3.

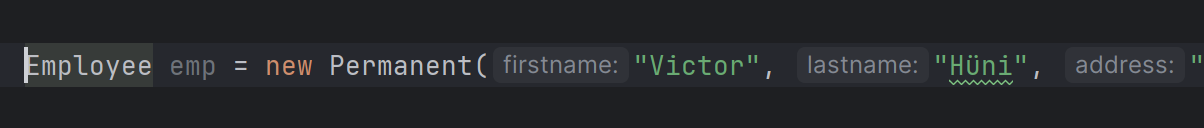
a) Dans mon modèle, je n’ai pas utilisé de classe et méthode abstraite car selon moi les classes les plus générales implémentent à chaque fois des attributs communs à ses classes spécialisés. Si l’on décide de bouger l’attribut experience de la classe fille TeachingEmployee vers la classe parent Employee on pourrait imaginer transformer la classe TeachnigEmployee en abstraite. L’utilité ici serait de garder la décorrélation entre Teaching et Adminsitrative Employee afin de permettre l’implémentation de méthode propre à chacun des deux types sans pour autant impacté l’autre.

b) Le polymorphisme par Overiding est utilisé pour les méthode calculateSalary() de la classe parent Employee mais la méthode de calcul va pouvoir différer pour les classe filles Permanet et Contractor. Dans notre code, l’implémentation n calculateSalary() m’est pas réalisée pour les classe fille AdministrativeEmployee.

J’ai utilisé du polymorphisme par Overloading dans le cadre des constructors avec différentes signatures pour les classes Permanet et Contractor qui permettent d’instancier des nouveaux collaborateurs enseignant avec ou sans leur données de salaires

c) « DRY » AKA “Do not repeat yourself” est une best practices de programmation qui conseille au développeur de ne jamais dupliquer du code. Au lieu de faire cela, il est donc préférable de créer une méthode qui implémentera ce code.

d) Admettons que dans mon application, l’instantiation des différentes classe fille d’Employee dans un objet typé Employee reviendra à du Dynamic Binding :



4)

1. Immeuble ne peut pas être instancié il faut soit faire x = new ImmeubleLocatif soit x = new ProprieteParEtage().

2. Batiment ne possède pas la méthode typeChauffage(). Z doit être typé au format Immeuble.

3. la méthode nombrePieces() est implémenté uniquement dans la classe fille Villa et doit donc être généralisé dans la classe abstraite MainsonIndividuelle pour forcer son implémentation dans Ferme également