraite | public abstract class Personne |
| Interface | public interface Declinable |
| Classe avec un constructeur par copie ou clone() | public class Joueur extends Personne implements Declinable  public class Remplacant extends Joueur |
| Définition de classe étendant Exception | public class DepassementPlafondSalaireException extends Exception |
| Gestion des exceptions | public class Club  public class Main |
| Utilisation du pattern singleton | public final class Match |

|  |
| --- |
| *Présentation de votre projet (max. 2 pages) : texte libre expliquant en quoi consiste votre projet.* |
| Notre projet consiste à simuler le fonctionnement d’un club de foot sur plusieurs années. En fin de chaque année nous allons voir les différentes caractéristiques de notre club. Tout d’abord notre club est composé de personnes, ces personnes peuvent être des joueurs, des remplaçants, un manager et un coach. Chaque personne a un nom et un salaire.  Les joueurs et les remplaçants sont au cœur de notre simulation, ces derniers possèdent une capacité et leur salaire dépend de leur capacité : plus leurs capacités sont élevées, plus ils méritent un salaire haut. Ensuite, leur salaire est renouvelé en fonction de leur contrat (par défaut 4 ans), lorsque leur contrat est fini, nous redéfinissons leur salaire en fonction de leur capacité. Les remplaçants ont des capacités plus basses que les joueurs principaux, et leur ratio capacité /salaire est également plus bas.  Or, notre club, pour toujours rester dans un statut de bénéfice, a décidé de mettre en place un plafond de salaire. En effet, si la somme des salaires des Personnes dépasse ce plafond, notre club va être en perte, ce n’est pas ce qu’on veut, donc dans ce cas nous levons une exception, et enlevons le joueur qui demande le plus grand salaire. Pour que l’équipe de joueurs reste toujours une équipe de 11 personnes, nous remontons le meilleur remplaçant (en termes de capacité), le faisons devenir joueur principal, et nous recrutons un nouveau remplaçant (généré aléatoirement).  Ensuite, chaque année, notre club participe à des Match (par défaut 80 par an). À chaque Match gagné le club fait un certain nombre de bénéfices et les capacités des joueurs vont augmenter en fonction du nombre de matchs gagnés et perdus à la fin de l’année (une fois les matchs passés). De plus, le coach va avoir une influence sur l’augmentation des capacités des joueurs, et le manager une influence sur le bénéfice pour le club pour chaque Match.  La capacité des joueurs et des remplaçants, lorsqu'ils dépassent un certain seuil d'âge, va commencer à diminuer, et quand l’âge dépasse un autre seuil, ce joueur va être renvoyé.  A la fin de chaque année, les données vont être affichées, et le nombre d’années de simulation est défini par une constante année max.  *PS : pour mieux voir le fonctionnement du code, vous pouvez modifier les données statiques qui servent de coefficient ou de bornes.* |

|  |
| --- |
| *Copier / coller vos classes et interfaces à partir d'ici :*  import java.util.ArrayList;  /\*\*  \* un Club est une unite qui regroupe tous les elements  \* il gere le personnel, prepare au match et ses elements changent selon le resultat  \*/  public class Club {  private Manager manager;  private Coach coach;  private ArrayList<Joueur> joueurs;  private ArrayList<Remplacant> remplacants;  private Match match;  private static int annee = 0;  // des valeurs par defaut  private static int nombreJoueurMax = 11;  private static int nombreRemplacantMax = 7;  private static double plafondSalaire = 1000;  private static int anneeMax = 15; // le nombre max d'annee pour evoluer  private static double benefice;  /\*\*  \* constructeur avec des param specifiees  \* @param manager  \* @param coach  \* @param joueurs  \* @param remplacants  \* @param match  \*/  public Club(Manager manager, Coach coach, ArrayList<Joueur> joueurs, ArrayList<Remplacant> remplacants,  Match match) {  this.manager = manager;  this.coach = coach;  this.joueurs = joueurs;  this.remplacants = remplacants;  this.match = match;  }  /\*\*  \* constructeur par defaut  \*/  public Club(){  this.manager = new Manager();  this.coach = new Coach();  this.joueurs = new ArrayList<Joueur>();  this.remplacants = new ArrayList<Remplacant>();  this.match = Match.getMatch(0);  }  /\*\*  \* renvoyer les Joueurs et Remplacants de plus de 35 ans en les supprimant  \*/  public void renvoyer() {  for(int i = 0; i < joueurs.size(); i++){  Joueur j = joueurs.get(i);  if(j.getAge() >= 38){  joueurs.remove(j);  System.out.println(j + " est renvoye");  i -= 1;  }  }  for(int i = 0; i < remplacants.size(); i++){  Remplacant r = remplacants.get(i);  if(r.getAge() >= 34){  remplacants.remove(r);  System.out.println(r + " est renvoye");  i -= 1;  }  }  }  /\*\*\*  \* avant la promotion, verifie si le salaire total est plus grand que le plafondSalaire  \* Si oui, throw new DepassementBudgetException  \* sinon, continue  \* @return  \* @throws DepassementPlafondSalaireException  \*/  public boolean budget() throws DepassementPlafondSalaireException{  // a finir  double sumsalaire = manager.salaire + coach.salaire;  for(Joueur j : joueurs){  sumsalaire += j.salaire;  }  for(Remplacant r : remplacants)  {  sumsalaire += r.salaire;  }  if(remplacants.size() == 0){  recrutement();  }  sumsalaire += new Joueur(Outil.remplacantLePlusCapable(remplacants)).getSalaire();  if(sumsalaire <= plafondSalaire){  return true;  }else{  throw new DepassementPlafondSalaireException("Le salaire depasse le plafond.");  }  }  /\*\*  \* enlever Le joueur le plus capable de la ArrayList<Joueur>  \* pour diminuer le salaire total  \*/  public void enleverJoueurLePlusCapable(){  Joueur capmax = Outil.joueurLePlusCapable(joueurs);  System.out.println("le joueur : " + capmax.nom+ " de capacite : " + capmax.capacite + "est enlevee");  joueurs.remove(capmax);  }  /\*\*  \* renouveler le contrat pour toute l'equipe  \*/  public void renouvelerContrat() {  for (Joueur joueur : joueurs) {  if(joueur.getContrat() == 0)  joueur.setContrat(Joueur.getContratDefaut());  }  for (Remplacant remplacant : remplacants) {  if(remplacant.getContrat() == 0)  remplacant.setContrat(Remplacant.getContratDefaut());  }  }    /\*\*  \* trouve un joueur dans la liste de remplacants en le nommant joueur principal  \* on peut créer des listes temporaires  \*/  public void promotion(){  if(remplacants.size() == 0){  recrutement();  }  Remplacant capmax = remplacants.get(0);  for(Remplacant r : remplacants){  if (capmax.capacite < r.capacite) {  capmax = r;  }  }  joueurs.add(new Joueur(capmax));  System.out.println(capmax + " obtient une promotion.");  remplacants.remove(capmax);  }  /\*\*  \* faire decliner tous les joueurs et remplacants  \* incrementation d'age et eventuellement diminer la capacite  \*/  public void decliner() {  for (Joueur joueur : joueurs) {  joueur.decliner();  }  for (Remplacant remplacant : remplacants) {  remplacant.decliner();  }  }  /\*\*\*  \*completer la liste de remplacants  \*/  public void recrutement(){  while(remplacants.size() < nombreRemplacantMax){  remplacants.add(new Remplacant()); //ajouter le joueur recruté  }  }  /\*\*  \* renvoie les joueur : age > 30  \* promotion  \* renouveler les contrats  \* recrutement des remplacants  \*/  public void mettreAJour(){  decliner();  renvoyer();  renouvelerContrat();  while(joueurs.size() < nombreJoueurMax){  try {  if (budget()) {  promotion();  }  } catch (DepassementPlafondSalaireException e) {  System.out.println(e.getMessage());  enleverJoueurLePlusCapable();  promotion();  }  }  recrutement();  }  /\*\*  \* simulation d'un concours de football  \* comparer les données du match et la somme de capacite de joueur principaux  \* et changer la capacite des joueurs principaux selon l'algorithme ecrit dans Outil.java  \*/  public void participerAuMatch(){  this.match = Match.getMatch(getCapaciteNotre());  match.resultMatch();  int nbVictoire = match.getNbVictoire();  int nbDefaite = match.getNbDefaite();  benefice = (Match.beneficeMatch \* Manager.addBenefice)\*nbVictoire;  coach.setCoachCapaciteMatch(nbVictoire,nbDefaite);  for (Joueur joueur : joueurs) {  int capaciteOriginal = joueur.getCapacite();  joueur.setCapacite(coach.addTeamCapacite+capaciteOriginal + Outil.capaciteDifference(nbVictoire, nbDefaite));  }  match.finirMatch();  annee += 1;  }  /\*\*  \* afficher des information pour l'utilisateur  \*/  public void sePresenter() {  System.out.println(this.toString());  }  /\*\*  \* afficher des information pour l'utilisateur  \* version2  \*/  public void sePresenter2() {  System.out.println("\*\*\*\*\*\*\*\*\*\*");  System.out.println("\*\*\*\*\*\*\*\*\*\*");  System.out.println("Club de football en fin d'annee " + annee + ".");  manager.sePresenter();  coach.sePresenter();  for (Joueur joueur : joueurs) {  joueur.sePresenter();  }  for (Remplacant remplacant : remplacants) {  remplacant.sePresenter();  }  System.out.println("capacite moyen=" + getCapaciteNotre() + "\n");  System.out.println("benefice du club=" + benefice + "\n");  System.out.println("\*\*\*\*\*\*\*\*\*\*");  System.out.println("\*\*\*\*\*\*\*\*\*\*");  }  /\*\*\*  \* Obtenir la capacite de tous les joueurs principaux  \* @return capaciteNotre  \*/  public int getCapaciteNotre(){  if(joueurs.size() == 0){  return 0;  }  int capaciteNotre = 0;  for (Joueur joueur : joueurs) {  capaciteNotre += joueur.getCapacite();  }  return capaciteNotre/joueurs.size();  }  @Override  public String toString() {  String str = "\*\*\*\*\*\*\*\*\*\*\n";  str += "\*\*\*\*\*\*\*\*\*\*\n";  str += "Club de football\n";  str += "annee " + annee + "\n";  str += manager.toString() + "\n";  str += coach.toString() + "\n";  for (Joueur joueur : joueurs) {  str += joueur.toString();  }  for (Remplacant remplacant : remplacants) {  str += remplacant.toString();  }  str += "capacite moyen " + getCapaciteNotre() + "\n";  str += "\*\*\*\*\*\*\*\*\*\*\n";  str += "\*\*\*\*\*\*\*\*\*\*\n";  return str;  }  /\*\*  \* completer les attributs manquants du Club  \*/  public void init() {  if(manager == null){  manager = new Manager();  }  if(coach == null){  coach = new Coach();  }  recrutement();  while(joueurs.size() < nombreJoueurMax){  promotion();  }  recrutement();  if(match == null){  match = Match.getMatch(getCapaciteNotre());  }  }  /\*\*  \* faire tourner le Club  \*/  public void run() {  while (annee <= anneeMax) {  mettreAJour();  participerAuMatch();  sePresenter2();  }  }  /\*\*  \* faire tourner le Club avec un nombre precise d'annees en parametres  \*/  public void run(int nbAnnees) {  while (annee <= nbAnnees) {  mettreAJour();  participerAuMatch();  sePresenter2();  }  }  public static void resetAnnee(){  annee = 0;  }  }  public class Coach extends Personne{  //attribut  // des valeurs par défaut  public static final double salaireDefaut = 1;  // valeurs pour une instance  private double capacite;  protected int addTeamCapacite = (int)(capacite / 40);  //methode  /\*\*\*  \* constructeur a partir de salaire et la capacite  \* @param salaire  \* @param capacite  \*/  public Coach(double salaire, double capacite) // Constructeur  {  super(salaire);  this.capacite = capacite;  }  /\*\*  \* constructeur a partir de capacite, salaire = salaireDefaut  \* @param capacite  \*/  public Coach(double capacite) // Constructeur  {  super(salaireDefaut);  this.capacite = capacite;  }  /\*\*  \* constructeur par défaut  \*/  public Coach()  {  this(Outil.aleatoireIntEntre(50,60));  }  /\*\*  \* afficher des information pour l'utilisateur  \*/  public void sePresenter()  {  System.out.println(this.toString());  }  public void setCoachCapaciteMatch(int nbvictoire, int nbdefaite){  if(nbvictoire > nbdefaite){  this.capacite += (nbvictoire - nbdefaite)/10;  }  else{  this.capacite-= (nbdefaite - nbvictoire) / 10;  }  }  //getters and setters  @Override  public String toString() {  return "Coach [nom=" + nom + ", capacite=" + capacite + ", salaire=" + salaire + "]\n";  }  public static double getSalairedefaut() {  return salaireDefaut;  }  public double getCapacite() {  return capacite;  }  public void setCapacite(double capacite) {  this.capacite = capacite;  }  }  /\*\*  \* lors de vieillissement, les capacites d'un joueur ou d'un coach declinent  \* ce qui lui rend moins capable et à la fin il doit être remplacé par un nouveau  \*/  public interface Declinable {  /\*\*  \* diminuer la capacite  \*/  public void decliner();  }  /\*\*  \* Exception declenche quand le salaire va depasser le plafond  \*/  public class DepassementPlafondSalaireException extends Exception{  /\*\*  \*  \*/  private static final long serialVersionUID = 1L;  public DepassementPlafondSalaireException(String message) {  super(message);  }  }  /\*\*  \* Un joueur est ce qui participe au match  \*/  public class Joueur extends Personne implements Declinable {  // attribut  protected static int seuilAge = 32; // si un joueur depasse cet age, il se decline  protected static int capaciteDeclin = 3; // la capacite qu'un joueur perd si son age depasse le seuil  protected static int contratDefaut = 4; // la duree de contrat  protected static int ageDefaut = 19;  protected final static double COEFFSALAIRECAPACITE = 1.5;// si un joueur est cree a partir d'un remplacant, salaire = capacite\*COEFFSALAIRECAPACITE    protected int capacite;  protected int age;  protected int contrat;  private final static int MIN = 60;  private final static int MAX = 95;  // méthode  /\*\*  \* constructeur a partir de capacite et age  \*/  public Joueur(int capacite, int age) {  super(capacite \*COEFFSALAIRECAPACITE );  this.capacite = capacite;  this.age = age;  this.contrat = contratDefaut;  }  /\*\*  \* constructeur par défaut utilisant des donnée aleatoire  \*/  public Joueur()  {  this(Outil.aleatoireIntEntre(MIN, MAX), ageDefaut);  this.contrat = contratDefaut;  }  /\*\*  \* constructeur a partir d'un remplacant  \*/  public Joueur(Remplacant r) {  super(r.salaire\*COEFFSALAIRECAPACITE, r.nom);  this.capacite = r.capacite;  this.age = r.age;  this.contrat = contratDefaut;  }  /\*\*  \* construction par copie  \*/  public Joueur(Joueur j) {  super(j.salaire);  this.capacite = j.capacite;  this.age = j.age;  this.contrat = contratDefaut;  }  /\*\*  \* augmenter l'age et diminuer capacite  \*/  public void decliner() {  age += 1;  contrat -= 1;  if (age >= seuilAge) {  capacite -= capaciteDeclin;  }  }  /\*\*  \* augmenter la capacite grace au coach  \*/  public void augmenterCapacite(Coach c) {  this.capacite += c.getCapacite();  }  /\*\*  \* afficher des information pour l'utilisateur  \*/  @Override  public void sePresenter() {  System.out.println(this.toString());  }  @Override  public String toString() {  return "Joueur [nom=" + nom + ", age=" + age + ", capacite=" + capacite + ", contrat=" + contrat + ", salaire=" + salaire + "]\n";  }  // getters and setters  public int getCapacite() {  return capacite;  }  public void setCapacite(int capacite) {  this.capacite = capacite;  if(capacite >= 100){  capacite = 100;  }  }  public int getAge() {  return age;  }  public void setAge(int age) {  this.age = age;  }  public int getContrat() {  return contrat;  }  public void setContrat(int contrat) {  this.contrat = contrat;  }  public static int getSeuilAge() {  return seuilAge;  }  public static void setSeuilAge(int seuilAge) {  Joueur.seuilAge = seuilAge;  }  public static int getContratDefaut() {  return contratDefaut;  }  public static void setContratDefaut(int contratDefaut) {  Joueur.contratDefaut = contratDefaut;  }  public static int getAgeDefaut() {  return ageDefaut;  }  public static void setAgeDefaut(int ageDefaut) {  Joueur.ageDefaut = ageDefaut;  }  public boolean equals(Joueur joueurCmp){  if(this.capacite != joueurCmp.capacite || this.age != joueurCmp.age || this.contrat != joueurCmp.contrat)  return false;  return true;  }  public Joueur clone(){  return new Joueur(this);  }  }  class Main {  public static void main(String[] args) {  Club clubSimu = new Club();  clubSimu.init();  try{  clubSimu.run(Integer.parseInt(args[0]));  }  catch(Exception e){  clubSimu.run();  }  System.out.println("Pour gerer le nombre d'annee a tourne, tapez comme 'java Main 16' pour une boucle de 16 annee.");  }  }  /\*\*  \* Logiquement un manager assure le bon fonctionnement financier d'un club  \* Ici on le manager represente tout le groupe logistique  \*/  public class Manager extends Personne{  // attribut  public static final double salaireDefaut = 1; // plus grand que salaire defaut d'une Personne  protected static double addBenefice = salaireDefaut\*1. ;  //méthode  /\*\*\*  \* constructeur avec salaire specife  \* @param salaire  \*/  public Manager(double salaire) {  super(salaire);  }  /\*\*  \* constructeur par defaut  \*/  public Manager() {  super(salaireDefaut);  }  /\*\*  \* constructeur par copie  \*/  public Manager(Manager m) {  super(m.salaire);  }  /\*\*  \* afficher des information pour l'utilisateur  \*/  public void sePresenter()  {  System.out.println(this.toString());  }  @Override  public String toString() {  return "Manager [nom=" + nom + ", salaire=" + salaire + "]\n";  }  }  /\*\*  \* le singleton cette classe contient des donnees du match  \*/  public final class Match {  private static Match match;  public final static double beneficeMatch = 10.;  private final static int MIN = 55;  private final static int MAX = 95;  private final static int NOMBREDEMATCH = 80;  private int capaciteNotre;  private int capaciteAdversaire;  private int nbVictoire;  private int nbDefaite;  /\*\*  \* creer un match a partir de la capacite de toute l'equipe  \*  \* @param capaciteNotre  \*/  private Match(int capaciteNotre) {  this.capaciteNotre = capaciteNotre;  }  /\*\*  \* calculer le nombre de vitoires et de defaites tourner NOMBREDEMATCH fois  \* capaciteAdversaire est tiree entre MIN et MAX  \*/  public void resultMatch() {  for (int i = 0; i < NOMBREDEMATCH; i++) {  capaciteAdversaire = Outil.aleatoireIntEntre(MIN, MAX);  if (capaciteNotre > capaciteAdversaire) {  nbVictoire++;  } else {  nbDefaite++;  }  }  System.out  .println("Dans cette competition, l'equipe a gagne " + nbVictoire + " fois, et a perdu " + nbDefaite + "fois.");  }  public final static Match getMatch() {  if (match != null) {  return match;  } else {  System.out.println("L'equipe ne s'engage dans aucun match.");  return null;  }  }  public final static Match getMatch(int capaciteNotre) {  match = new Match(capaciteNotre);  return match;  }  /\*\*  \* supprimer le singleton  \*/  public void finirMatch() {  nbVictoire = 0;  nbDefaite = 0;  match = null;  }  public int getCapaciteNotre() {  return capaciteNotre;  }  public void setCapaciteNotre(int capaciteNotre) {  this.capaciteNotre = capaciteNotre;  }  public int getCapaciteAdversaire() {  return capaciteAdversaire;  }  public void setCapaciteAdversaire(int capaciteAdversaire) {  this.capaciteAdversaire = capaciteAdversaire;  }  public int getNbVictoire() {  return nbVictoire;  }  public void setNbVictoire(int nbVictoire) {  this.nbVictoire = nbVictoire;  }  public int getNbDefaite() {  return nbDefaite;  }  public void setNbDefaite(int nbDefaite) {  this.nbDefaite = nbDefaite;  }  }  import java.util.ArrayList;  /\*\*  \* Cette classe ne comprend que des methodes  \* certaines sont des algorithme  \*/  public class Outil {  /\*\*  \*  \* @param min  \* @param max  \* @return Un double dans [min, max)  \*/  public static double aleatoireDoubleEntre(int min, int max) {  return Math.random()\*(max - min) + min;  }  /\*\*  \*  \* @param min  \* @param max  \* @return un entier entre [min, max]  \*/  public static int aleatoireIntEntre(int min, int max){  return (int) (Math.random()\*(max - min + 1) + min);  }  /\*\*  \* tirer une lettre aleatoire  \* pour le nom des personnes  \*/  public static char aleatoireLettre(){  return (char)((int)(Math.random()\*26)+'A');  }  /\*\*  \* tirer une chaine de caracteres aleatoire  \* pour le nom des personnes  \*/  public static String aleatoireChaine(){  String s = "";  for(int i = 0; i < 3; i++){  s += aleatoireLettre();  }  return s;  }  /\*\*  \* tirer n caracteres aleatoire  \* pour le nom des personnes  \* @param n  \*/  public static String aleatoireChaine(int n){  String s = "";  for(int i = 0; i < n; i++){  s += aleatoireLettre();  }  return s;  }  /\*\*  \* tire un nombre aleatoire entre 0 et 1, si plus grand que seuil, return true  \*/  public static boolean avoirLieuSeuil(double seuil)  {  return (Math.random() < seuil);  }  /\*\*\*  \* si victory number > defeat number, capacite +=1+ (int) (victory number - defeat number)/coefv(10)  \* si victory number < defeat number, capacite +=1+ (int) abs(victory number - defeat number)/coefd(20)  \* @param nbVictoire  \* @param nbDefaite  \* @return la difference de capacite a ajouter a la capacite originale  \*/  public static int capaciteDifference(int nbVictoire, int nbDefaite) {  if(nbVictoire > nbDefaite){  return 1 + (int) ((nbVictoire - nbDefaite)/10);  }else{  return 1 + (int) ((nbDefaite - nbVictoire)/20);  }  }  /\*\*\*  \*  \* @param remplacants  \* @return remplacant le plus capable (une instance)  \*/  public static Remplacant remplacantLePlusCapable(ArrayList<Remplacant> remplacants) {  if(remplacants.size() == 0){  return null;  }  Remplacant capmax = remplacants.get(0);  for(Remplacant r : remplacants){  if (capmax.capacite < r.capacite) {  capmax = r;  }  }  return capmax;  }  /\*\*\*  \*  \* @param joueurs  \* @return joueur le plus capable (une instance)  \*/  public static Joueur joueurLePlusCapable(ArrayList<Joueur> joueurs) {  if(joueurs.size() == 0){  return null;  }  Joueur capmax = joueurs.get(0);  for(Joueur j : joueurs){  if (capmax.capacite < j.capacite) {  capmax = j;  }  }  return capmax;  }  }  /\*\*  \* Une personne est soit jouer, soit manager, soit coach  \* On definit le salaire quand on etend cette classe  \*/  public abstract class Personne {  // attributs  protected String nom; // tirer au hazard par Outil.lettreAleatoire() 3 fois  protected double salaire; //en million  public static double salaireDefaut = 1;  // méthode  /\*\*\*  \* constructeur usuel avec salaire specifie  \* @param salaire  \*/  protected Personne(double salaire) //Constructeur  {  this.nom = Outil.aleatoireChaine();  this.salaire = salaire;  }  /\*\*\*  \* constructeur usuel avec salaire et nom specifie  \* @param salaire  \* @param nom  \*/  protected Personne(double salaire, String nom)  {  this.nom = nom;  this.salaire = salaire;  }  /\*\*  \* constructeur par defaut  \* salaire = salaireDefaut  \*/  protected Personne()  {  this(salaireDefaut);  this.nom = Outil.aleatoireChaine();  }  // getters and setters  public String getNom() {  return nom;  }  public void setNom(String nom) {  this.nom = nom;  }  public double getSalaire() {  return salaire;  }  public void setSalaire(double salaire) {  this.salaire = salaire;  }  // méthode abstraite  /\*\*  \* afficher des information pour l'utilisateur  \*/  public abstract void sePresenter();  }  /\*\*  \* Un remplacant peut devenir un joueur  \* Sa capacite ne sera pas prise en compte  \* et donc obtenir un salaire plus eleve  \*/  public class Remplacant extends Joueur{  // attribut  public static double salaireDefaut = 1;  protected final static double COEFFSALAIRECAPACITEREM = 1.;  private final static int MINR = 30;//temporaire  private final static int MAXR = 65;  //méthode  public Remplacant(int capacite, int age) {  super(capacite, age);  this.salaire = salaireDefaut;  }  /\*\*  \* constructeur par copie  \* @param r le remplacant a copier  \*/  public Remplacant(Remplacant r) {  this.capacite = r.capacite;  this.salaire = r.salaire;  this.age = r.age;  this.contrat = contratDefaut;  }  /\*\*  \* constructeur par defaut  \*/  public Remplacant()  {  super(Outil.aleatoireIntEntre(MINR, MAXR), Outil.aleatoireIntEntre(ageDefaut, ageDefaut + 3));  this.salaire = this.capacite \* COEFFSALAIRECAPACITEREM;  }  public void sePresenter(){  System.out.println(this.toString());  }  @Override  public String toString() {  return "Remplacant [nom=" + nom + ", age=" + age + ", capacite=" + capacite + ", contrat=" + contrat + ", salaire=" + salaire + "]\n";  }  } |