Interrogation Corrigée

1. Soit le code `Java` suivant :

```java
public class Voiture {
    private String marque;

    public Voiture(String marque) {
        this.marque = marque;
    }

    @Override
    public String toString() {
        return marque;
    }
}
```

```java
public class AppVoitures {
    public static void main(String[] args) {
        Voiture v1 = new Voiture("Peugeot");
        Voiture v2 = new Voiture("Renault");
        Voiture v4 = new Voiture("Mercedes");
        Voiture v3 = new Voiture("Citroën");

        v1 = v2;
        v2 = v3;
        v3 = v4;
        v4 = v1;

        System.out.println("Terminé");
    }
}
```

À la dernière ligne de la fonction `main(String args[])`, c-à-d juste avant d'afficher le message "*Terminé*", combien d'objets **Voiture** persistent dans le programme (à savoir, ne sont pas enlevés par le *Garbage Collector*) ? Justifiez.

Solution:
Il reste 3 objets. L'objet de marque `"Peugeot"` n'est plus accessible et va donc être ramassé par le *Garbage Collector*.

Et combien de références **Voiture** reste-t-il dans le programme au même moment ?

Solution:
Les 4 références persistent.

2. Dans les énoncés suivants, on considère que **B** est une classe qui hérite de **A**, et le terme "*caractéristique*" désigne à la fois les attributs et les méthodes.

   Indiquez le ou les énoncés qui sont **faux**.

   (a) **B** a hérité uniquement les caractéristiques de **A** qui ne sont pas privées.
   (b) **B** a hérité toutes les caractéristiques de **A**, qu'elles soient privées ou pas.
   (c) **A** a accès à toutes les caractéristiques de **B**, qu'elles soient privées ou publiques.
   (d) Une méthode dont un paramètre est de classe **A** peut être appelée en passant en argument une instance de classe **B**.

Solution:
Les énoncés faux : **a** et **c**
Lorsqu'une classe hérite d'une autre, elle hérite de l'intégralité d'attributs et méthodes et ce, bien que les méthodes/attributs privés demeurent inaccessibles.

3. Indiquez la ou les propositions qui sont **vraies**.

   Si **A** est une classe déclarée abstraite, cela implique

   (a) qu'elle doit obligatoirement déclarer une méthode concrète.
   (b) qu'elle doit obligatoirement déclarer une méthode abstraite.
   (c) qu'elle peut déclarer une méthode abstraite.
   (d) qu'elle ne peut être instanciée que si une de ses méthodes n'est pas abstraite.
   (e) qu'elle ne peut pas être instanciée.

Solution:
Les énoncés vrais : **c** et **e**
Une classe abstraite est non-instanciable par définition. Elle peut avoir des méthodes concrètes ou abstraites (potentiellement aucune).

4. Soit le code Java suivant :

```java
public class Etudiant {
    private String nom;

    public Etudiant(String nom) {
        this.nom = nom;
    }

    public String getEtablissement() {
        return "Universite Montpellier";
    }

    public String getIdentifiant() {
        return nom + " : " + getEtablissement();
    }
}
```

```java
public class EtudiantIUT extends Etudiant {
    private String departement;

    public EtudiantIUT(String nom, String dep) {
        super(nom);
        departement = dep;
    }

    @Override
    public String getEtablissement() {
        return departement;
    }
}
```

```java
public class AppEtudiants {
    public static void main(String[] args) {
        Etudiant e1 = new EtudiantIUT("Loulou", "Info");
        Etudiant e2 = new Etudiant("Riri");
        EtudiantIUT e3 = new EtudiantIUT("Fifi", "Chimie");

        System.out.println(e1.getIdentifiant());

        e1 = e2;
        e2 = e3;

        System.out.println(e1.getIdentifiant());
        System.out.println(e2.getIdentifiant());
        System.out.println(e3.getIdentifiant());
    }
}
```

Y a-t-il des erreurs de compilation ou exécution dans ce programme ?
Si oui, expliquez brièvement lesquelles.
Si non, donnez les résultats de l'exécution de la fonction principale.

Solution:
Il n'y a aucune erreur. On applique correctement le principe de substitution.
Le résultat d'affichage :
Loulou : Info
Riri : Universite Montpellier
Fifi : Chimie
Fifi : Chimie

5. Soit le code Java suivant :

```java
public class VeloElectrique extends Velo {
    private int boost;

    public VeloElectrique(int vitesse, int boost) {
        super(vitesse);
        this.boost = boost;
    }

    @Override
    public int getVitesse() {
        return super.getVitesse() + boost;
    }
}
```

```java
public class Velo {
    private int vitesse;

    public Velo(int vitesse) {
        this.vitesse = vitesse;
    }

    public int getVitesse() {
        return this.vitesse;
    }

    public int getVitesseMax() {
        return this.getVitesse() + 5;
    }
}
```

```java
public class AppVelos {
  public static void main(String[] args) {
     Velo velo = new VeloElectrique(15, 10);
     VeloElectrique veloElec = new VeloElectrique(15, 10);

     System.out.println(velo.getVitesse());
     System.out.println(velo.getVitesseMax());

     System.out.println(veloElec.getVitesse());
     System.out.println(veloElec.getVitesseMax());
  }
}
```

Quel est le résultat d'affichage de la fonction principale `main(String args[])` ? Justifiez.

Solution:
25
30
25
30
Les deux objets référencés par **velo** et **veloElec** sont construits de manière identique. Ainsi, le résultat d'exécution des différentes méthodes sont identiques pour les deux objets.

6. Soit le code Java suivant :

```java
public abstract class Animal {
    private String nom;

    public Animal(String nom) {
        this.nom = nom;
    }

    public void manger() {
        System.out.println("Je mange");
    }
}
```

```java
public class Chien extends Animal{
   public Chien(String nom) {
       super(nom);
   }

   public void aboyer() {
       System.out.println("Je suis" + super.nom);
       System.out.println("Wouaf wouaf !");
   }
}
```

```java
public class AppAnimal {
    public static void main(String[] args) {
        Animal a1 = new Animal("Toto");
        Animal c1 = new Chien("Fifi");
        Chien c2 = new Chien("Rex");
        a1.manger();

        c1.manger();
        c1.aboyer();

        c2 = c1;
        c2.manger();

        Chien c3 = new Animal("Loulou");
        c3.aboyer();
    }
}
```

Y a-t-il des erreurs de compilation ou exécution dans ce programme ?
Si oui, citez lesquelles et expliquez brièvement pourquoi.
Si non, donnez les résultats de l'exécution de la fonction principale.

Solution:

- L'instruction **Animal a1 = new Animal("Toto")** de la classe principale, ne compile pas car **Animal** est une classe abstraite.
- L'instruction **c1.aboyer()** ne compile pas car le type apparent de **c1** est **Animal** (la référence **c1** est de type **Animal**). Et ce, bien que l'objet référencé est un **Chien**.
- L'instruction **c2 = c1** de la classe **AppAnimal** ne compile pas car **c2** est une référence de type **Chien**, alors que **c1** est une référence de type **Animal**.
- Dans la méthode **aboyer()** de la classe **Chien**, on essaie d'accéder à **super.nom**. Or, cet attribut est privé dans **Animal**, ce qui provoque donc une erreur de compilation.
- L'instruction **Chien c3 = new Animal("Loulou")** de la classe **AppAnimal** ne compile pas car **c3** est une référence de type **Chien**.

7. Soit le code Java suivant correspondant à la modélisation orientée objet d'un réseau social.

```java
import java.util.ArrayList;

public class Utilisateur {
    private ArrayList<Utilisateur> amis;

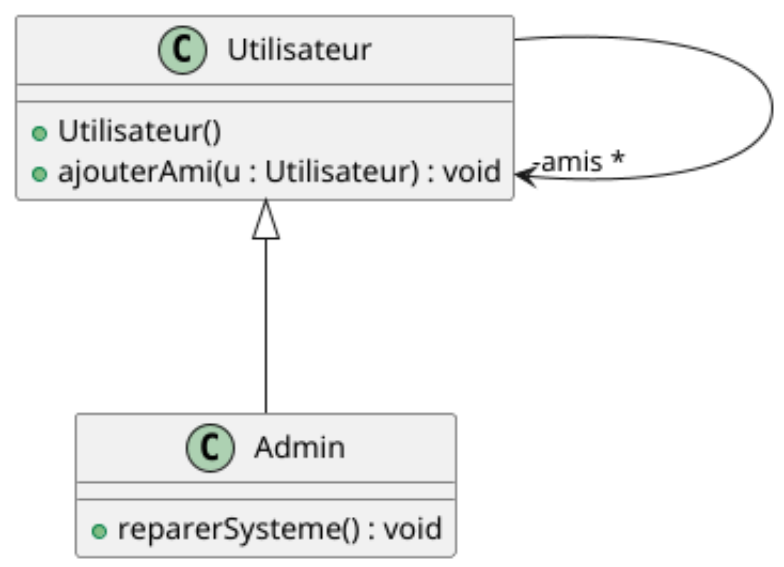
    public Utilisateur() {
        amis = new ArrayList<>();
    }

    public void ajouterAmi(Utilisateur u) {
        amis.add(u);
    }
}
```

```java
public class Admin extends Utilisateur{
    public void reparerSysteme() {
        System.out.println("Je répare le système");
    }
}
```

Dessinez un diagramme de classe correspondant à ce code

Solution:



Voici comment on note les visibilités en UML :

+ : public
- : private
: protected
~ : visibilité par défaut (ou par paquetage)

Le logiciel PlantUML les dessine sous forme de cercle vert (public), carré rouge (privé), diamant jaune (protected). Il s'agit juste d'une manière d'afficher les choses.