



# TP : Implémenter des interfaces à partir d'un diagramme de classes simple

(Implémenter des conceptions simples)

 En fin d'énoncé, vous avez un exercice  
en autonomie, à faire en dehors des heures de cette séance 😊 

## Exercice 1 : Rendez-vous au zoo ...

Vous venez d'intégrer une équipe qui participe au développement d'une application pour le propriétaire d'un zoo. Le propriétaire a déjà fait part de ses besoins.

**La phase d'analyse** a montré que le propriétaire souhaite pouvoir disposer des informations suivantes :

-----  
Noise due to animals

-----  
George : Oink oink!  
Sarabi : Roar!  
Donald : Quack!  
Simba : Roar!  
Riri : Quack!  
Fifi : Quack!  
Loulou : Quack!  
Peppa : Oink oink!  
-----

Welcome Visitors !  
-----

Hello! I'm a visitor. I'm an adult.  
Hello! I'm a visitor. I'm an adult.  
Hello! I'm a visitor. I'm a child. I'm under 1 years old : I'm a baby !  
Hello! I'm a visitor. I'm a child. I'm under 1 years old : I'm a baby !  
Hello! I'm a visitor. I'm a child. I'm between 1 and 6 years old : I'm a toddler !  
Hello! I'm a visitor. I'm a child. I'm between 6 and 12 years old : I'm a school-age child!  
Hello! I'm a visitor. I'm a child. I'm between 12 and 18 years old : I'm a teenager !  
-----

Overall noise in zoo  
-----

George : Oink oink!  
Sarabi : Roar!  
Donald : Quack!  
Simba : Roar!  
Riri : Quack!  
Fifi : Quack!  
Loulou : Quack!  
Peppa : Oink oink!

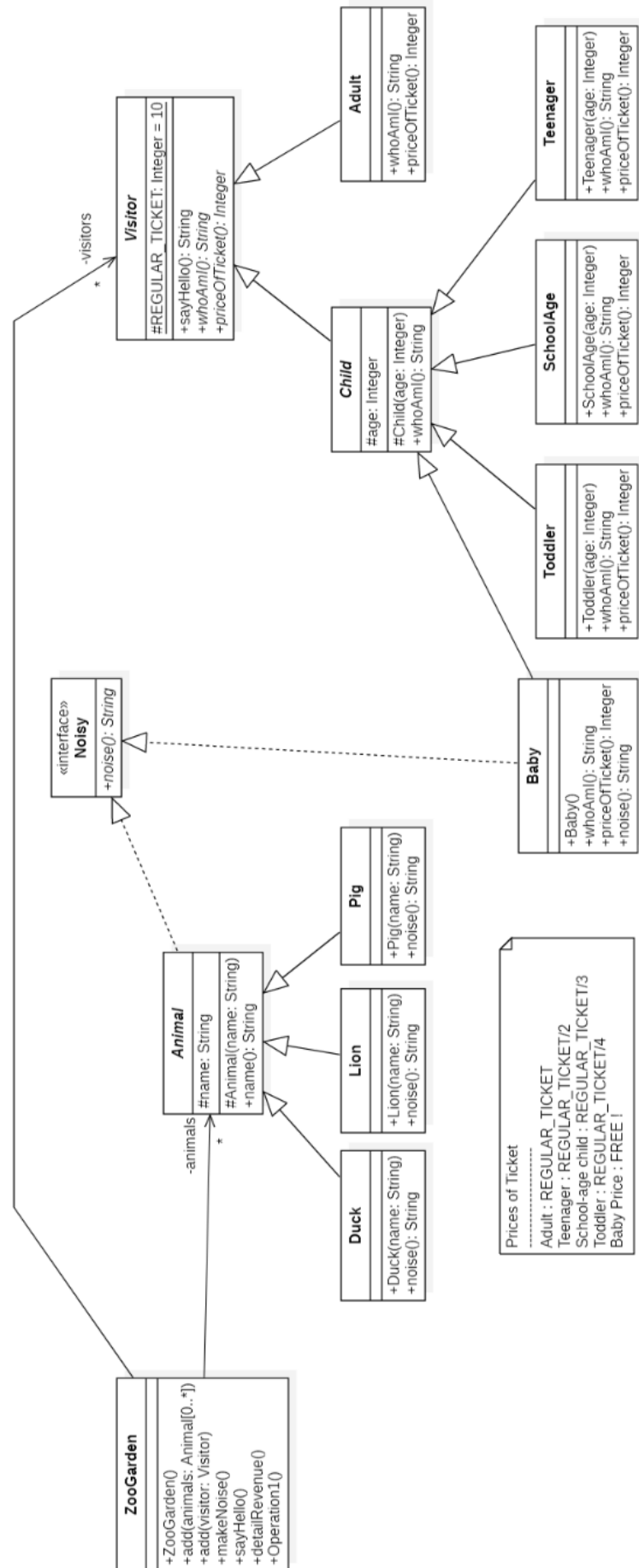
waaaaaaaaaaaa  
waaaaaaaaaaaa  
-----

Zoo Revenue  
-----

I'm an adult. : 10 Euros  
I'm an adult. : 10 Euros  
I'm a child. I'm under 1 years old : I'm a baby ! : 0 Euros  
I'm a child. I'm under 1 years old : I'm a baby ! : 0 Euros  
I'm a child. I'm between 1 and 6 years old : I'm a toddler ! : 2 Euros  
I'm a child. I'm between 6 and 12 years old : I'm a school-age child! : 3 Euros  
I'm a child. I'm between 12 and 18 years old : I'm a teenager ! : 5 Euros  
-----

Global Revenue : 30 Euros  
-----

**La phase de conception**, déjà réalisée par vos collègues, les a conduit à l'élaboration du diagramme de classes suivant :



Vous êtes maintenant chargé de **l'implémentation de cette application**.

→ Commencez par **créer un projet zoo**.

→ Dans un package **zoo.application**, récupérez la classe **ZooMain** disponible sur le **gist** :

<https://unil.im/zoomain>

(<https://gist.github.com/iblasquez/306444951d48cf7034ac3d8a2c8712ad> )

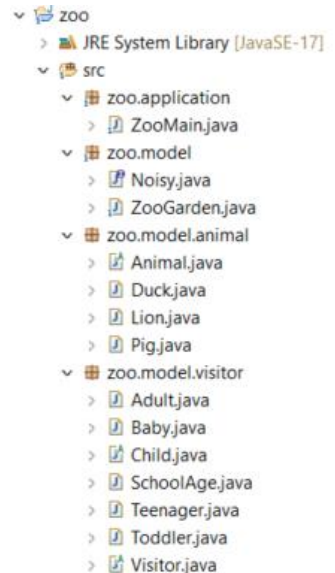
→ Dans un package **zoo.model** , récupérez la classe **ZooGarden** disponible sur le **gist** :

<https://unil.im/zoogarden>

(<https://gist.github.com/iblasquez/80abe4f518e4e7eec450ad74f5b42950>)

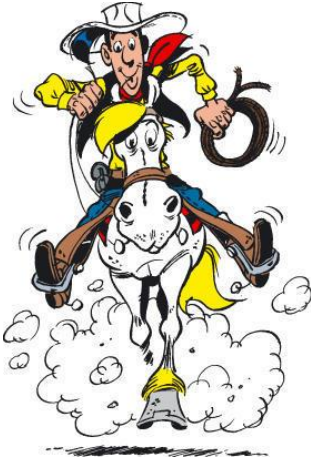
→ Maintenant, c'est à vous de jouer ! Il ne vous reste plus :

- qu'à **implémenter le diagramme de classes** précédent dans le package **zoo.model** et plus particulièrement à le répartir dans les packages **zoo.model.animal** et **zoo.model.visitor** de manière à ce que :
  - la **structure de votre projet zoo** soit similaire à la structure ci-contre.
  - la **rétro-conception de votre code** permette de retrouver un diagramme de classes conforme à celui qui a été réalisé dans la phase de conception et que vous deviez implémenter 😊
- qu'à **lancer la classe ZooMain**, une fois l'implémentation du diagramme de classes réalisée et **vérifier que vous obtenez bien sur la console l'affichage attendu par le propriétaire du zoo**. (celui de la phase d'analyse donné sur la première page de cet énoncé).  
Ne touchez pas à l'implémentation de **ZooMain** : c'est votre implémentation qui doit vous mener vers le jeu d'essai attendu !!!



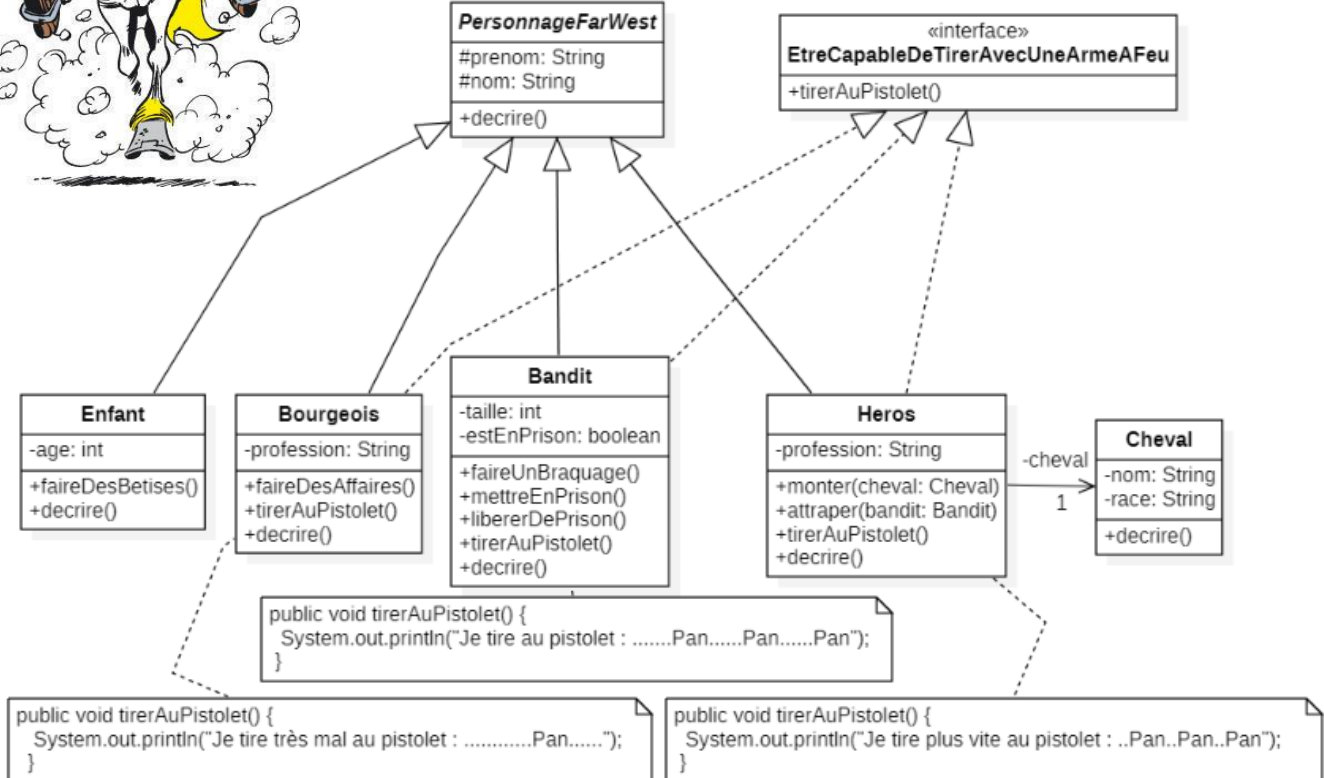
### Qualité de code :

- Vous veillerez bien sûr à éviter au maximum la duplication dans votre code
- Lorsque votre application fonctionnera et vous donnera l'affichage attendu, vous la passerez à **SonarLint** pour éventuellement peaufiner la qualité de code. Pour cet exercice, ne tenez pas compte des issues autour des `System.out.println`



## Exercice 2 : Rendez-vous au Far West ...

- ❑ Rendez-vous dans votre projet **farwest** ou créez le, si vous n'en disposez pas encore d'un 😊
- ❑ Implémentez le diagramme de classes suivant (vous vérifierez l'architecture avec une petite **rétro-conception** 😊)



### Remarques :

→ Pour les méthode faireDesBetises, faireDes Affaires et faireUnBraquage , un simple //TODO suffira comme implémentaion.

→ Pour la méthode décrire le résultat attendu à la console donné ci-dessous devrait vous aider 😊

❑ Récupérez la classe **FarWestMain** disponible sur le **gist** : <https://unil.im/farwest> (<https://gist.github.com/iblasquez/5597413cb37a30a2e7e8129f2d6aedf1> )

Faites-en sorte que ce code compile, puis exécutez-le pour vérifier et valider le comportement de votre application. Si votre implémentation est correcte, vous devriez obtenir sur la console un affichage similaire à :

-----  
Les personnages de la caravane sont :

Lucky Luke! Je suis cow-boy et mon cheval est Jolly Jumper de race appaloosa  
 Joe Dalton! Je mesure 150 cm et je suis Libre  
 Averell Dalton! Je mesure 190 cm et je suis Libre  
 Zacharie Martins! Je suis inventeur  
 Phineas ! J'ai 10ans

-----  
Les personnages capable de tirer au pistolet sont :

Lucky Luke ! Je tire plus vite au pistolet : ..Pan..Pan..Pan  
 Joe Dalton ! Je tire au pistolet : .....Pan.....Pan.....Pan  
 Averell Dalton ! Je tire au pistolet : .....Pan.....Pan.....Pan  
 Zacharie Martins ! Je tire très mal au pistolet : .....Pan.....

## Travail à faire en autonomie pour les deux prochaines semaines de TP :

*Et si vous cancaniez  
un peu avec votre canard  
en plastique : la correction !*



❑ Vous devez regarder la vidéo suivante :

<https://youtu.be/sDddTLViLHU> ( et uniquement cette vidéo)

afin de comprendre pourquoi la conception que vous avez proposée pose certainement quelques problèmes même si vous pensez avoir bien respecter les principes de conception orienté objet que vous avez vu jusqu'à présent : l'encapsulation, l'héritage, le polymorphisme, ... 😊

Cet exercice était juste là pour vous sensibiliser à la difficulté de mettre en œuvre une « bonne » conception. Il vous permet également de prendre conscience qu'il est important de passer du temps à **bien réfléchir à la conception** du projet avant de passer à son implémentation pour que ce dernier soit **facilement extensible** (eh oui, le client vous demandera toujours d'ajouter une nouvelle fonctionnalité qui n'était pas prévue au départ)

Bien sûr, nous n'attendons pas de vous, cette année, que vous soyez capable de produire une telle conception. Cette année est destinée à vous familiariser avec les **bases de la Conception Orientée Objet (COO)** : bien maîtriser les notions d'**abstraction**, d'**encapsulation**, d'**héritage** et **polymorphisme** serait déjà un bon point ! ...

**#teasing** : L'année prochaine (et tout au long de votre carrière de développeurs), vous découvrirez des notions plus avancées de COO (dont les fameux principes évoqués dans la vidéo, par exemple les design patterns) et reviendrez sans doute, avec plaisir, sur cet exemple de danses des canards, bien connu des développeurs 😊

❑ Si vous vous demandez à quoi ressemble une conception qui répond à toutes les besoins du problème initial. Vous trouverez ci-après le **diagramme de classes** proposé par la super équipe des concepteurs « Head First » et **extrait du livre « Design Patterns Tête la Première »** de E&E Freeman

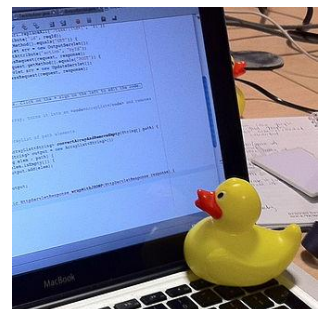
**Après avoir visionné la vidéo, nous vous conseillons de reprendre pas à pas l'énoncé du problème, et d'expliquer à votre canard en plastique, point par point, comment la conception proposée ci-après répond à toutes les spécifications et contraintes de cet énoncé.**

**Remarque** : La **méthode du canard en plastique** est une méthode, bien réelle, inventée par les développeurs et utilisée en génie logiciel dans les phases de débogage par exemple.

[https://fr.wikipedia.org/wiki/M%C3%A9thode\\_du\\_canard\\_en\\_plastique](https://fr.wikipedia.org/wiki/M%C3%A9thode_du_canard_en_plastique)

<https://medium.com/@ThomasGadroy/la-technique-du-canard-en-plastique-9ce33a37bb4>

<https://medium.com/nicoopratt/la-m%C3%A9thode-du-canard-en-plastique-d3ee48176925>



Cet exemple montre que le fait d'associer une interface à une classe (via une Relation A-UN) permet de favoriser le **découplage** entre les composants de l'application (classes/interfaces) donc de produire un code de meilleure qualité ... et de changer au besoin *le comportement* de vol ou de cancan à l'exécution 😊

