Interrogation d'informatique
PCC-ASINSA 2ème année - Avril 2016

Durée totale : *1h30*
Documents autorisés : *Aucun - téléphones portables interdits.*

Informations générales :
- Respectez les **conventions de codage** et choisissez des **noms de variables expressifs**.
- Le barème est indicatif (sur 24 pts, qui seront ramenés à 20).
- Une présentation de mauvaise qualité sera pénalisée (jusqu'à -1 pt).
- Le sujet est sur 6 pages, et les exercices sont distincts.

Exercice 1 Smart Playlist MP3

Contexte. Une start-up souhaite développer un nouveau système de lecteur MP3 intelligent qui s'adapte aux préférences de l'utilisateur. Un titre musical est actuellement défini selon le standard IDV3.2 :

```
public class MP3Tag{
  String title; // Titre de la chanson
  String artist; // Nom de l'interprète
  String album; // Nom de l'album
  String songWriter; // Nom de l'auteur compositeur
  String year; // Année de parution
  String comments; // Commentaire sur la chanson
  int track; // Numéro de la piste
  int genre // genre musical
  Image cover // Image de la pochette de l'album
}
  // son Constructeur
  public MP3Tag(String ti, String ar, String al, String s, String y, String com,
  int tr, int g, Image cov){
    title=ti; artist=ar; album=al; songWriter=s;
    year=y; comments=com; track=tr; genre=g; cover=cov;
}
```

La start-up décide de définir une nouvelle classe `MP3Tag2`, héritant de la classe `MP3Tag` en ajoutant trois nouvelles informations :

```
Date lastPlay; // Date de la dernière écoute
int playCount; // Nb fois que le titre a été écouté depuis son ajout dans la liste
int rating; // La note donnée par l'utilisateur
```

Ce nouveau baladeur MP3 propose de placer automatiquement les titres préférés (ayant le meilleur `rating`) au début de la liste de lecture et de supprimer automatiquement les titres qui ne sont plus écoutés depuis un mois et dont la note est nulle, situés en fin de liste. La préférence est définie par la valeur de la note (attribut `rating`), un titre étant d'autant plus préféré que sa note est grande. La note initiale donnée par l'utilisateur décroît automatiquement quand le titre n'est plus écouté. Le nouveau lecteur MP3 permet aussi d'ajouter de nouveaux titres à la playlist avec une note nulle, la date d'ajout du nouveau titre et un compteur mis à zero et d'insérer des anciens titres déjà notés.

Structure de données.

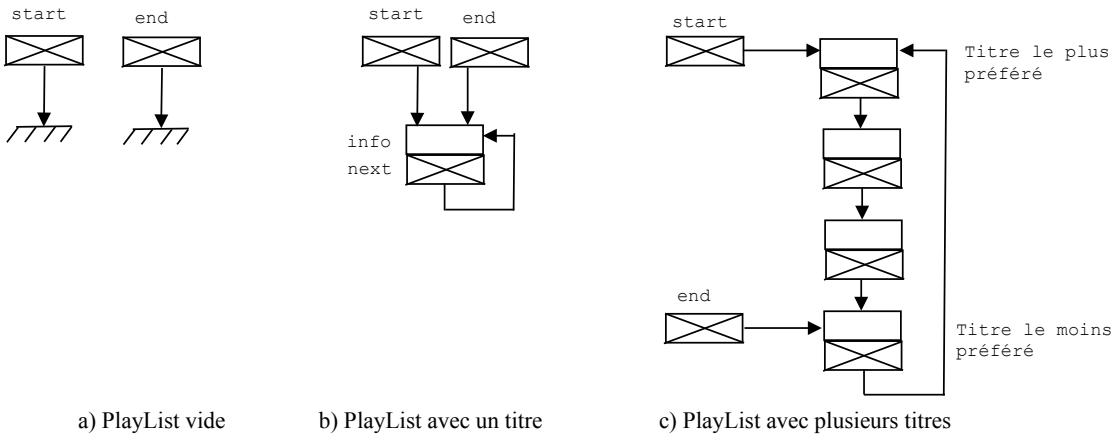

FIGURE 1 – Schémas de PlayLists.

La playlist sera codée par une liste chaînée mono-directionnelle circulaire dans laquelle le dernier élément pointe sur le premier élément de la liste (figure 1). La liste doit toujours être maintenue bouclée pendant toutes les opérations sur la playlist. Une liste circulaire possède la propriété de ne jamais se terminer, simplifiant ainsi la lecture en boucle infinie. La liste est définie par deux attributs qui pointeront le début et la fin de la liste. Ces pointeurs facilitent les méthodes d'insertion en tête et en fin de liste. On définit un maillon de la chaine de la manière suivante :

```
public class Element {
  public MP3Tag2 info; // Les informations MP3 étendues
  public Element next; // Référence vers le prochain élément
  public Element (MP3Tag2 aSong) // Constructeur
     info=aSong;
     next=null; // provisoirement avant son chaînage dans la playlist
} }
```

Une playlist est définie par :

```
public class PlayList {
  // Arrêt de la lecture par l'évènement extérieur "appui sur le bouton stop"
  boolean stopPlay;
  // Référence sur le début de la playlist pour la lecture de la liste
  public Element start;
  // Référence sur la fin de la playlist pour l'insertion d'un nouveau titre
  public Element end;
}
```

(Q1.1) Définition de la classe `MP3Tag2` (1 pt)

Définissez une classe fille `MP3Tag2` dérivée de la classe mère `MP3Tag` qui complète les informations standards avec les trois informations complémentaires `lastPlay`, `playCount`, `rating` (la note) et écrire son constructeur.
(Total 1pt)
public class MP3Tag2 extends MP3Tag {

Date lastPlay ;
int playCount ;
int rating ;
(0.5pt pour ne pas dupliquer les attributs de la classe mère et l'usage de extends)

public MP3Tag2(String ti, String ar, String al, String s, String y, String com, int tr, int g, Image cov, Date l, int p, int r){
super(ti,ar,al,s,y,com,tr,g,cov) ; (0.5pt pour le super)
lastPlay=l ;
playCount=p ;
rating=r ;
}
}

(Q1.2) Constructeur de la classe `PlayList` (0,5 pt)

Créez le constructeur de la classe `PlayList` qui crée une liste vide (voir figure 1a)
(Total 0,5 pt)
public PlayList() {
start=null ;
end=null ;
} Complétez la classe `PlayList` en définissant les méthodes suivantes :

(Q1.3) Définition de la méthode `play()` (1 pt)

Cette méthode joue les titres en boucle dans l'ordre décroissant des préférences. La liste est supposée déjà triée par ordre décroissant de l'attribut `rating` depuis la référence `start` jusqu'à la référence `end`. On sort de la boucle de lecture quand `stopPlay` est vrai lorsque l'utilisateur appuiera sur le bouton stop (code source supposé déjà implémenté). On utilisera la méthode statique `playSong(String title)` de la classe `AudioSystem` pour lire un titre. Le code `AudioSystem.playSong(titre)` où `titre` est une chaîne de caractères désignera donc le fait de jouer le titre. On supposera que tous les titres sont différents. Les attributs `playCount` et `lastPlay` doivent être mis à jour à chaque lecture. Pour ce dernier, on suppose disposer d'une méthode `getcurrentDate()` qui donne directement la date courante. La liste est supposée non vide.
(Total 1pt)
public void Play(){
Element p=start ;
while (!stopPlay) { (0.5 pour la boucle l'appel correct à PlaySong)
AudioSystem.PlaySong(p.info.title) ;
p.info.lastPlay= GetcurrentDate() ; (0.5 pour la mise a jour de lastplay+playCount)
p.info.playCount++ ;
p=p.next ; (et le chaînage correct)
}
}

(Q1.4) Définition de la méthode `addFirst(MP3Tag2 newSong)` (1,5 pt)

Cette méthode ajoute le tout premier titre dans une playlist vide (voir figure 1b). La liste est supposée vide, il est donc inutile de faire le test.

```
 (Total 1,5pt)
public void AddFirst(MP3Tag2 newSong) {
   Element p=new Element(newSong) ; (0,5pt pour la création d'un nouveau élément)
   p.next=p ; (0,5pt pour le chaînage correct de la boucle)
   start=p ; (0,25pt pour le chaînage correcte de start)
   end=p ; (0,25pt pour le chaînage correcte de end)
}
```

(Q1.5) Définition de la méthode `addNewSong(MP3Tag2 newSong)` (1,5 pt)

Cette méthode ajoute un nouveau titre non noté et donc, à la fin de la playlist. La note (`rating`) de ce nouveau titre doit être nulle, son compteur doit être mis à zéro. La liste est supposée contenir au moins un (voir figure 1b) ou plusieurs titres (voir figure 1c).

```
 (Total 1,5pt)
public void AddNewSong(MP3Tag2 newSong){
   Element p=new Element(newSong) ; (0,5 création d'un élément)
   p.info.lastPlay= null ; (facultatif - non noté)
   p.info.playCount=0 ; (0,25pt pour initialisations )
   p.info.rating=0 ;
   end.next=p ; (0,25pt pour le chainage à la fin de la liste)
   p.next=start ; (0,25pt pour le chainage du bouclage)
   end=p ; (0,25pt pour la mise à jour du pointeur de fin de liste )
}
```

(Q1.6) Définition de la méthode `add(MP3Tag2 aSong)` (3 pts)

Cette méthode ajoute un titre déjà écouté et déjà noté (`rating`) dans la playlist. La liste doit être triée par ordre décroissant de l'attribut `rating`, le titre le mieux noté étant situé en tête de liste pointée par l'attribut `start` et le titre le moins bien noté est situé en fin de liste, pointé par l'attribut `end`. La liste est supposée contenir un ou plusieurs titres (voir figure 1b et 1c). Il faut maintenir à jour les références `start` et `end` si on ajoute le titre aux extrémités de la liste. Nous vous conseillons de traiter en cas particuliers les cas de l'insertion en tête et de l'insertion en fin de liste.

```
 (Total 2,5 pt)
public void Add(MP3Tag2 aSong){
   Element p=new Element(aSong) ;
   // si on insère en tête de liste cela fonctionne meme si la liste contient un seul élément
   if (p.info.rating>start.info.rating) {
       p.next=start ;
       start=p ; //0.5pt test et insertion correct au début de la liste
       end.next=start ; //0.25pt pour la mise à jour de end.next car start change
    }
    else
    // si on insère en fin de liste cela fonctionne même si la liste contient un seul élément
       if (p.info.rating<end.info.rating) {
           end.next=p ; //0.5pt test et insertion correct à la fin de la liste
```

```
            end=p ;
            end.next=start ; //0.25pt pour la mise à jour de end.next
        }
        else {
            // chercher l'endroit ou insérer au milieu de la liste,
            // Avec les 2 tests précédents on est certain que la liste contient plus d'un élément
            Element previous=start ;
            Element current=start ; // 0,75pt recherche de l'endroit où insérer
            while ((current! =end)&&(current.info.rating>p.info.rating)) {
                previous=current ;
                current=current.next ;
            }
           // inutile de mettre à jour start et end ni de faire un test sinon noté si il y a des tests
            p.next=current ; //0.75pt insertion correct au milieu de la liste
            previous.next=p ;
        }
}
```

(Q1.7) Définition de la méthode `deleteLowRankedSongs()` (1,5 pts)

Cette méthode supprime tous les titres dont la note (`rating`) est nulle. La liste est supposée non vide et peut contenir un ou plusieurs titres. On supposera que l'on a déjà appelé une méthode qui recalcule les notes de tous les titres suivant la date de dernière écoute et le nombre de fois que le titre a été écouté. On suppose aussi que cette méthode a ré-ordonné la liste de lecture en fonction des nouvelles notes par ordre décroissant.

(Total 1,5pt)

```
    public void DeleteLowRankedSongs(){
    Element p=start ; // élément courant
    Element pp=start ; // élément précédent
    while ((p! =end)&&(p.info.rating>0)) {
        pp=p ; // 0.5pt pour chercher où supprimer
        p=p.next ; // avec un élément précédent
     }
    if ((start==end)&&(p.info.rating==0)) { // 0.5pt si la liste contient un seul titre
        start=null ; end=null ; // le garbage collector fera le ménage
     }
     else // sinon elle contient plusieurs titres
       if (p.info.rating==0) pp.next=start ; // 0.5pt pour boucler à partir du prédécesseur
}
```

Exercice 2 Questions générales BD

(Q2.1) Pour qu'un client (par exemple Workbench) se connecte à un serveur de base de données (exemple : le serveur MySQL du tp), il doit spécifier un certain nombre d'informations pour établir la connexion. Citez au moins trois d'entre elles. (0,75 pts)

3 réponses parmi : nom/adresse du serveur, nom de la base de données, port, login, mot de passe - 0,25 pt par bonne réponse

(Q2.2) Soit la relation `R(A int(5), B int(5))` de clé primaire `A`. Les tuples de la table suivante forment-ils un contenu valide de la relation `R` ? Justifiez votre réponse.(0,75 pts)

A	B
2	1
3	1
4	1

Oui ce contenu est valide : il respecte bien la contrainte de clé primaire, car à chaque valeur de A correspond bien une unique valeur de B ; autrement dit, on a une seule ligne (un seul tuple) par valeur de la clé primaire.
oui : 0,25 pt - explication : 0,5 pt

Exercice 3 Base de données pour le suivi d'une équipe de basket

Un club de basket a créé une base de données relationnelle pour suivre les joueurs de son équipe de ProA. Les informations sont mémorisées pour la saison courante uniquement. On suppose qu'un joueur ne change pas de numéro de maillot au cours d'une saison. Le schéma de la base est le suivant. Dans chaque relation, les attributs formant la clé sont soulignés.

`Joueurs(numero int(11), nomj varchar(50), taille number(5,2), poids number(5,2))`
Un joueur est identifié par son numéro de maillot. On mémorise son nom, sa taille, son poids.
`Matchs(dateMatch date, equipeAdverse varchar(60), ptsEquipe int(11), ptsAdverse int(11), lieu varchar(40))`
Un match est identifié par sa date car l'équipe ne joue jamais deux matchs le même jour. Pour un match, on mémorise le nom de l'équipe adverse, le nombre de points total obtenu par l'équipe, le nombre de points total obtenu par l'équipe adverse, le lieu du match (on indique "local" si le match a eu lieu à domicile).
`Stats(numero int(11), dateMatch date, nbTrois int(3), nbDeux int(3), nbFrancs int(3), nbRebonds int(3))`
Cette relation mémorise les statistiques de chaque joueur à chaque match qu'il a joué. Pour des raisons de simplification, on ne mémorise que le nombre de tirs marqués à trois points, à deux points, le nombre de lancers francs marqués et le nombre de rebonds saisis en défense. L'attribut `numero` est une clé étrangère référençant `Joueurs` ; `dateMatch` est une clé étrangère référençant `Matchs`.
`Seances(noSeance int(10), dateSeance date, heureSeance int(2), typeSeance varchar(40), idEntraineur varchar(10))`
Une séance d'entrainement est identifiée par un numéro. Elle a lieu à une date et une heure donnée (10h, 13h, 16h, 19h). On mémorise aussi son type qui indique ce qui a été le plus travaillé pendant la séance (endurance, renforcement musculaire, technique...). Tous les joueurs sont venus à la première séance du mois de janvier. L'attribut `idEntraineur` ne peut être indéfini et est une clé étrangère référençant la relation `Entraineurs`.
`Entrainements(numero int(11), noSeance int(10), commentaire varchar(200))`
Cette relation mémorise quelles séances d'entrainements les joueurs ont suivi. A chaque séance, l'entraineur peut laisser un commentaire sur chaque joueur ; s'il ne le fait pas, l'attribut est laissé indéfini (null). L'attribut `numero` est une clé étrangère référençant `Joueurs` ; `noSeance` est une clé étrangère référençant `Seances`.
`Entraineurs(idEntraineur varchar(10), nome varchar(50), tel varchar(10))`

Certaines informations sur les entraineurs sont mémorisées : l'identifiant, le nom et le numéro de téléphone de personnes qui sont déjà intervenues ou pourront intervenir comme entraineur d'une séance.

Au cours d'un match, le nombre total de points obtenus par l'équipe est la somme des points obtenus par chacun de ses joueurs. Le nombre de points obtenus par un joueur est égal à la somme :

— du produit de 3 par le nombre de tirs à trois points qu'il a marqués,
— du produit de 2 par le nombre de tirs à deux points qu'il a marqués,
— du nombre de lancers francs qu'il a marqués.

(Q3.1) Retro-conception (3 pts)

Proposez un modèle conceptuel utilisant la syntaxe UML qui, transformé en schéma relationnel selon les règles usuelles, produit le schéma relationnel précédent.

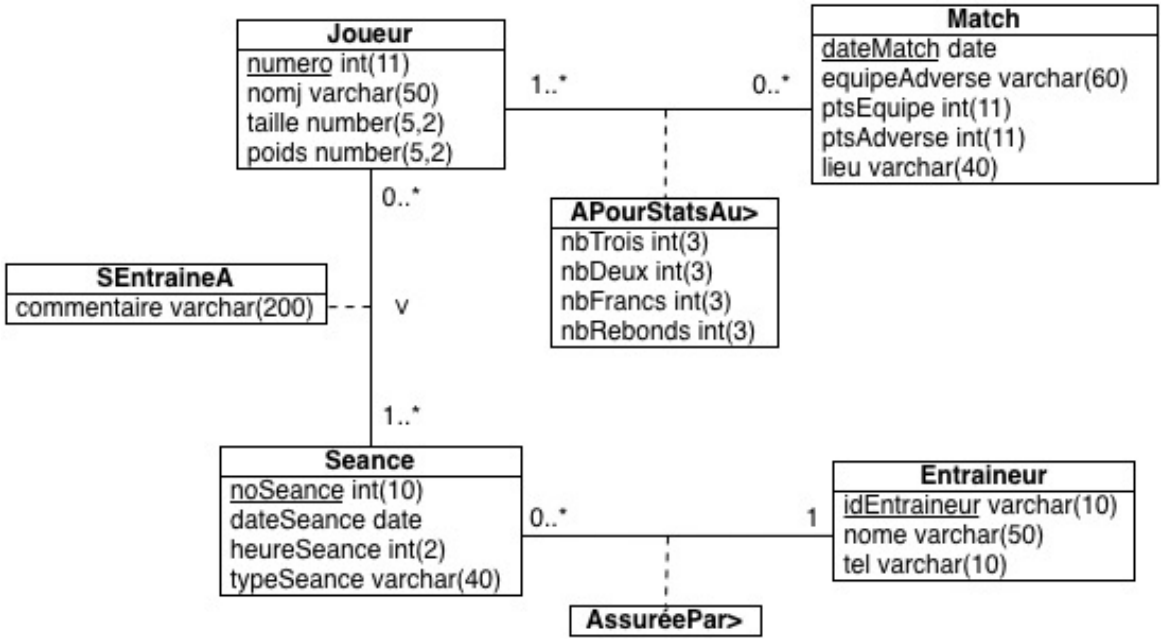

FIGURE 2 – Schéma conceptuel Basket (syntaxe UML).

1 pt par association correcte (y compris cardinalités et entités qu'elles joignent).

(Q3.2) Requêtes SQL (6,5 pts)

Pour chacune des recherches suivantes, écrivez une requête SQL qui permet d'afficher les résultats souhaités. Sauf mention contraire, les doublons seront éliminés, *mais l'utilisation de* `distinct` *de manière systématique et non pertinente sera sanctionnée.*

1. (0,5 pt) Combien de paniers à deux points ont été marqués par le joueur numéro 23 pour l'année en cours ?
   type requête : pas de jointure - sélection - fonction d'agregat
   select sum(nbDeux)
   from stats
   where numero = 23 ;
2. (1 pt) Quels sont les joueurs qui n'ont jamais fait l'objet d'un commentaire lors des séance d'entrainement ? Afficher leur numéro et leur nom.

type requête : imbriquée - not in
select numero, nom
from joueurs j
where numero not in (select distinct numero from entrainements where commentaire is not null) ;
La jointure naturelle avec une condition qui vérifie si l'attribut est null ne suffit pas pour "jamais". On peut aussi faire un count

3. (1 pt) Afficher le nombre total de points obtenus par chaque joueur depuis le début de la saison. On affichera le numéro de maillot, le nom du joueur, et le nombre de points.
type requête : jointure 2 relations - group by - fact sum - calcul
*select s.numero, nomj, sum(3*nbTrois+2*nbDeux+nbFrancs)*
from stats s, joueurs j
where s.numero=j.numero
group by s.numero, nomj ;

4. (1 pt) Afficher le nom des joueurs qui ont participé à la séance de l'entraineur 'Kunter' le 15-03-2016 et qui ont marqué au moins 4 tirs à 3 points lors du match du 18-03-2016. Afficher aussi le nombre de tirs à 3 points (nbTrois).
type requête : jointure plusieurs relations - bcp de sélections - pas de group by
select j.nomj, s.nbtrois
from stats s, joueurs j, seances c, entrainements e
where s.numero=j.numero
and s.numero=e.numero
and e.noseance = c.noseance
and s.dateMatch = '18-03-2016'
and s.nbtrois >=4
and c.entraineur = 'Kunter'
and c.dateseance = '15-03-2016' ;

5. (1 pt) Quelles sont les 3 séances les plus récentes où le joueur numéro 32 a travaillé l'endurance ? Afficher le numéro de séance, la date et l'heure ainsi que l'entraineur.
type requête : 1 jointure - order by - limit
select s.noseance, dateseance, heureseance, entraineur
from seances s, entrainements e
where s.noseance=e.noseance
and numero = 32
and type = 'endurance'
order by date seance desc
limit 0,2

6. (1 pt) Quelles sont les dates des matchs ayant eu lieu à domicile et qui ont été perdus par l'équipe et quelle était l'équipe adverse ? Afficher la date du match, le nom de l'équipe adverse et le nombre de points total de chacune des équipes. On souhaite présenter les résultats de sorte que les matchs où l'écart de points est le plus important apparaissent en premier.
type requête : condition de sélection et order by un soupçon plus original
select datematch, equipeadverse, ptsequipe, ptsadverse
from matchs
where lieu = 'local'

and ptsequipe<ptsadverse
order by ptsadverse-ptsequipe desc ;

7. (1 pt) Depuis le 01-01-2016, quelles sont les séances d'entrainement auxquelles ont assisté plus de 8 joueurs. On affichera tous les attributs de chaque séance et le nombre de joueurs présents.
select e.noseance, dateseance, heureseance, typeseance, entraineur, count(numero)
from entrainements e, seances s
where e.noseance=s.noseance
and dateseance>'01-01-2016'
group by e.noseance, dateseance, heureseance, typeseance, entraineur
having count(numero)>=8
Considérer group by ok dès que group by nuisance present.

(Q3.3) Conception (3 pts)

Le règlement interne de la ligue en charge des compétitions oblige tous les clubs de basket à mettre en place des mesures efficaces et démontrables pour lutter contre des produits dopants et/ou illégaux. Pour cela, chaque joueur doit passer des examens médicaux. Le médecin responsable d'un examen fixe la date de l'examen. Il envoie une convocation à un ou plusieurs joueurs, à une date qui précède celle de l'examen. Un joueur ne passe jamais un même examen deux fois le même jour.

Le club mémorise des informations sur les convocations adressées aux joueurs, sur les médecins responsables des différents examens, sur les examens eux-mêmes et sur leurs résultats. Il souhaite pouvoir poser des questions du type : Quel est le numéro, le nom, et le téléphone du médecin ayant envoyé la convocation d'identifiant 2016CO185 à la date du 13/03/2016 ? Quels sont les joueurs pour lesquels le résultat de l'examen d'identifiant 2016EX34 est 'positif' et quelle était la date de cet examen ? Quel est le protocole de l'examen d'identifiant 2016EX34 et quelle molécule teste-t-il (on suppose qu'un examen ne concerne qu'une seule molécule) ? Quel est le joueur concerné par la convocation d'identifiant 2016CO185 ? Le club souhaite qu'il n'y ait pas d'attribut indéfini à gérer.

Question : En utilisant la syntaxe UML, proposez un modèle conceptuel qui modélise ces informations et qui contient le type d'entité `Joueurs`. Il représente une extension du modèle UML de la question (Q3.1).

Pour gagner du temps, vous pouvez ne pas indiquer le type des attributs dans l'un et l'autre modèle.

0,5 pour les 4 entités, même si pbs d'attributs.
2 pts pour les 4 associations avec cardinalité 1.
0,5 pour l'association passe.
Ne pas enlever de points si les types des attributs ne sont pas indiqués.
Ne pas enlever de points si dateExamen est directement mis comme attribut du type d'entité Examen.
Accepter des cardinalités 0.. à la place des 0..*.*
Accepter tout modèle UML qui, transformé, donne le même schéma relationnel que celui-ci.

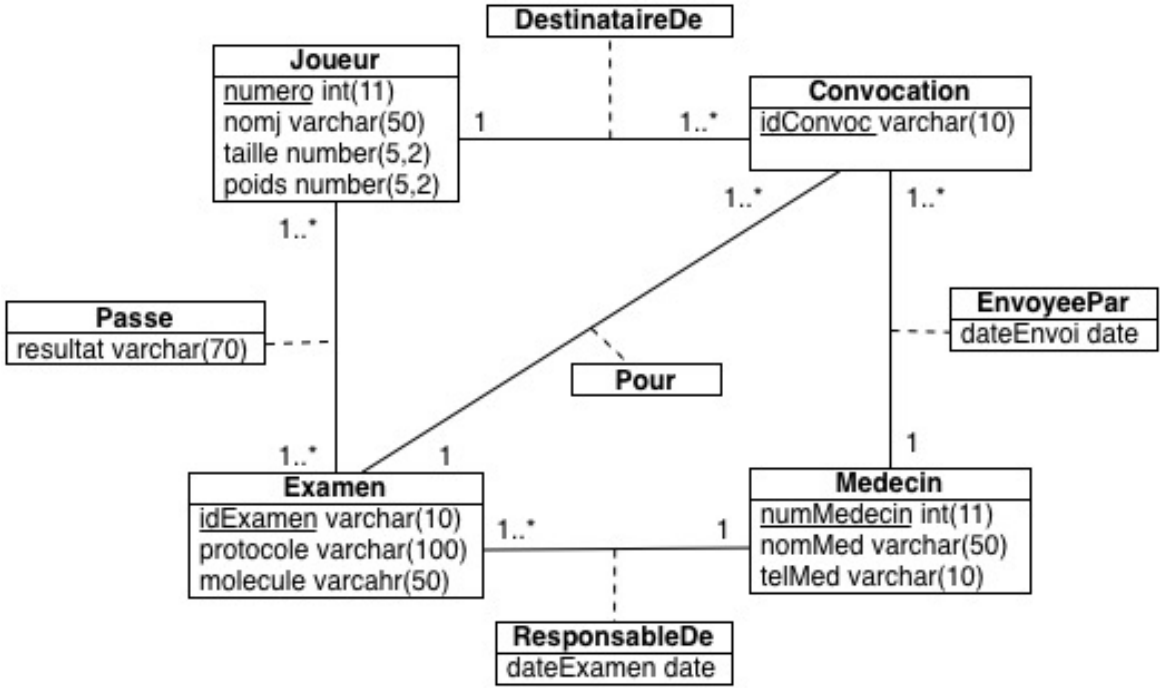

FIGURE 3 – Schéma conceptuel Examens médicaux (syntaxe UML).