U.B.S. - I.U.T. de Vannes
Département Informatique

TD3
(Partie GL)
R2.01
Développement Orienté Objets
BUT-Info-1A

I. Borne, R. Fleurquin, L. Naert, B. Le Trionnaire

Mars 2023

Objectifs du TD

- Cartographie de code Java sur le plan statique (diagramme de classes) et dynamique (diagramme de séquence).
- Découverte des notions objets : héritage, chaînage des constructeurs, accès et visibilité, package, interface, classe abstraite, membre statique.

Exercice 1

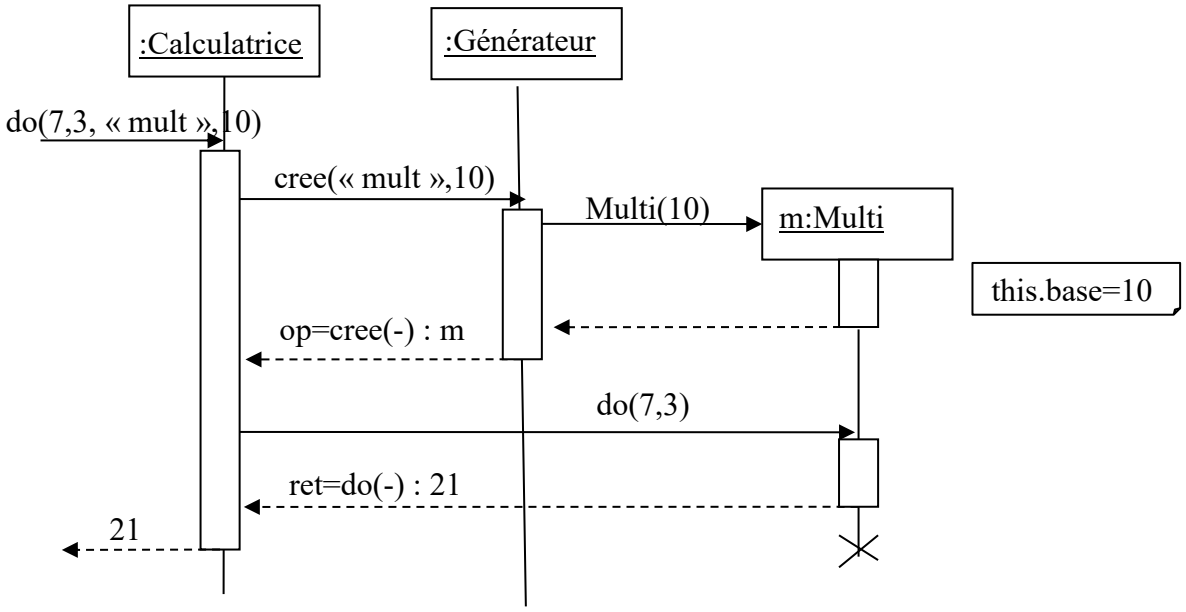
Soit le programme JAVA ci-dessous, donnez le diagramme de classes de conception lui correspondant.

```
class A {

      private B unB;
      private boolean a1;

      public A( ) {
            this.unB = new B();
      }

      public void m( C unC ) {
            D unD = new D();
      }
}

class B {
      private D unD;
      private int b1;
      public B( ) { this.unD = new D(); }
}

class C {
      public void n( C unC, D unD ) {;}
}

class D {
      public void o( ) { B unB = new B(); }
}
```

Exercice 2

On considère le diagramme de séquence ci-dessous décrivant une exécution particulière d'un système.

a) Identifiez dans ce diagramme, les lignes de vie, les zones d'activation (comprenez leur situation), les messages synchrones, les retours de message, les messages de création, les passages de paramètres et valeurs de retour, les variables locales éventuelles ou attributs.
b) Donnez le diagramme de classes décrivant la structure minimum requise pour mener à bien cette interaction.
c) Donnez la trame du code java que l'on peut déduire pour chacune des classes concernées.



Exercice 3

a) Donnez le diagramme de classes de conception du code ci-dessous.
b) Donnez le diagramme de séquence complet correspondant à l'exécution de la méthode `m()`. On ne représentera pas les affichages.

```
class A {
      public void m() {
            B unB = new B();
            C unC1, unC2;
            unC1 = unB.n();
            unC2 = unB.n();
            if ( unC1.egale( unC2 ) ) System.out.println( "Des jumeaux, comme
      c'est beau !" );
      }
}
```

```
class B {
      private int id;

      public B( ) { this.id = 1; }
      public C n() {
            (this.id)++;
            return new C ( this.id );
      }
}

class C {
      private int val;

      public C ( int v ) { this.val = v; }
      public boolean egale ( C copain ) {
            int v;
            boolean ret;

            v = copain.val;
            if ( v == this.val ) ret = true;
            else ret = false;

            return ret;
      }
}
```

Exercice 4

On considère le code Java ci-dessous.

a) Donnez sa représentation diagramme de classes de conception UML.
b) Représentez avec un diagramme de séquences l'exécution du code suivant l'appel de la méthode `exercice()` sur une instance de la classe `Lanceur`.
c) Ce programme illustre trois des modes possibles de connaissance de l'identité d'un autre objet. Lesquels ?
d) Deux instances différentes de la classe `Identite` ont-elles toujours des valeurs différentes pour leur attribut `monId` ? Pourquoi ?

```
class Lanceur {
      private IdentiteFactory aFactory;

      public Lanceur ( IdentiteFactory f ) { this.aFactory = f; }
      public void exercice() {
            Identite id1, id2;
            Worker w1;
            Worker w2 = new Worker();
            id2 = this.aFactory.getOne();
            w2.work ( id2 );
            w1= new Worker();
            id1 = this.aFactory.getOne();
            w1.work (id1);
      }
}

class Identite {
      private int monId;
      public Identite ( int a ) { this.setId( a ); }
      private void setId ( int a ) { this.monId = a; }
      public int getId() { return this.monId; }
}

class IdentiteFactory {
      private int cle;
      public IdentiteFactory( ) { this.cle = 1; }
      public Identite getOne() {
            (this.cle)++;
            return new Identite ( this.cle );
      }
}

class Worker {
      private int numero;
      public void work( Identite a ) {
            this.numero = a.getId());
      }
}
```

Exercice 5

Soit le code Java ci-dessous.

a) Donnez sa représentation diagramme de classes UML équivalente.

b) Dans quel ordre faut-il compiler ces classes une par une ?

c) Donnez la représentation diagramme d'objets UML d'une instance de chacune de ces classes.

d) Expliquez pourquoi la classe D est la seule qui ne compile pas. Que peut-on en déduire sur les membres hérités depuis une super-classe et leur usage dans une sous-classe ?

```
class A {
   private int a;
   public void ma(){this.a=1;}
}

class B extends A {
   private int b;
   public void mb(){...}
}

class C extends B {
   private int c;
   public void mc(){}
}

class D extends A {
   private int d;
   public void md(){this.ma() ; this.a=2 ;}
}
```

Exercice 6

On considère le code Java ci-dessous. Il compile et s'exécute sans erreur.

a) Donnez le diagramme de classes de cette application.

b) Dessinez une instance de la classe *Capitaine*.

c) Dans quel ordre faudrait-il compiler ces classes une à une ? En fait, on constate qu'il suffit de compiler *Capitaine.java* pour obtenir la compilation totale de l'application. Qu'a donc fait le compilateur *javac* ?

d) Expliquez ce que fait ce code à l'exécution (objet créé et valeurs de ses attributs).

e) On rajoute :

- à la classe *Homme* la méthode

```
public Homme(){System.out.println("Constructeur de Homme");}
```

- à la classe *Marin* la méthode

```
public Marin(){System.out.println("Constructeur de Marin");}
```

- on sauvegarde ces deux modifications et on recompile toutes les classes.

On obtient à l'exécution l'affichage :

```
Constructeur de Homme
Constructeur de Marin
```

Expliquez ce comportement.

f) On ajoute à la classe *Capitaine* la méthode

```
public Capitaine(){System.out.println("Constructeur de Capitaine");}
```

On sauvegarde et on recompile toutes les classes. On obtient l'affichage :

```
Constructeur de Homme
Constructeur de Marin
Constructeur de Capitaine
```

Expliquez ce comportement.

g) On rajoute à classe *Marin* un second constructeur.

```
public Marin(int g){System.out.println("Constructeur de Marin2");}
```

L'application compile sans problème mais l'exécution reste la même. Comment s'appelle cette possibilité en Java. Comment faire pour que la classe Capitaine use de cette seconde version de constructeur ?

h) On supprime, le premier constructeur sans paramètre de la classe *Marin* et l'on remet également l'ancien constructeur de la classe *Capitaine* (celui de la question 5). On recompile toutes les classes. Une erreur de compilation apparaît dans la classe *Capitaine* : « `constructor Marin cannot be applied to given types… required : int found no arguments` ». Expliquez cette erreur.

i) Modifiez le code Java ci-dessous pour initialiser « correctement » les attributs des classes *Capitaine*, *Marin* et *Homme*.

```
public class Bateau {
     private  int longeur ;
     private String type ;
}



public class Domicile {
     private String rue;
     private int numero ;
     private String ville;

}

public class Homme {
     private String nom;
     private int age;
     private Domicile monDom;
     private Homme[] amis;
}

public class Marin extends Homme {
     private  int grade ;
     private Bateau monBateau ;
}

public class Capitaine extends Marin {
     private int gradeC;
     public static void main(String[] args){
           Capitaine cook=new Capitaine();
     }
}
```

Exercice 7

Soit le code Java ci-dessous.

a) Donnez sa représentation diagramme de classes UML équivalente.

b) Dans quel ordre faut-il compiler ces classes ?

c) Donnez la représentation UML d'une instance de chacune de ces classes.

d) Quel serait le résultat d'un appel de la méthode `mc()` sur une instance de la classe C ?

e) Que se passe-t-il lors d'un appel à la méthode `mc()` sur une instance de la classe C si l'on remplace préalablement le code de la méthode `ma()` par `{ System.out.println ( "mal" ) ; }` sans recompiler la classe A ? Même question en ne recompilant que la classe A et non la classe C ?

```
class A {
   public void ma(){
         System.out.println("bien");
   }
}

class B extends A {
   public void mb(){
         System.out.println("va");
         this.ma();
   }
}

class C extends B {
   public void mc(){
         System.out.println("Tout");
         this.mb();
   }
}
```

Exercice 8

a) Soit le code Java ci-dessous, donnez sa représentation équivalente en diagramme de classes UML.

b) Quelles sont les instructions du programme ci-dessous réalisant un accès d'héritage et celles réalisant un accès inter-objets ?

c) Donnez le diagramme de séquence détaillant l'exécution de la méthode `mc()`. On ne représentera pas les messages provenant de `System.out.println`.

d) Donnez l'affichage résultant de l'exécution de la méthode `mc()`.

```
class A {
   public int a1;
   public A( ) { this.a1 = 1; }
   public int ma ( int v ) {
         return ( v + this.a1 );
   }
}

class B extends A {
   public int b1;
   public B( ) { this.b1 = 2; }
   public int mb ( int v ) {
         int i;
         A a = new A();
         i = this.ma ( this.a1 );
         i = i + a.ma ( a.a1 );
         return ( i + v );
   }
}

class C extends A {
   public int c1;
   public C( ) { this.c1 = 4; }
   public void mc() {
         A unA = new A();
         B unB = new B();
         unA.a1 = 0;
         this.c1 = this.a1 + unB.b1;
         System.out.println ( unB.mb(this.c1) );
   }
}
```

Exercice 9

Soit le code Java ci-dessous.

a) Quelles différences faites-vous entre usufruit d'héritage et droit d'accès inter-objets ? Rappelez quelles sont les règles d'usufruit et d'accès inter-objets en JAVA pour des membres déclarés successivement `public, protected, package`(rien) et `private`.

b) Donnez le diagramme de classes UML équivalent à ce code.

c) Donnez la représentation diagramme d'objets UML d'une instance de chacune des classes du code ci-dessous. Précisez en commentaire les membres hérités sans usufruit, les membres hérités avec usufruit et les membres définis par les classes dont sont issus ces objets.

d) Quelles sont les instructions du programme ci-dessous qui seront rejetées par le compilateur pour non respect des règles de droit d'héritage ?

```
package p;
public class A {
    public    float a;
    protected float b;
    float     c;
    private   float d;

    public void setD ( float v ) {
        this.d = v;
    }
    protected float getD1() {
        return this.getD2();
    }
    void setA ( float v ){
        this.a = v;
    }
    private float getD2() {
        return this.d;
    }
}

package p;
class B extends A {

    void n() {
        this.a = 2;
        this.b = 3;
        this.c = 4;
```

```
            this.d = 5;
            this.setD ( 3 );
            this.a = this.getD1();
            this.setA ( 3 );
            this.a = this.getD2();
      }

}
```

```
package q;
import p.A;
class C extends A {

      void n(){
            this.a = 2;
            this.b = 3;
            this.c = 4;
            this.d = 5;
            this.setD ( 3 );
            this.a = this.getD1();
            this.setA ( 3 );
            this.a = this.getD2();
      }

}
```

Exercice 10

a) Donnez le diagramme de classes UML équivalent au code ci-dessous.

b) Donnez la représentation UML d'une instance de chacune des classes du code ci-dessous. Précisez en commentaire les membres hérités sans usufruit, les membres hérités avec usufruit et les membres définis par les classes dont sont issus ces objets.

c) Quelles sont les instructions du programme ci-dessous qui seront rejetées par le compilateur pour des problèmes d'accès d'héritage et d'accès inter-objets ?

```
package p;
public class A {
     public int a1;
     protected int a2;
     int a3;
     private int a4;
     public A( ) {
           this.a1 = 1;
           this.a2 = 1;
           this.a3 = 3;
           this.a4 = 4;
     }
     private void ma() {
           A unA = new A();
           B unB = new B();
           unA.a4 = unB.a3;
           unB.a1 = unB.a2;
           unB.a4 = unA.a1;
     }
}

package p;
public class B extends A {
     public void mb() {
           A unA = new A();
           this.a1 = unA.a2;
           unA.a1 = this.a2;
           this.a3 = unA.a4;
           unA.a3 = this.a4;
     }
}
```

```
package p;
class C {
     protected void mc() {
          A unA = new A();
          B unB = new B();
          unA.a2 = this.a1;
          unA.a1 = unB.a2;
          unA.a3 = unB.a1;
          unA.a4 = unB.a3;
          unB.mb(); unA.ma();
     }
}

package q;
import p.*;
public class D {
     void md() {
          A unA = new A();
          B unB = new B();
          this.a2 = unA.a3;
          unB.a1 = 2;
          unA.a4 = unB.a2;
          unB.a3 = unA.a2;
          unB.a4 = unA.a1;
     }
}

package q;
import p.*;
class E extends B {
     void me() {
          B unB = new B();
          C unC = new C();
          this.a1 = 3;
          this.a2 = 4;
          unB.a1 = 2;
          unB.mb();
          unB.a2 = unB.a3;
          this.a3 = unB.a2;
          this.a4 = unB.a4;
     }
}
```

Exercice 11

a) Donnez le diagramme de classes UML correspondant au programme ci-dessous.

b) Donnez l'ordre de compilation des classes.

c) Indiquez quelles sont les instructions de ce programme réalisant un accès d'héritage et celles réalisant un accès inter-objets.

d) Donnez le diagramme de séquence correspondant à l'exécution de la méthode `main()`. On ne représentera pas l'instruction `println()` finale. Quel résultat est affiché après l'exécution de ce programme ?

```
package toto;
public class A {
      protected int a1;
      public A( ) { this.a1 = 1; }
      void ma1 ( int i ) { this.a1 = i; }
}

package toto;
class B extends A {
      private D unD;
      public B( ) { this.unD = new D(); }
      public void mb1 ( B b ) {
            b.a1 = 2;
            this.a1 = 3;
            b.ma1 ( 4 );
      }
}

package toto;
class C extends B {
      public static void main( String[] args ) {
            B b = new B();
            C c = new C();
            b.ma1 ( 2 );
            b.mb1 ( c );
            c.mb1 ( b );
            System.out.println ( ( b.a1 + c.a1 ) );
      }
}

package toto;
class D {...}
```

Exercice 12

a) Soit le programme JAVA ci-dessous, donnez le diagramme de classes de conception lui correspondant.

b) Expliquez le déroulement du constructeur de la classe `MotoHonda`.

c) Que se passerait-il si l'on compilait la classe `MotoHonda` après lui avoir rajouté le constructeur `MotoHonda() { ;}` ?

d) A quoi sert de mot clé *abstract* ? Quel est l'intérêt de déclarer une classe ou un membre de classe `abstract` ? Quelles sont les conséquences de cette déclaration pour les sous-classes ?

e) A quoi sert le mot clé `static` ? Quel est l'usage fait des membres `static` de la classe `MotoHonda` ?

```
abstract class Vehicule {
      protected String immat;
      protected Puissance uneP;

      Vehicule ( String im, Puissance puiss ) {
            this.immat = im;
            this.uneP = puiss;
      }
      public abstract void demarrer();
      public abstract void arreter();
}

class Moto extends Vehicule {
      private boolean etatM;
      private boolean isTrial;

      Moto ( String im, Puissance puiss, boolean trial ) {
            super ( im, puiss );
            this.isTrial = trial;
            this. etatM = false;
      }
      public void demarrer() { this.etatM = true; }
      public void arreter() { this.etatM = false; }
      public boolean getEtat() { return this.etatM; }
}
```

```
class MotoHonda extends Moto {
      private static int nbVendu = 0;
      public static final String marque = "Honda";
      private Proprietaire[] listeProp;
      private Modele unModele;

      MotoHonda ( String im, Puissance puiss, boolean trial, Modele mod, Proprietaire[] pro
) {
            super ( im, puiss, trial );
            this.unModele = mod;
            this.listeProp = pro;
            MotoHonda.addVendu();
      }
      public static int nbVendu() { return MotoHonda.nbVendu; }
      private static void addVendu() { (MotoHonda.nbVendu)++; }
 }

class Puissance { ... }

class Proprietaire { ... }

class Modele { ... }
```

Exercice 13

a) Soit le programme JAVA donné ci-dessous, donnez le diagramme de classes de conception lui correspondant.

b) Rappelez ce qu'est une interface et son intérêt.

c) Quelles différences faites vous entre une classe dont tous les membres sont abstraits et une interface ?

d) Parmi les instructions suivantes lesquelles sont correctes du point de vue du compilateur ?

```
SorteDeFruit f; SorteDeSphere s; s = new Fruit(); f = new Fruit();
s = new Orange(); f = new Orange();

interface SorteDeFruit {
      public void murir();
      public void cueillir();
}

class Fruit implements SorteDeFruit {
      private Color maCoul;
      private Date maDate;
      public void murir() { //ici le code }
      public void cueillir() { //ici le code }
}

interface SorteDeSphere {
      public void rouler();
}

class Orange extends Fruit implements SorteDeSphere {
      public void rouler() {//ici le code de l'implémentation }
      ...
}
```