INFO-H-200 : Programmation orientée objet
Correction Examen Juin 2015 - Partie JAVA

Hugues Bersini

Année académique 2014-2015

---

1 Question 1 : Java vers UML

Réalisez le diagramme de classes le plus complet qui soit que l'on pourrait générer à partir du code Java qui suit. Réalisez également le diagramme de séquence, également le plus complet qui soit, que l'on pourrait générer au départ de l'instruction "unTest.demarre()".

```
public class Examen5 {
  private ArrayList<IA> mesA = new ArrayList<IA>();

  public static void main(String[] args){
    Examen5 unTest = new Examen5();
    unTest.demarre();
  }

  public Examen5(){
    mesA = new ArrayList<IA>();
    mesA.add(new B(2,3));
    mesA.add(new C(3,4));
  }

  public void demarre(){
    for(IA a : mesA){
      a.faireA();
    }
  }
}
```

```
public abstract class A implements IA {
  private int a;

  public A(int a){
  }

  public abstract void faireA();
}
```

```
public class B extends A{
  private int b;

  public B(int a, int b){
    super(a);
  }

  public void faireA(){
    System.out.println("Je suis B");
    int a = 0;
    if (a == 0){
      new D();
    }
  }
}
```

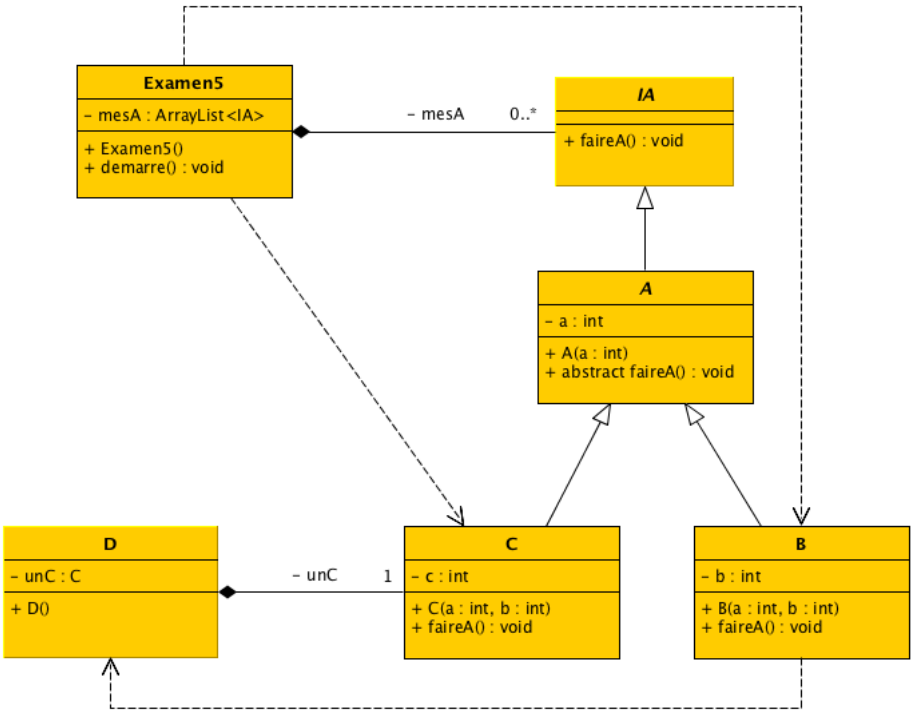
```
public class C extends A{
  private int c;

  public C(int a, int b){
    super(a);
  }
```

```
  public void faireA(){
    int a = 0;
    System.out.println("Je suis C");
    while(a == 0){
      new D();
      a += 1;
    }
  }
}
```

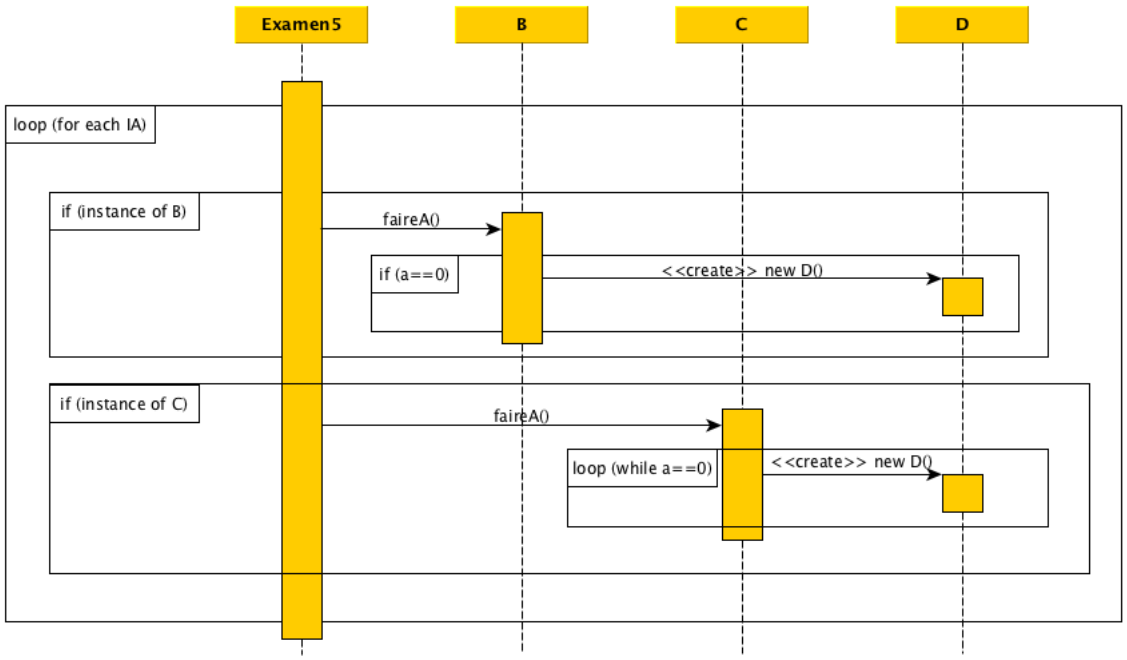
```
public class D {
  private C unC;

  public D(){
    unC = new C(2,4);
  }
}
```

```
public interface IA {
  void faireA();
}
```

1.1 Diagramme de classes



1.2 Diagramme de classes



2 Question 2 : Java

Corrigez à même le code les quatre petits programmes qui suivent de manière à ce qu'ils puissent s'exécuter sans erreur. Dans le code, indiquez les erreurs qui, en l'absence de correction, se produiraient à la compilation ou à l'exécution.

2.1 Code 1

```
public class Examen {
  public static void main(String[] args){
    Etudiant bob = new Etudiant("Bob", "Morane", 20);
    Etudiant marie = new Etudiant();            // Constructeur inexistant
    Etudiant Louis;
    Etudiant joelle = null;
    Louis.vieillit();                   // Objet Louis non défini
    joelle.vieillit();                    // Objet joelle non défini
    System.out.println(bob);
  }
}
```

```
public class Etudiant{
  private String nom, prenom;
  private int age;

  public Etudiant(String nom, String prenom, int age){
    this.nom = nom;
    this.prenom = prenom;
    this.age = age;
  }

  public vieillit(){                    // public void viellit();
    age++;
  }
}
```

2.2 Code 2

```
public class Compte {
  private static int nbre = 0;
  private int nom;

  public void augmente(){
    nbre++;
  }

  public static void donneNom(){
    System.out.println(nom);    // Cannot make static reference to non static field nom
  }
}
```

```
public class Examen2 {
  private Compte c;

  public static void main(String[] args){
    c = new Compte();           // Cannot make static reference to non static field c
    c.augmente();             // Cannot make static reference to non static field c
  }
}
```

2.3 Code 3

```
public class Examen3 {
  public static void main(Strin[] args){
    EtudiantPolytech x = new Etudiant("Delon", "Alain");
  }
}
```

```
public class Etudiant {
  private String nom, prenom;
  private int age;                // protected int age for age to be visible

  public Etudiant(String nom, String prenom){
    this.nom = nom;
    this.prenom = prenom;
    age = 18;
  }
```

```
  public void vieillit(){
    age+=1;
  }
}
```

```
public class EtudiantPolytech extends Etudiant{
  private String filiere;

  public EtudiantPolytech(String filiere){    // Missing super constructor call. Should receive nom & prenom as
      parameters then super(nom, prenom);
    this.filiere = filiere;
  }

  public int vieillit(){
    age += 2;          // field age is not visible
    return age;          // field age is not visible
  }
}
```

3 Code 4

```
public class Examen7 {
  public static void main(String[] args){
    B b = new B(1,2,"trois", "quatre");
    b.decrisToi();
  }
}
```

```
public abstract class A extends Object{
  private int a,b;
  private String c;

  public A(int a, int b, String c){
    super();
    this.a = a;
    this.b = b;
    this.c = c;
  }

  public abstract void decrisToi();
}
```

```
public class B extends A{
  private String d;

  public B(int a, int b, String c, String d){ // Super constructor missing. Remove this.a = a, this.b=b,
      this.c=c and replace by super(a,b,c);
    this.a = a;               // Could work if visibility of a,b and c was modified to protected
    this.b = b;
    this.c = c;
    this.d = d;
  }

  // public void decrisToi(){} is missing
}
```

4 Question 3

Retrouvez la sortie console du code sans erreur repris ci-dessous. Réalisez le diagramme de classes (sans les attributs ni les méthodes) qui reprend la structure du code.

```
public class Examen6 {
  private ArrayList<Etudiant> lesEtudiants;
  private Langage java,c,python;
  private Projet p1, p2, p3, p4;

  public static void main(String[] args){
    new Examen6();
  }

  public void creationEtudiant(){
    lesEtudiants = new ArrayList<Etudiant>();
    lesEtudiants.add(new EtudiantPolytech(1,true));
    lesEtudiants.add(new EtudiantPolytech(5,true));
    lesEtudiants.add(new EtudiantPolytech(7,true));
    lesEtudiants.add(new EtudiantPolytech(9,false));
    lesEtudiants.add(new EtudiantSolvay(3,true));
    lesEtudiants.add(new EtudiantSolvay(6,true));
    lesEtudiants.add(new EtudiantSolvay(7,false));
    lesEtudiants.add(new EtudiantSolvay(9,false));
    lesEtudiants.add(new EtudiantPsycho(2,false));
    lesEtudiants.add(new EtudiantPsycho(3,false));
    lesEtudiants.add(new EtudiantPsycho(1,false));
    lesEtudiants.add(new EtudiantPsycho(1,false));
  }

  public Examen6(){
    java = new LangageOO(10);
    python = new LangageOO(5);
    c = new LangageProceduraux(10);
    creationEtudiant();
    Projet[] lesProjets = new Projet[4];
    lesProjets[0] = new ProjetDifficile(java, lesEtudiants.get(0), lesEtudiants.get(4), lesEtudiants.get(8));
    lesProjets[1] = new ProjetFacile(java, lesEtudiants.get(1), lesEtudiants.get(5), lesEtudiants.get(9));
    lesProjets[2] = new ProjetDifficile(c, lesEtudiants.get(2), lesEtudiants.get(6), lesEtudiants.get(10));
    lesProjets[3] = new ProjetMoyen(python, lesEtudiants.get(3), lesEtudiants.get(7), lesEtudiants.get(11));
    int heuresTP = 0;
    for(Etudiant e: lesEtudiants){
      e.programme(heuresTP);
      heuresTP += 2;
      if(heuresTP == 10){
        heuresTP = 0;
      }
    }
    int temps = 1;
    for (Projet p : lesProjets){
      p.evaluer(temps);
      temps += 1;
    }
    for(Etudiant e : lesEtudiants){
      System.out.println(e.getCote());
    }

  }
}
```

```
public abstract class Etudiant {
  private int QI;
  private boolean sex;
  private Projet leProjet;
  private int cote;

  public Etudiant(int QI, boolean sex){
    this.leProjet = leProjet;             // Instruction inutile. Elle aurait pu être retirée.
  }

  public void recoitProjet(Projet leProjet){
    this.leProjet = leProjet;
  }

  public abstract void programme(int heuresTP);

  public int getQI(){
    return this.QI;
  }

  public boolean getSex(){
```

```
    return this.sex;
  }

  public void accroitQI(int x){
    QI += x;
  }

  public void setCote(int cote){
    this.cote = cote;
  }

  public int getCote(){
    return this.cote;
  }
}

public class EtudiantPolytech extends Etudiant{
  public EtudiantPolytech(int QI, boolean sex){
    super(QI, sex);
  }

  @Override
  public void programme(int heuresTP){
    if(getSex()){
      accroitQI(2*(heuresTP + 10));
    }else{
      accroitQI(heuresTP + 10);
    }
  }
}

public class EtudiantPsycho extends Etudiant{
  public EtudiantPsycho(int QI, boolean sex){
    super(QI, sex);
  }

  @Override
  public void programme(int heuresTP){
    if(getSex()){
      accroitQI(2*(heuresTP + 1));
    }else{
      accroitQI(heuresTP + 1);
    }
  }
}

public class EtudiantSolvay extends Etudiant{
  public EtudiantSolvay(int QI, boolean sex){
    super(QI, sex);
  }

  @Override
  public void programme(int heuresTP){
    if(getSex()){
      accroitQI(2*(heuresTP + 5));
    }else{
      accroitQI(heuresTP + 5);
    }
  }
}

public abstract class Langage {
  private int difficulte;
  public Langage(int difficulte){
    this.difficulte = difficulte;
  }

  public int getDifficulte(){
    return difficulte;
  }
}

public class LangageOO extends Langage{
  public LangageOO(int difficulte){
    super(difficulte);
  }
}

public class LangageProceduraux extends Langage{
  public LangageProceduraux(int difficulte){
    super(difficulte);
```

```
  }
}
```

```
public abstract class Projet {
  private Langage langage;
  private Etudiant[] lesEtudiants;
  private int cote;

  public Projet(Langage langage, Etudiant e1, Etudiant e2, Etudiant e3){
    this.langage = langage;
    lesEtudiants = new Etudiant[3];
    lesEtudiants[0] = e1;
    lesEtudiants[1] = e2;
    lesEtudiants[2] = e3;
    cote = 0;
  }

  public void evaluer(int temps){
    cote = temps*langage.getDifficulte();
    for(Etudiant e: lesEtudiants){
      cote += e.getQI();
    }
    evaluerEtudiant();
  }

  public void accroitCote(int x){
    cote += x;
  }

  public void evaluerEtudiant(){
    int x = (int) (cote/3);
    for(Etudiant e : lesEtudiants){
      e.setCote(x);
    }
  }
}
```

```
public class ProjetDifficile extends Projet{
  public ProjetDifficile(Langage langage, Etudiant e1, Etudiant e2, Etudiant e3){
    super(langage, e1, e2, e3);
  }

  public void evaluer(int temps){
    super.evaluer(temps);
    accroitCote(10);
  }
}
```

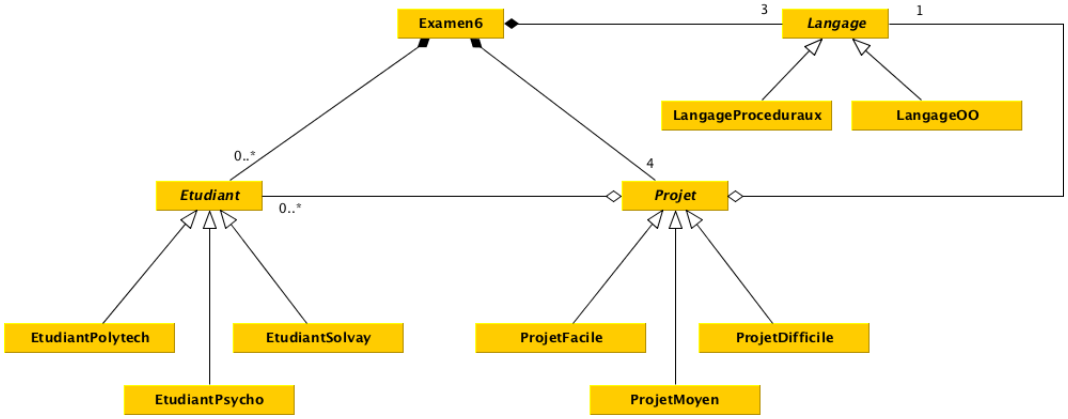
```
public class ProjetFacile extends Projet{
  public ProjetFacile(Langage langage, Etudiant e1, Etudiant e2, Etudiant e3){
    super(langage, e1, e2, e3);
  }

  public void evaluer(int temps){
    super.evaluer(temps);
    accroitCote(0);
  }
}
```

```
public class ProjetMoyen extends Projet{
  public ProjetMoyen(Langage langage, Etudiant e1, Etudiant e2, Etudiant e3){
    super(langage, e1, e2, e3);
  }

  public void evaluer(int temps){
    super.evaluer(temps);
    accroitCote(5);
  }
}
```

4.1 Sortie console

```
13
15
17
16
13
15
17
16
13
15
17
16
```

4.2 Diagramme de classes



5 Question 4

Rajoutez ce qui sera nécessaire afin que l'instruction : "Etudiant jobs = (Etudiant) gates.clone();" s'exécute correctement, en produisant ce qu'on serait susceptible d'attendre.

```
public class Examen4 {
  public static void main(String[]  args){
    Syllabus s[] = new Syllabus[2];
    s[0] = new Syllabus("Orienté Objet");
    s[1] = new Syllabus("Les fondements de l'informatique");
    Etudiant gates = new Etudiant("Gate","Bill",s);
    Etudiant jobs = (Etudiant) gates.clone();
  }
}
```

```
public class Etudiant {
  private String nom, prenom;
  private int age;
  private Syllabus[] mesSyllabus;

  public Etudiant(String nom, String prenom, Syllabus[] mesSyllabus){
    this.nom = nom;
    this.prenom = prenom;
    this.mesSyllabus = mesSyllabus;
  }

  //private String getNom(){
  //  return nom;
  //}

  //private String getPrenom(){
  //  return prenom;
  //}

  private int getAge(){
    return age;
  }

  //private Syllabus[] getSyllabus(){
  //  return mesSyllabus;
  //}

  //public Etudiant clone(){
  //  Etudiant temp = new Etudiant(this.getNom(), this.getPrenom(), this.getSyllabus());
  //  return temp;
  //}
}
```

```
public class Syllabus {
  public String intitule;

  public Syllabus(String intitule){
    this.intitule = intitule;
  }
}
```

6 Question 5

Réalisez dans une syntaxe approximative (mais respectenant les principes "Orienté-objet") un code Java contenant deux classes. Une classe "Point" avec ses coordonnées en x, y et une méthode permettant de calculer la distance euclidienne entre deux points et une classe "Cercle" dont le centre est un objet "Point" et comprenant des méthodes permettant de calculer l'aire, le périmètre, de translater et d'agrandir le cercle.

```java
public class Point {
  private int x;
  private int y;

  public Point(int x, int y){
    this.x = x;
    this.y = y;
  }

  public int getX(){
    return this.x;
  }

  public int getY(){
    return this.y;
  }

  public void setX(int x){
    this.x = x;
  }

  public void setY(int y){
    this.y = y;
  }

  public double distance(Point point){
    double distance = Math.sqrt(Math.pow((this.x-point.getX()),2)+Math.pow((this.y-point.getY()),2));
    return distance;
  }

  // A static method is imaginable with two Point as parameters
}
```

```java
public class Cercle {
  private Point centre;
  private int rayon;

  public Cercle(Point p, int rayon){
    this.centre = p;
    this.rayon = rayon;
  }

  public void agrandir(int rayon){
    this.rayon = this.rayon + rayon;
  }

  public void translate(int x, int y){
    this.centre.setX(this.centre.getX()+x);
    this.centre.setY(this.centre.getY()+y);
  }

  public double aire(){
    double aire = Math.PI*rayon*rayon;
    return aire;
  }

  public double perimeter(){
    double perimeter = 2*Math.PI*rayon;
    return perimeter;
  }
}
```