Examen, 1h30, documents de l'UE GLS autorisés

NOM : **Prénom :** **Signature :**

Barème : Exo 1 : 1.5+1.5, Exo 2 : 3, Exo 3 : 3, Exo 4 : 1+1+2+2+2+3.

Exercice 1 Répondre de manière concise et précise aux questions suivantes.
1.1. Indiquer l'intérêt des outils vus en TP sur la partie « test » (JUnit, JaCoCo, Mockito, PITest) et dire en quoi ils sont complémentaires.
1.2. Expliquer les différents éléments du listing suivant. **Répondre sur le sujet.**

```
1 import java.lang.annotation.*;
2 @Documented
3 @Retention(RetentionPolicy.SOURCE)
4 @Target(ElementType.TYPE)
5 public @interface Utility {
6 }
```

Exercice 2 On considère une poignée `c` de type `Class<?>`.
2.1. Donner l'instruction Java qui permet d'initialiser cette poignée à partir du nom (chaîne de caractères) d'une classe.
2.2. Donner l'instruction Java qui permet de récupérer, dans la classe associée à `c`, la méthode publique de nom « m1 » qui prend un seul paramètre de type entier (`int`).
2.3. Donner les instructions Java qui permettent de collecter dans une liste toutes les méthodes publiques de la classe associée à `c` qui prennent un seul paramètre de type entier (`int`).
2.4. Donner le code Java qui permet de récupérer une méthode de nom « m1 » pas forcément publique de la classe associée à `c` (ou de l'une de ses superclasses) qui prend un seul paramètre de type entier (`int`).

Exercice 3 : Test structurel

On considère la fonction `mystere` suivante :

```
1  int mystere(int x) {
2      if (x <= 0) {
3          x = -x;
4      } else {
5          x = 1 - x;
6      }
7      if (x == -1) {
8          y = 0;
9      } else {
10         y = x - 1;
11     }
12     return 1 / y;
13 }
```

3.1. Dessiner le graphe de contrôle correspondant aux instructions de cette fonction.
3.2. Combien y a-t-il de chemins ?
3.3. Compléter le tableau suivant en indiquant pour chaque variable (en ligne) l'ensemble de ses utilisations : on donnera le numéro de ligne en distinguant les utilisations dans des calculs (*c-use* (*computation*)) et les utilisations dans des conditions (*p-use* (*predicate*)). On donnera aussi l'ensemble des paires *def-use* pour chaque variable.

variable	*c-use*	*p-use*	paires *def-use*
x			
y			
z			

Exercice 4 : Langage entités/relations (ER)

Nous définissons un langage de modèlisation de données selon l'approche entités/relations exploitée dans le domaine des bases de données. Ce langage s'appelle *ER*. Le listing 1 donne la description en Xtext d'une syntaxe textuelle d'une première version de ce langage.
4.1. Expliquer ce que signifient les éléments `Model` (ligne 3), `'model'` (ligne 3), `name` (ligne 3), `ID` (ligne 3), `entities` (ligne 4), `+=` (ligne 4), `Entity` (ligne 4), `*` (ligne 4), `entity` (ligne 13), `[Entity]` (ligne 13) du listing 1.
4.2. Donner le métamodèle de ER qui est produit par la grammaire Xtext du listing 1.
4.3. Donner la représentation tabulaire (ou le diagramme d'objets selon votre préférence), conforme au métamodèle ER, correspondant au modèle du listing 2.
4.4. Exprimer en OCL les contraintes suivantes :

1. Les noms des modèles, entités et relations ne doivent être ni nul ni de taille nulle.
2. Dans un modèle entités/relations, les noms des entités et des relations doivent être différents les uns des autres.

Listing 1 – Syntaxe textuelle pour ER décrite en Xtext

```
grammar fr.n7.gls.er.xtext.ER with org.eclipse.xtext.common.Terminals
generate er "http://www.n7.fr/gls/er/xtext/ER"
Model: 'model' name=ID 'is'
       (entities+=Entity | relations+=Relation)*
   'end' ;

Entity: 'entity' name=ID ';' ;

Relation:
   'relation' name=ID
   'between' parts+=Part ('and' parts+=Part)* ';' ;

Part: name = ID 'is' cardinal=Cardinal entity=[Entity] ;

enum Cardinal : One = "one" | Many = "many" ;
```

Listing 2 – Un exemple de modèle ER exprimé avec la syntaxe Xtext du listing 1

```
model test00 is
   entity Requin ;
   entity Jacques ;
   relation est_un between
      individu is many Jacques
      and
      espece is one Requin ;
end
```

Listing 3 – Transformation M2T Acceleo

```
[comment encoding = UTF-8 /]
[module toJava('http://www.n7.fr/gls/er/xtext/ER')]
[template public generateElement(aModel : Model)]
[comment @main/]
[for (anEntity : Entity | aModel.entities)]
[file (aModel.name + '/' + anEntity.name + '.java' , false, 'UTF-8')]
package [aModel.name/];
public class [anEntity.name/] { }
[/file]
[/for]
[for (aRelation : Relation | aModel.relations)]
[file (aModel.name + '/' + aRelation.name + '.java' , false, 'UTF-8')]
package [aModel.name/];
public class [aRelation.name/] {
[for (aPart : Part | aRelation.parts)]
  private [generateType(aPart)/] [aPart.name/];
[/for]
}
[/file]
[/for]
[/template]
[template public generateType(aPart : Part)]
[if (aPart.cardinal = Cardinal::Many)]Collection<[/if]
[aPart.entity.name/]
[if (aPart.cardinal = Cardinal::Many)]>[/if]
[/template]
```

4.5. Donner le résultat de la transformation M2T Acceleo du listing 3 quand elle est appliquée sur l'exemple du listing 2.
4.6. Le langage étudié précédemment est trop limité pour construire des modèles satisfaisants (l'exemple fourni par exemple n'est pas satisfaisant : `Requin` et `Jacques` ne devraient pas être des entités mais des instances d'entités dont `Requin` et `Jacques` seraient les noms.

Pour permettre la construction de ce genre de modèles, les modèles entités/relations associent aux entités et aux relations des attributs qui possèdent un nom, une cardinalité/multiplicité et un type (entier ou chaîne de caractères dans le cadre de cet exercice). Le listing 4 contient un exemple d'une extension possible de la syntaxe Xtext étudiée précédemment (listing 1).

Listing 4 – Un exemple de modèle ER exprimé avec une extension de la syntaxe Xtext du listing 1 que vous devrez définir

```
1  model test01 is
2     entity Espece qualified by
3        name is one string /* Requin par exemple */ ;
4     entity Individu qualified by
5        name is one string /* Jacques par exemple */ ;
6     relation est_un /* Contenu des especes par exemple */ qualified by
7        comments is many string
8        and
9        number is one integer
10    between
11       individu is many Individu
12       and
13       espece is one Espece ;
14 end
```

4.6.1. Compléter le métamodèle associé à la syntaxe Xtext initiale (listing 1) pour prendre en compte les attributs (on ne donnera que les nouvelles métaclasses et les relations associées).
4.6.2. Compléter le fichier Xtext du listing 1 (on ne donnera que les modifications du fichier précédent).
4.6.3. Compléter le fichier Acceleo du listing 3 pour générer les attributs privés correspondants dans les classes Java (on ne donnera que les modifications du fichier précédent).