# INFO-H-200 : Programmation orientée objet
# TP 5 - UML Diagrammes de classes

Professeur : Hugues Bersini
http://cs.ulb.ac.be/public/teaching/infoh200

Année académique 2015-2016

---

### Exercice 5.1 : Diagramme de classes

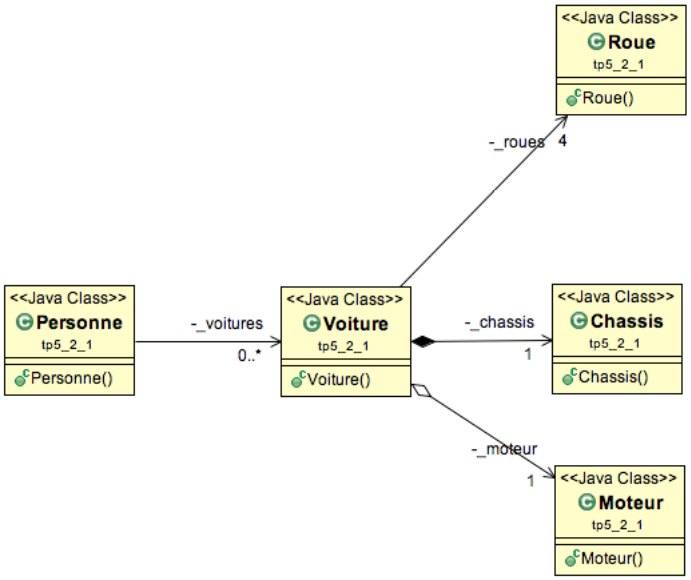
Dessiner pour les différentes situations suivantes le diagramme de classes simplifié correspondant :

1. Un pays possède une capitale. (1 étant la cardinalité par défaut)
2. Une personne utilise une fourchette pour diner.
3. Un fichier contient des enregistrements. Si l'on supprime le fichier, nous perdons donc les enregistrements liés.
4. Des personnes utilisent un langage pour un projet.
5. Une équipe est composée de plusieurs personnes. Le démantelement d'une équipe ne mène donc pas à la suppression des personnes qui travaillaient sur celui-ci.
6. Une route connecte deux villes.
7. Un dessin est soit du texte, soit une forme géométrique, soit un groupe de dessins.
8. Un ordinateur est composé d'un ou plusieurs moniteurs, d'un boîtier, d'une souris optionnelle et d'un clavier. Un boîtier possède un châssis métallique, une carte mère, plusieurs barrettes de mémoire (RAM, ROM et cache), un ventilateur optionnel, des supports de stockage (disquette, disque-dur, CD-ROM, DVD-ROM...), et des cartes périphériques (son, réseau, graphique...). Un ordinateur possède toujours au moins un lecteur de disquette ou un disque-dur.
9. L'Université comporte de nombreux inscrits dans ses registres ; du personnel administratifs et techniques, mais aussi des enseignants, des étudiants et des chercheurs (qui sont tous des personnes). Certains étudiants peuvent être des chercheurs (les doctorants) ou des enseignants (les assistants enseignants). Certaines personnes (étudiants ou non) peuvent être à la fois chercheurs et enseignants. Réalisez cet exercice en deux étapes : 1) Réaliser un diagramme selon l'intuition avec de multiples héritages. 2) Une classe ne pouvant hériter que d'une seule classe, proposez une solution employant des interfaces de manière appropriée. Aussi, veillez à spécifier si les classes sont "abstract", "interface" ou "final".

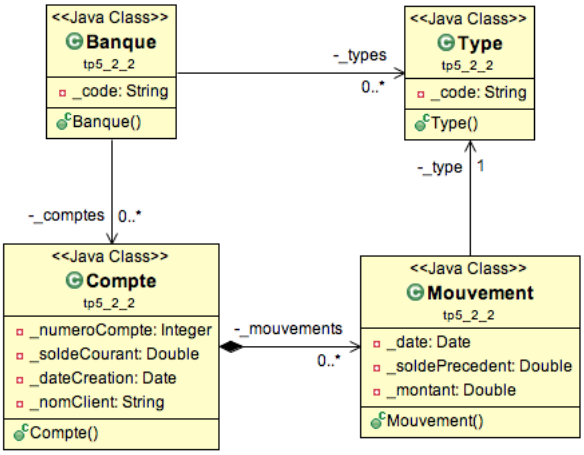
## Exercice 5.2 : Interprétation des diagrammes de classes

Expliquer et interpréter le contenu des diagrammes de classes partiels suivants. Donner ensuite le squelette de code Java correspondant à ces diagrammes.

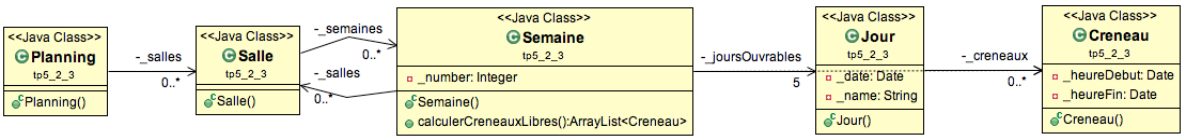
1.



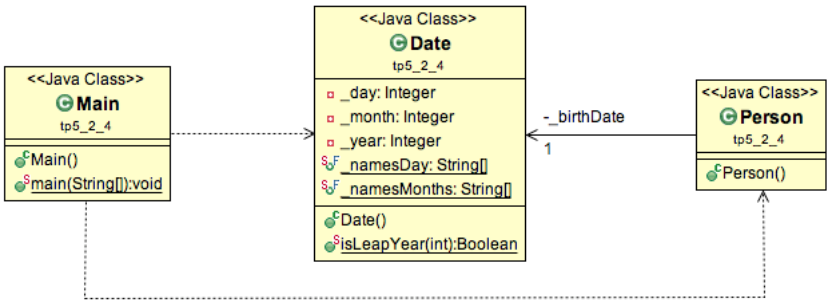

2.




3.

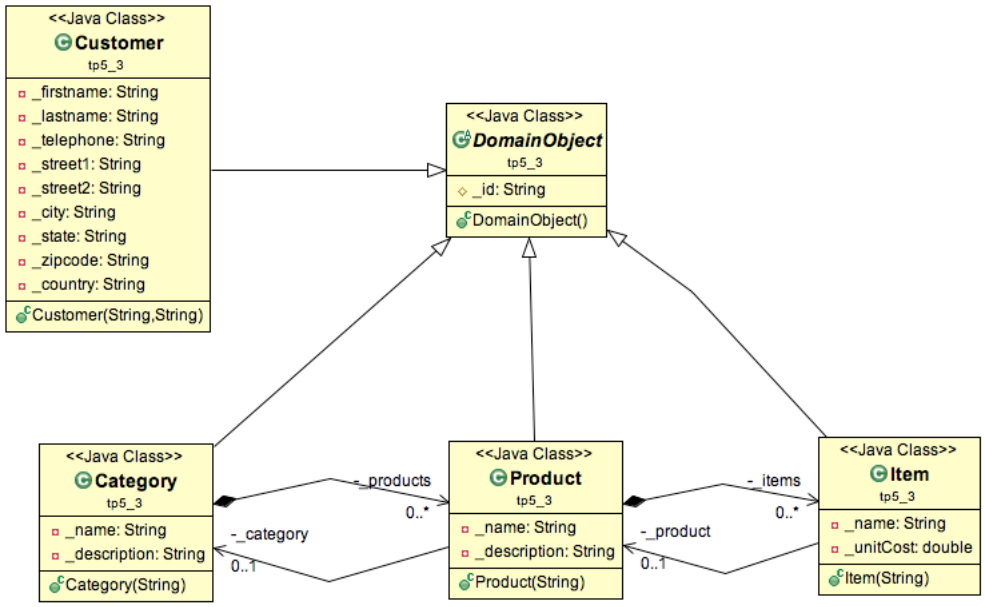



4.



## Exercice 5.3 : Implémentation de diagrammes de classes

Ecrivez le squelette de code Java qui pourrait être automatiquement généré à partir des diagrammes de classes UML suivants. 'Collection' fait référence à une ArrayList.




## Exercice 5.4 : Bomberban

Bomberman est un jeu vidéo où le joueur incarne une poseur de bombes, le but étant de faire exploser les adversaires pour gagner. Une partie se déroule sur un plateau (i.e : Plateau) carré de 15x15 blocs. Le joueur peut se déplacer sur certains blocs (i.e : BlocVide), d'autres sont des murs cassables (i.e : BlocCassable) alors que les derniers sont incassables (i.e : BlocIncassable). Lorsqu'un ennemi est tué ou que des blocs cassables sont détruits, des bonus (i.e : Bonus) peuvent apparaître comme par exemple : Une vie de plus (i.e : BonusVie), une bombe de plus (i.e : BonusBombe),...
Voici quelques indices supplémentaires :

- Prévoyez un fichier Main.java contenant la fonction public static void main(...). Vous instancieriez votre jeu depuis cette fonction.
- Prévoyez une classe Jeu.java instanciant les différents éléments nécéssaires au bon déroulement du jeu. (i.e : Plateau,...)
- Un jeu se compose d'un seul plateau.
- Le plateau de jeu se compose de 225 blocs.
- Chaque bloc du plateau peut être soit un BlocVide, un BlocCassable, un BlocIncassable ou un Objet (e.e : Bombe ou Bonus)
- Les instances de Joueur et Ennemi sont tous des personnages.
- Les Bloc et les Personnages doivent implémenter l'interface InteractionExplosion qui leur indique quelle méthode implémenter pour régir leur comportement lorsqu'ils subissent une explosion.
- Les Personnages implémentent l'interface InteractionBonus qui décrit la méthode loot() leur permettant de ramasser des objets.

Proposez un diagramme de classe modélisant le jeu décrit ci-dessus.

## Exercice 5.5 : Diagramme de classe d'un programme Java

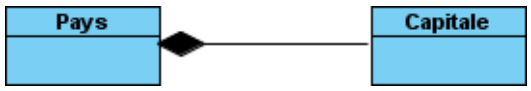
Construisez le diagramme de classes complet pour le code Java du programme Illustrator fourni au TP précédent.

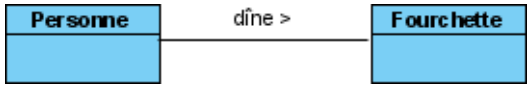
# INFO-H-200 - Programmation orientée objet
# TP 5

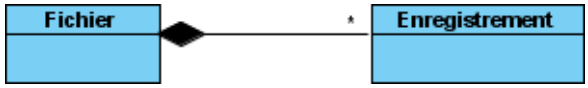
Corrections

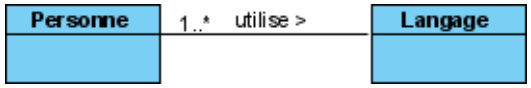
---

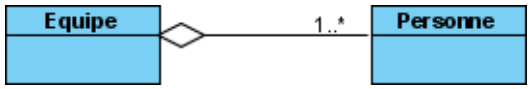
## Exercice 5.1

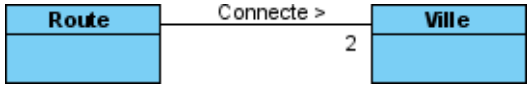
1.


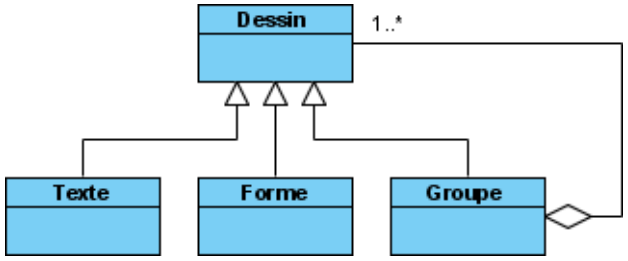

2.


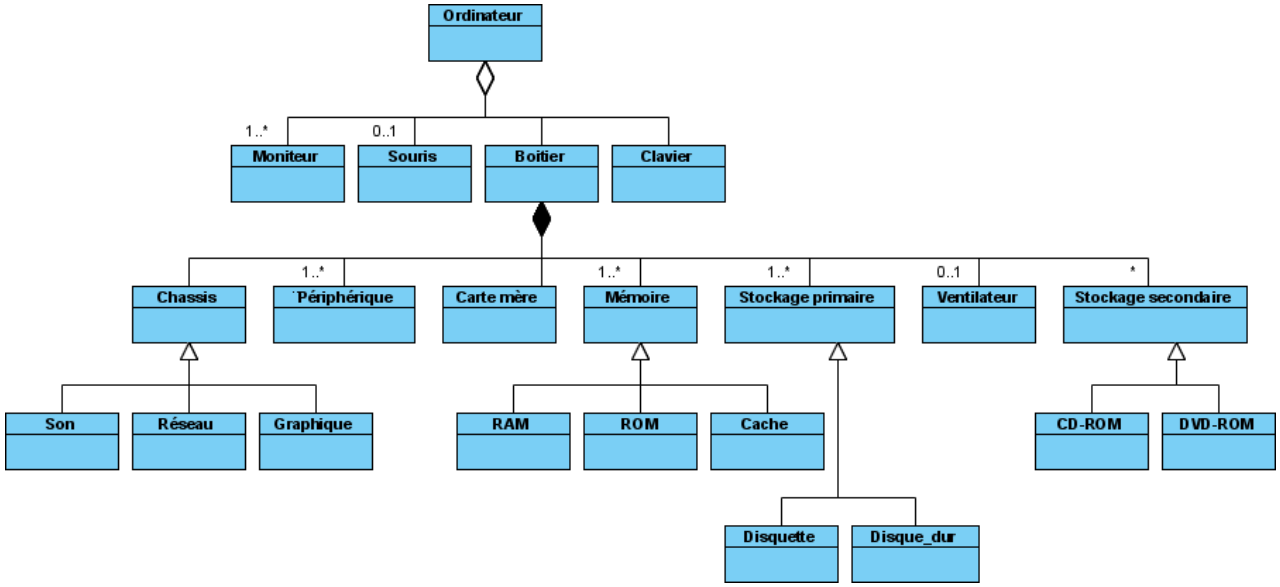

3.



4.


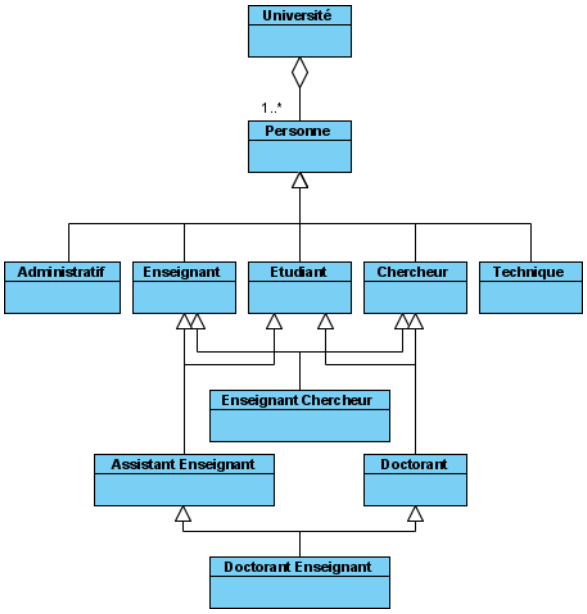

5.



6.


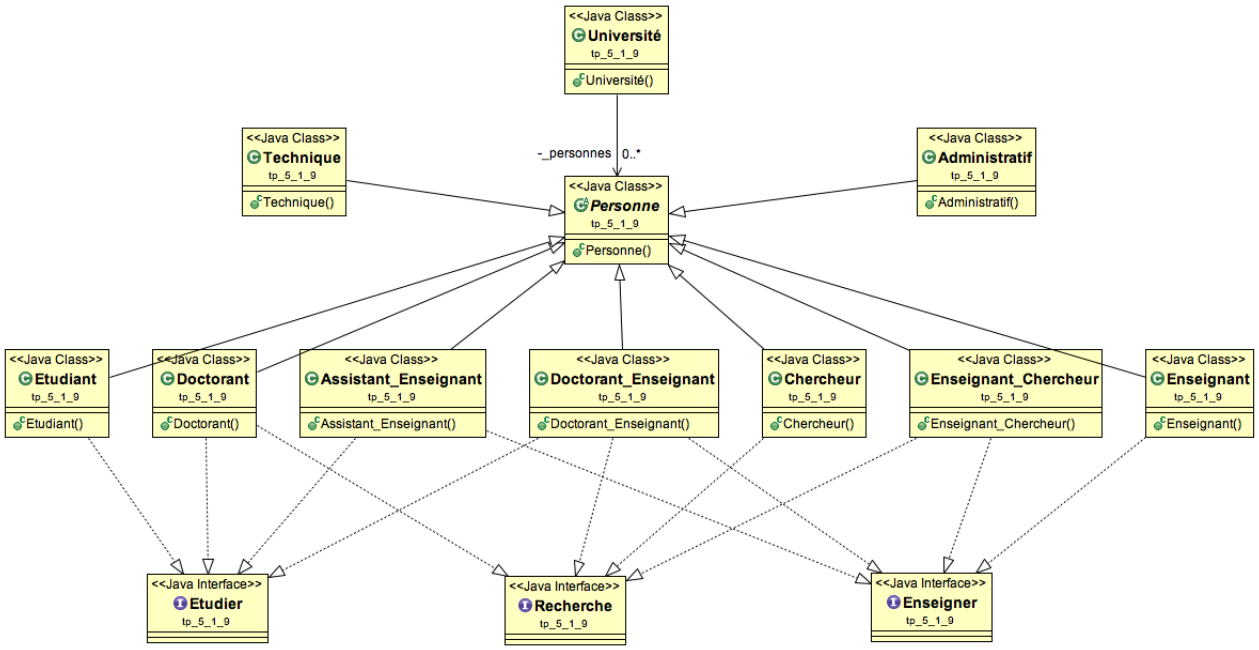

7.



8.


9. (a) Intuition :




(b) Avec interfaces :



## Exercice 5.2

Les fichiers Java correspondant sont fournis en attaché.

1. L'ensemble du diagramme décrit un voiture. Une voiture appartient : soit à aucun propriétaire, soit à une seule personne. Par contre, une même personne peut avoir de 0 jusqu'à une infinité de voitures. Le châssis est un élément indissociable d'une voiture, d'où la composition (i.e : agrégation forte). Par contre, le moteur et les roues peuvent être utilisés dans d'autres voitures. Une voiture a également obligatoirement quatre roues. Les cardinalité sont assimilables '1' sont assimilables à '1..1', de plus, les indiquer n'est pas obligatoire puisqu'elle sont la cardinalité par défaut (voir slides). Donc, un moteur ne peut servir que pour une seule voiture à la fois, et une voiture ne peut avoir qu'un seul moteur.
2. Un mouvement bancaire est une opération concernant un compte. Un mouvement possède une date de réalisation, un montant (en crédit ou en débit) et est d'un certain type. Il est dépendant d'un compte (si on supprime un compte, tous les mouvements disparaissent également) ; à l'inverse, les comptes sont indépendants d'une banque. Il existe plusieurs types de mouvements bancaires, mais un mouvement est d'un seul type ! Par contre, la banque en accepte de différentes sortes également.
3. La classe `Créneau` contient les informations d'un créneau horaire déterminé par une heure de début et une heure de fin. Un créneau est associé à un jour. Il permet d'identifier un créneau d'occupation d'une salle un jour donné.
   La classe `Jour` contient les informations sur l'ensemble des créneaux occupés ce jour. Un jour est déterminé par une date et un nom (jour de la semaine). Il est possible de savoir si un jour correspond à un jour férié et quels sont ses créneaux libres. Cette dernière opération est possible grâce à l'association navigable entre les classes Jours et Creneau, qui permet d'avoir la liste des créneaux occupés pour un jour donné. Cette classe permet donc d'identifier les créneaux d'occupation d'une salle pour un jour donné.
   La classe `Semaine` contient les informations sur les créneaux composant l'ensemble des jours de la semaine. Une semaine est représentée par son numéro. L'association navigable de la classe Semaine vers la classe Jour permet d'avoir accès aux informations sur les créneaux occupés de l'ensemble des jours de la semaine. Cette association est nécessaire pour la réalisation de l'opération calculerCreneauxLibres(), qui permet d'identifier les créneaux libres pour une semaine donnée. De plus, nous avons l'information suivante : une Semaine est composée de cinq jours ouvrables exactement. Cette classe permet donc d'identifier les créneaux d'occupation d'une salle pour une semaine donnée.
   La classe `Salle` contient les informations sur l'occupation d'une salle. Une salle est représentée par son numéro. L'association navigable entre la classe Salle et la classe Semaine permet de récupérer l'ensemble des informations sur l'occupation de la salle pour un nombre quelconque de semaines. Une semaine est associée à une seule salle. Une même semaine (en terme de numéro) existera en autant d'exemplaires que de salles occupées cette même semaine. Chacune de ces semaines représente l'occupation d'une salle en particulier. Cette classe permet donc d'identifier les créneaux d'occupation d'une salle pour un ensemble de semaines.
   La classe `Planning` contient les informations sur l'occupation d'un ensemble de salles (éventuellement vides). Elle ne contient aucune propriété. Grâce à une association vers la classe Salle, il est possible d'accéder à l'ensemble des salles et de récupérer les informations sur leur occupation.
4. Une Personne possède une Date de naissance. La classe Main instancie des objets de la classe Person. Elle utilise aussi la classe elle-même (e.g : Les attributs ou fonctions statiques).

# Exercice 5.3

## Fichier Customer.java

```
package tp5_3;

public class Customer extends DomainObject {
    private String _firstname;
    private String _lastname;
    private String _telephone;
    private String _street1;
    private String _street2;
    private String _city;
    private String _state;
    private String _zipcode;
    private String _country;

    public Customer(String firstname, String lastname) {
    }
}
```

## Fichier Category.java

```
package tp5_3;

import java.util.ArrayList;

public class Category extends DomainObject {
    private String _name;
    private String _description;
    private ArrayList<Product> _products;

    public Category(String name) {
    }
}
```

## Fichier Product.java

```
package tp5_3;

import java.util.ArrayList;

public class Product extends DomainObject {
    private String _name;
    private String _description;
    private ArrayList<Item> _items;
    private Category _category;

    public Product(String name) {
    }
}
```

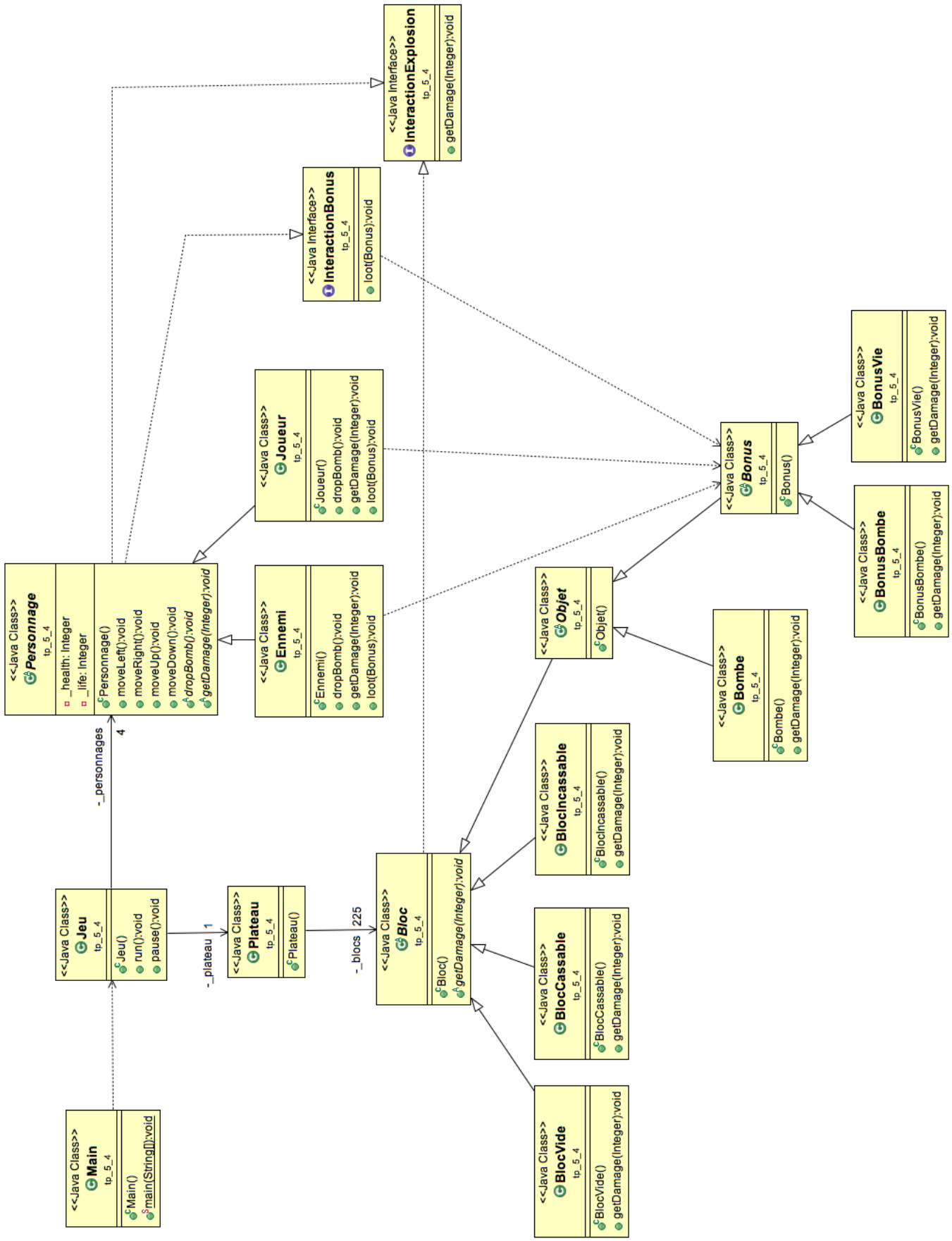
## Fichier Item.java

```
package tp5_3;

public class Item extends DomainObject{
    private String _name;
    private double _unitCost;
    private Product _product;

    public Item(String name) {
    }
}
```

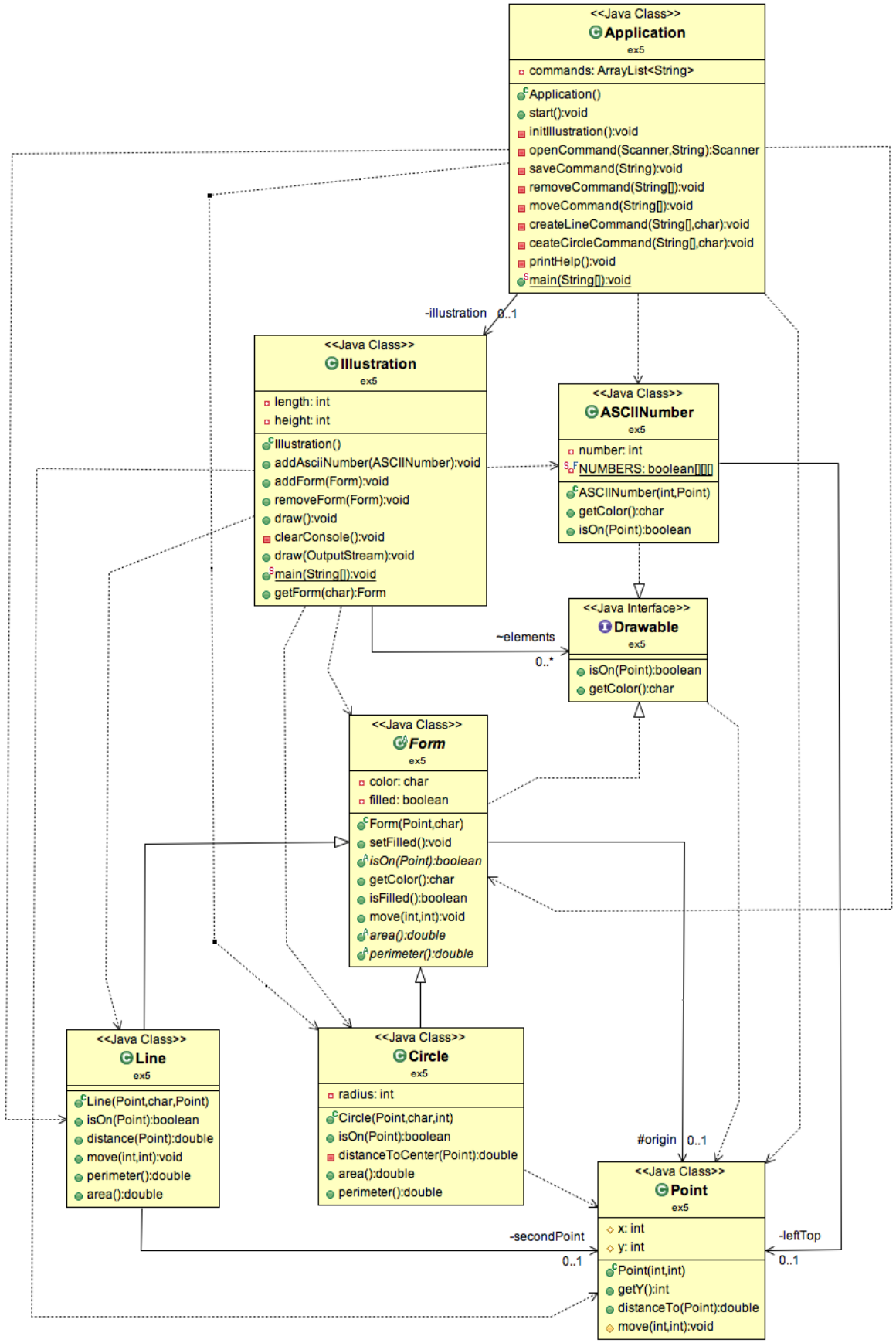
## Fichier DomainObject.java

```
package tp5_3;

public abstract class DomainObject {
    protected String _id;

    public DomainObject() {
    }
}
```

## Exercice 5.4


<<Java Interface>>
InteractionExplosion
tp_5_4
getDamage(Integer):void
<<Java Interface>>
InteractionBonus
tp_5_4
loot(Bonus):void
<<Java Class>>
Personnage
tp_5_4
_health: Integer
_life: Integer
Personnage()
moveLeft():void
moveRight():void
moveUp():void
moveDown():void
dropBomb():void
getDamage(Integer):void
<<Java Class>>
Joueur
tp_5_4
Joueur()
dropBomb():void
getDamage(Integer):void
loot(Bonus):void
<<Java Class>>
Ennemi
tp_5_4
Ennemi()
dropBomb():void
getDamage(Integer):void
loot(Bonus):void
<<Java Class>>
Bonus
tp_5_4
Bonus()
<<Java Class>>
BonusVie
tp_5_4
BonusVie()
getDamage(Integer):void
<<Java Class>>
BonusBombe
tp_5_4
BonusBombe()
getDamage(Integer):void
<<Java Class>>
Objet
tp_5_4
Objet()
<<Java Class>>
Bombe
tp_5_4
Bombe()
getDamage(Integer):void
- _personnages
4
<<Java Class>>
Jeu
tp_5_4
Jeu()
run():void
pause():void
- _plateau
1
<<Java Class>>
Plateau
tp_5_4
Plateau()
- _blocs
225
<<Java Class>>
Bloc
tp_5_4
Bloc()
getDamage(Integer):void
<<Java Class>>
BlocIncassable
tp_5_4
BlocIncassable()
getDamage(Integer):void
<<Java Class>>
BlocCassable
tp_5_4
BlocCassable()
getDamage(Integer):void
<<Java Class>>
BlocVide
tp_5_4
BlocVide()
getDamage(Integer):void
<<Java Class>>
Main
tp_5_4
Main()
main(String[]):void

## Exercice 5.5


<<Java Class>>
Application
ex5
commands: ArrayList<String>
Application()
start():void
initIllustration():void
openCommand(Scanner,String):Scanner
saveCommand(String):void
removeCommand(String[]):void
moveCommand(String[]):void
createLineCommand(String[],char):void
ceateCircleCommand(String[],char):void
printHelp():void
main(String[]):void
-illustration
0..1
<<Java Class>>
Illustration
ex5
length: int
height: int
Illustration()
addAsciiNumber(ASCIINumber):void
addForm(Form):void
removeForm(Form):void
draw():void
clearConsole():void
draw(OutputStream):void
main(String[]):void
getForm(char):Form
<<Java Class>>
ASCIINumber
ex5
number: int
NUMBERS: boolean[][][]
ASCIINumber(int,Point)
getColor():char
isOn(Point):boolean
<<Java Interface>>
Drawable
ex5
isOn(Point):boolean
getColor():char
~elements
0..*
<<Java Class>>
Form
ex5
color: char
filled: boolean
Form(Point,char)
setFilled():void
isOn(Point):boolean
getColor():char
isFilled():boolean
move(int,int):void
area():double
perimeter():double
<<Java Class>>
Line
ex5
Line(Point,char,Point)
isOn(Point):boolean
distance(Point):double
move(int,int):void
perimeter():double
area():double
<<Java Class>>
Circle
ex5
radius: int
Circle(Point,char,int)
isOn(Point):boolean
distanceToCenter(Point):double
area():double
perimeter():double
#origin
0..1
<<Java Class>>
Point
ex5
x: int
y: int
Point(int,int)
getY():int
distanceTo(Point):double
move(int,int):void
-secondPoint
0..1
-leftTop
0..1