Rappel Séance 1 (week-end 30-31/10/2021)

1. Créer un projet java avec le nom "TPs Java"
2. Créer un package p1= “ma.education.tp1.introduction”
3. Dans le package (p1), créer une classe java **Salle** définie par un id de type **long** et un nom de type **String**.
4. Dans la classe Salle créer les trois constructeurs suivants:
   a. Un constructeur sans paramètres
   b. Un constructeur pour le nom de la salle
   c. Un constructeur pour l’id et le nom de la salle
5. Dans le package (p1), créer une classe Test qui contient la méthode main
6. **En utilisant le mot réservé new,** dans la méthode main, instancier les trois objets suivants.
   a. Un objet (o1) en utilisant le constructeur sans paramètres
   b. Un objet (o2) ayant le nom “Salle Informatique”
   c. Un objet (o3) ayant l’id 2 et le nom “Salle des cours”
7. Afficher dans la consoles les valeurs des attributs des objets o1, o2 et o3
8. Exécuter la classe Test. (Dans Eclipse, clic droit à l'intérieur de la classe Test>run as>java application)
9. Voir le résultat de l’affichage dans la console Eclipse

Séance 2 (week-end 13-14/10/2021)

Dans cette séance et la séance prochaine, nous souhaitons atteindre les objectifs suivants:

Objectif 1 : Comprendre l'héritage, le cast explicite et implicite, super et super()
Objectif 2 : Utiliser les modificateur d'accès: private, public, protected et par défaut.
Objectif 3 : Différencier entre la classe **Class** et la classe **Object** du java.
Objectif 4 : Utiliser le mot réservé “**static**”
Objectif 5 : Créer des classes selon le patron de conception : **Singleton**.

Complement Youtube de cette séance : https://youtu.be/Am44ZzPxRcc

I. L'héritage, le cast explicite et implicite, super et super()

Maintenant nous allons supposer que les salles sont de deux types : les laboratoires et les salles de cours.

1. Créer les deux classes filles Laboratoire et SalleCours de la classe mère “ma.education.tp1.introduction.Salle”.
2. La classe Laboratoire contient l’attribut type (String), la classe SalleCours contient l’attribut nombre de places(byte)

3. Mettre en commentaire le constructeur sans paramètres de la classe “ma.education.tp1.introduction.Salle”. Remarquer les erreurs de compilation dans les classes filles.

 *Les classe filles appellent par défaut le constructeur sans paramètres de leur classe mère en utilisant super()*

4. Corriger ces erreurs en donnant des arguments à super() au niveau des constructeurs des classes fille Laboratoire et SalleCours.

```
public Laboratoire(long id, String nom, String type) {
        super(id, nom);
        this.type = type;
}
public SalleCours(long id, String nom, int nbPlaces) {
        super(id, nom);
        this.nbPlaces = nbPlaces;
}
```

5. Créer la classe TestHeritage contenant la méthode main et tester les instanciations et les déclarations suivantes dans la méthode main:

```
Salle s1=new SalleCours(1, "Salle 1", 20);
SalleCours s2= new SalleCours(2, "Salle 2", 20);
SalleCours s3=s1;
SalleCours s4=s2;
```

6. Dans la question précédente, trouver la ligne qui donne une erreur de compilation et justifier pourquoi.
7. Utiliser le cast explicite pour corriger cette erreur. Justifier que le cast explicite est valide dans ce cas.
8. Tester les instanciations et les déclarations suivantes dans la méthode main:

```
Salle s5=new Laboratoire(2, "LABO", "CHIMIE");
SalleCours s6= new SalleCours(2, "Salle 2", 20);
SalleCours s7=s5;
SalleCours s8=s6;
```

9. Dans la question précédente, trouver la ligne qui donne une erreur de compilation et justifier pourquoi.
10. Serait-il possible d’utiliser un cast explicite pour corriger cette erreur? Justifier pourquoi.

 *NB: Si le cast explicite n’est pas valide, une erreur d'exécution se lance:* ***Exception in thread "main" java.lang.ClassCastException***

II. La visibilité : private, public, protected et par défaut.

1. Mettre les attributs de la classe “ma.education.tp1.introduction.Salle” publiques
   public long id;

public String nom;

2. Créer une classe AccesSamePackage contenant une méthode main.
3. Créer un objet de la classe Salle ayant le nom “Salle A” dans la méthode main de la classe AccesSamePackage. Afficher les valeurs des attributs nom et id.
4. Créer une classe AccesHorsPackage contenant une méthode main.
5. Créer un objet de la classe Salle ayant le nom “Salle B” dans la méthode main de la classe AccesHorsPackage . Afficher les valeurs des attributs nom et id.

Remarque : Les deux classes AccesSamePackage et AccesHorsPackage arrivent à afficher l’id et le nom de la classe Salle sans aucun problème.

6. Maintenant rendre les attributs de la classe “ma.education.tp1.introduction.Salle” private
   private long id;
   private String nom;
7. Remarquer que les deux classes AccesSamePackage et AccesHorsPackage n’arrivent plus à accéder aux attributs id et nom déclarés private.

private*: Si un attribut, une méthode ou un constructeur sont déclarés private, alors ils seront visibles seulement dans le bloc de leur classe.*

8. Maintenant supprimer le modificateur d’accès “private” aux attributs de la classe “ma.education.tp1.introduction.Salle”
   long id;
   String nom;
9. Remarquer que seule la classe AccesSamePackage arrive à accéder aux attributs id et nom contrairement à la classe AccesHorsPackage qui présente des erreurs de compilation. On dit que les attributs id et nom ont la visibilité par défaut.

aucune visibilité*: Si un attribut, une méthode ou un constructeur sont laissés sans visibilité, alors ils seront visibles dans toutes les classes du même package que leur classe.*

10. Maintenant déclarer les attributs de la classe “ma.education.tp1.introduction.Salle” protected
    protected long id;
    protected String nom;
11. Remarquer que la classe AccesSamePackage arrive à accéder aux attributs id et nom contrairement à la classe AccesHorsPackage . Alors la visibilité **protected** donne l’accès aux classes du même package.
12. La classe AccesHorsPackage présente toujours des erreurs de compilation. Essayer de rendre la classe AccesHorsPackage fille de la classe Salle en utilisant “extends Salle”.
13. Dans la classe AccesHorsPackage créer la méthode afficher suivante :

```
public  void afficher(Salle s) {
        System.out.println(s.id );
        System.out.println(s.nom );
        System.out.println(id );
        System.out.println(nom );
```

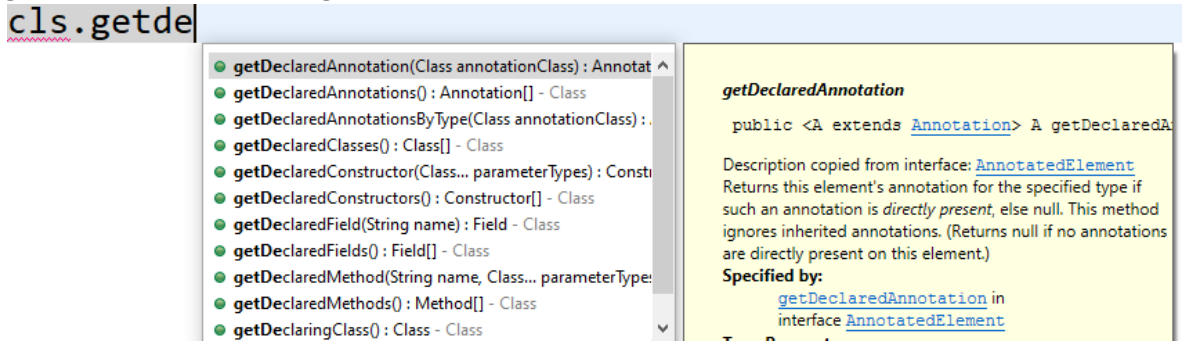
}

14. Que remarquez- vous par rapport à l'accès aux attributs déclarés **protected**?

> ***protected**: Si un attribut, une méthode ou un constructeur sont déclarés **protected**, alors ils seront visibles dans toutes les classes du même package + les classes filles qui existent hors package **(mais par l'accès se fait par héritage et non pas par référence)***

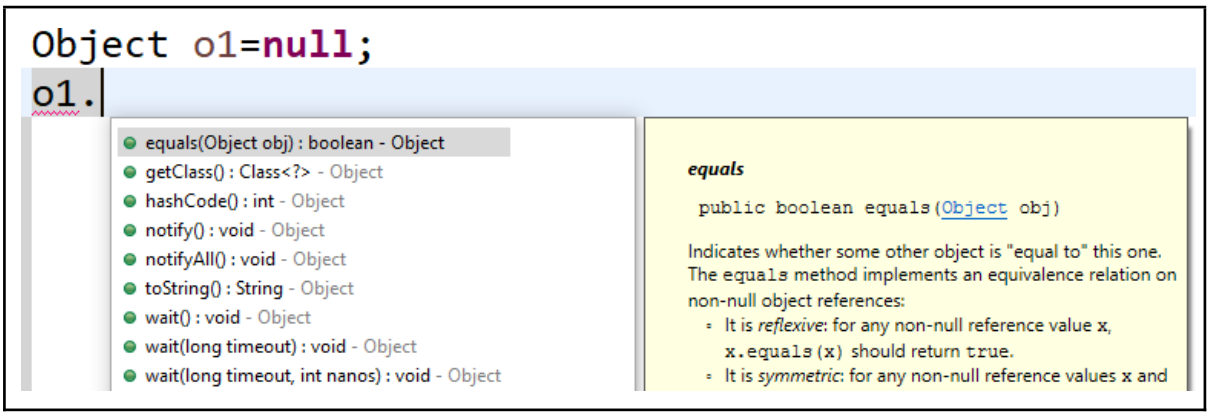
III. Différence entre la classe Class et la classe Object

1. Dans le projet "TPs Java", créer un package p2= "ma.education.tp2.reflection"
2. Dans (p2), créer une classe TestReflection contenant la méthode main.
3. Dans la méthode main, créer un programme java qui affiche les nom des attributs de la classe "ma.education.tp1.introduction.Salle"
4. Dans la méthode main de la classe TestReflection, créer un programme java qui affiche les nom des méthodes de la classe "ma.education.tp1.introduction.Salle"
5. Dans la méthode main de la classe TestReflection, **En utilisant la reflection**, instancier trois objets de la classe "ma.education.tp1.introduction.Salle"
   a. Un objet (o1) en utilisant le constructeur sans paramètres
   b. Un objet (o2) ayant le nom "Salle Informatique"
   c. Un objet (o3) ayant l'id 2 et le nom "Salle des cours"

> *la classe **Class** contient les méthodes **getDeclaredFields(), getDeclaredMethods(),getDeclaredConstructors()...** :*
>
> 
>

6. Dans la méthode main, comparer les objets o2 et o3 en utilisant la méthode equals de la classe Object. Afficher le résultat de la comparaison dans la console.
7. Recréer la méthode equals dans la classe "ma.education.tp1.introduction.Salle" pour retourner true si deux objets portent le même id.

NB: Quelques méthodes de la class Object:



IV. Le mot réservé “static”

Le mot clé static en Java est un modificateur utilisé pour économiser l’espace mémoire. Cela aide à gérer la mémoire occupée par les objets et leurs attributs. Le mot-clé static garantit qu’un seul espace mémoire sera créé pour représenter l’attribut déclaré static.

1. Créer un nouveau package p3= “ma.education.tp2.statickeyword”
2. Créer une classe nouvelle classe Etudiant avec les attributs id(long), nom (String) et nbEtudiants (int).

Pour le moment laisser la visibilité des attributs de la classe Etudiant "public".

3. D’abord Comparer les attributs de la classe Etudiant en donnant les réponses aux questions suivantes:
   a. Est ce que chaque étudiant a son propre Id? : Oui ou non
   b. Est ce que chaque étudiant a son propre nom? : Oui ou non
   c. Est ce que chaque étudiant a son propre nb Etudiant? : Oui ou non

Si la réponse est “oui” alors l’attribut est non static

Si la réponse est “non” alors l’attribut doit être static.

4. Déclarer l’attribut nbEtudiants en utilisant le mot réservé static
5. Créer un constructeur pour les trois attributs comme suivant.

```
public Etudiant (long id, String nom,int nb) {
        this.id = id;
        this.nom = nom;
        nbEtudiants+=nb;
}
```

6. Dans le package p3, créer la classe TestStatic contenant la méthode main.
7. Dans la méthode main, instancier les deux objets suivants:

```
Etudiant e1= new Etudiant(1, "AHMED", 20);
Etudiant e2= new Etudiant(2, "SAID", 30);
```

8. Afficher les attributs de (e1) et (e2)
9. Que dites vous par rapport à la valeur de l’attribut **static** nbEtudiants?

NB : Étant donné que les variables statiques appartiennent à une classe, elles seront accessibles directement à l’aide du nom de la classe et ne nécessitent aucune

référence d'objet.
Les variables statiques ne peuvent être déclarées qu'au niveau de la classe
Les variables statiques sont accessibles sans création d'objet
Une méthode statique ne peut utiliser « this. » et « super. » dans son corps.
Une méthode statique ne peut accéder ni aux variables non statiques ni aux méthodes non statiques

V. Le patron de conception "Singleton"

Le singleton est un patron de conception (design pattern) dont l'objectif est de restreindre l'instanciation d'une classe à un seul objet (ou bien à quelques objets seulement). Il est utilisé lorsqu'on a besoin exactement d'un objet pour coordonner des opérations dans un système. Le modèle est parfois utilisé pour son efficacité, lorsque le système est plus rapide ou occupe moins de mémoire avec peu d'objets qu'avec beaucoup d'objets similaires.

1. Créer un nouveau package p4= "ma.education.tp4.singleton"
2. Créer une classe Terre définie par les attributs distanceToSoleil (double) et surface (double)
3. Vu qu'il n'y a qu' une seule Terre, cette classe doit créer un seul objet dans la mémoire. Pour le faire, on doit assurer les trois règles suivantes
   a. Verrouiller le constructeur de cette classe en utilisant la visibilité "private"

```
private Terre(double distanceToSoleil, double surface) {
        this.distanceToSoleil = distanceToSoleil;
        this.surface = surface;
}
```

   b. Créer un attribut private static ayant le type Terre et le nom instance

```
private static Terre instance;
```

   c. Créer une méthode public et static ayant le type de retour Terre et le nom getInstance

```
public static Terre getInstance(double distanceToSoleil, double surface) {
        if (instance == null)
                instance = new Terre(distanceToSoleil, surface);
        return instance;
}
```

   d. Créer une classe TestSingleton avec la méthode main:
   e. Dans la méthode main créer les deux objets suivants.

```
Terre t1= Terre.getInstance(1000, 2000);
Terre t2= Terre.getInstance(4000, 5000);
```

   f. Afficher la distanceToSoleil et la surface pour l'objet (t1)
   g. Afficher la distanceToSoleil et la surface pour l'objet (t2). Quelle est votre remarque par rapport aux nombre d' objets créés dans la mémoire.

Correction du rappel du week-end 30-31/10/2021

Le code java de la classe Salle:

```
package ma.education.tp1.introduction;

public class Salle {
	/* Les attributs de la classe Salle*/
	long id;
	String nom;
	/* Le constructeur de la classe Salle
	 * Avec 2 paramètres id et nom*/
	public Salle(long id, String nom) {
		this.id = id;
		this.nom = nom;
	}
	/* Le constructeur de la classe Salle
	 * Avec un seul paramètre nom*/
	public Salle(String nom) {
		this.nom = nom;
	}
	/* Le constructeur de la classe Salle
	 * Sans paramètres*/
	public Salle() {
	}
}
```

Le code java de la classe Test:

```
package ma.education.tp1.introduction;

public class Test {

	public static void main(String[] args) {
		/*Instanciation (Création) des trois
		 * objets o1,2 and 3*/
		Salle o1=new Salle();
		Salle o2=new Salle("Salle Informatique");
		Salle o3=new Salle("Salle des cours");
		/*Affichage des valeurs des attributs des
		 * objets o1,2 and 3*/
		System.out.println(o1.id+" "+o1.nom);
		System.out.println(o2.id+" "+o2.nom);
		System.out.println(o3.id+" "+o3.nom);
	}
}
```

Correction du TP du week-end 30-31/10/2021

Différence entre la classe Class et la classe Object

```
package ma.education.tp2.reflection;

import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;

import ma.education.tp1.introduction.Salle;

public class Test {

	public static void main(String[] args)
			throws InstantiationException,
IllegalAccessException, IllegalArgumentException,
InvocationTargetException {
		/* Affichage des nom attributs */
		Class cls = Salle.class;
		Field[] attributs = cls.getDeclaredFields();
		for (Field att : attributs) {
			System.out.println(att.getName());
		}

		/* Affichage des nom methodes */
		Method[] methods = cls.getDeclaredMethods();
		for (Method meth : methods) {
			System.out.println(meth.getName());
		}

		/* Instanciation des objets par reflection */
		Salle o1 = null;
		Salle o2 = null;
		Salle o3 = null;
		Constructor[] cons = cls.getDeclaredConstructors();
		for (Constructor c : cons) {
			int count = c.getParameterCount();
			switch (count) {
			case 0: {
				o1 = (Salle) c.newInstance();
				break;
			}
			case 1: {
				o2 = (Salle) c.newInstance("Salle
Informatique");
				break;
			}
			case 2: {
				o3 = (Salle) c.newInstance(2, "Salle des
cours");
				break;
```

```
                }
                default: {
                      break;
                }
                }
          }
          /*
           * Affichage des valeurs des attributs des objets
o1,o2 et o3
           */
          System.out.println(o1.id + " " + o1.nom);
          System.out.println(o2.id + " " + o2.nom);
          System.out.println(o3.id + " " + o3.nom);
     }
}
```

L'héritage, le cast explicite et implicite, super et super()

```
/*La classe fille Laboratoire*/
package ma.education.tp2.reflection;

import ma.education.tp1.introduction.Salle;

public class Laboratoire extends Salle {
     public String type;

     public Laboratoire(long id, String nom, String type) {
          super(id, nom);
          this.type = type;
     }
}
```

```
/*La classe fille SalleCours*/
package ma.education.tp2.reflection;

import ma.education.tp1.introduction.Salle;

public class SalleCours extends Salle{
     public int nbPlaces;

     public SalleCours(long id, String nom, int nbPlaces) {
          super(id, nom);
          this.nbPlaces = nbPlaces;
     }
}
```

```
package ma.education.tp2.reflection;
```

```
import ma.education.tp1.introduction.Salle;

public class TestHeritage {
	public static void main(String[] args) {
		Salle s1=new SalleCours(1, "Salle 1", 20);
		SalleCours s2= new SalleCours(2, "Salle 2", 20);
		SalleCours s3=(SalleCours)s1;
		SalleCours s4=s2;
	}
}
```

La visibilité : public private protected et par défaut

```
package ma.education.tp1.introduction;

public class AccesSamePackage {
	public static void main(String[] args) {
		Salle s1 = new Salle("Salle A");
		System.out.println(s1.id );
		System.out.println(s1.nom );
	}
}
```

```
package ma.education.tp2.reflection;

import ma.education.tp1.introduction.Salle;

public class AccesHorsPackage extends Salle{
	public static void main(String[] args) {
		Salle s1 = new Salle("Salle B");
		System.out.println(s1.id );
		System.out.println(s1.nom );
	}
	public  void afficher(Salle s) {
		System.out.println(s.id );
		System.out.println(s.nom );
		System.out.println(id );
		System.out.println(nom );
	}
}
```

Le mot réservé static

```
package ma.education.tp3.statickeyword;
```

```
public class Etudiant {
	public long id;
	public String nom;
	public static int nbEtudiants;

	public Etudiant(long id, String nom,int nb) {
		this.id = id;
		this.nom = nom;
		nbEtudiants+=nb;
	}
}
```

```
package ma.education.tp3.statickeyword;

public class TestStatic {
	public static void main(String[] args) {
		Etudiant e1= new Etudiant(1, "AHMED", 20);
		Etudiant e2= new Etudiant(2, "SAID", 30);
		System.out.println(e1.id+" "+e1.nom+"
"+e1.nbEtudiants);
		System.out.println(e2.id+" "+e2.nom+"
"+e2.nbEtudiants);
	}
}
Resultats :
1 AHMED 50
2 SAID 50
```

Le patron de conception “Singleton”

```
package ma.education.tp2.singleton;

public class Terre {
	public double distanceToSoleil;
	public double surface;
	private static Terre instance;

	private Terre(double distanceToSoleil, double surface) {
		this.distanceToSoleil = distanceToSoleil;
		this.surface = surface;
	}

	public static Terre getInstance(double distanceToSoleil,
double surface) {
		if (instance == null)
			instance = new Terre(distanceToSoleil, surface);
```

```
            return instance;
      }
}
```

```
package ma.education.tp4.singleton;

public class TestSingleton {
      public static void main(String[] args) {
            Terre t1= Terre.getInstance(1000, 2000);
            Terre t2= Terre.getInstance(4000, 5000);
            System.out.println(t1.distanceToSoleil+"
"+t1.surface);
            System.out.println(t2.distanceToSoleil+"
"+t2.surface);

      }
}
Résultats d”affichage :
1000.0 2000.0
1000.0 2000.0
```