public class A {  
 public int a;  
 public static int *b*=1;  
 public void print(){  
 System.*out*.println("a = "+a +" "+" b= "+*b*);

}  
   
}

public static void main(String[] args) {  
 // System.out.println("Hello world!");  
 A obja=new A();  
 A objb=new A();  
 obja.a=22;  
 obja.print();//a = 22 b= 1  
 objb.print();//a = 0 b= 1

obja.a=22;  
obja.*b*=100;  
obja.print();//a = 22 b= 100  
objb.print();//a = 0 b= 100

Les attributs static sont partagés. Si un objet a change un attribut static tous les autres objets ont leur attribut static modifiés.

A.*b*=1200;  
obja.print();//a = 22 b= 1200  
objb.print();//a = 0 b= 1200

Un appel direct A.b remplace obja.b ou objb.b.

Mais A.a provoque une erreur car a n’est pas static.

**Application compter le nombre d’instance :**

public class CompteurInstance {  
 private static int *compteur*=0;  
  
 CompteurInstance(){  
 *compteur*++;  
 }  
 public void printCompteur(){  
 System.*out*.println("compteur = "+*compteur*);}  
  
}

CompteurInstance obj1=new CompteurInstance();  
CompteurInstance obj2=new CompteurInstance();  
CompteurInstance obj3=new CompteurInstance();  
  
obj1.printCompteur();// compteur = 3

*//Dans le main*CompteurInstance obj1=new CompteurInstance();  
CompteurInstance obj2=new CompteurInstance();  
CompteurInstance obj3=new CompteurInstance();  
  
obj1.printCompteur();*// compteur = 3*

*//Dans le main*CompteurInstance obj1=new CompteurInstance();  
obj1.printCompteur();*// compteur = 1*CompteurInstance obj2=new CompteurInstance();  
CompteurInstance obj3=new CompteurInstance();  
  
Des qu’on a un new , même après une instance, tous les attributs static sont mis à jour.

**Variable non inittialisé ou null**

**En java les méthodes qu’on veut appeler dans une même classe sont static. Si on fait pas de static si on appelle la methode ou fonction on aura une erreur.**

public class A {  
 public String s;  
 public int x;  
 public boolean b;  
  
  
 public A(){}  
  
 public static int fs(String s){  
 return s.length();  
 }  
  
 public static int fint(int x){  
 return x+100;  
 }  
 public static boolean fb(boolean b){  
 return b;  
 }  
}

public static void main(String[] args) {  
 System.*out*.println("Hello world!");  
  
 A a=new A();  
 String s=a.s;  
 int x=a.x;  
 boolean b=a.b;  
  
 System.*out*.println(" string s : "+s);*// affiche null* System.*out*.println(" int x : "+x); *//affiche 0* System.*out*.println(" boolean b : "+b);*//affiche false*

int x1= A.*fint*(x);  
boolean b1= A.*fb*(b);  
  
System.*out*.println(" int x1 : "+x1); *//affiche 0+100=100*System.*out*.println(" boolean b1 : "+b1);*//affiche false*int l=A.*fs*(s);// affice null pointerException

**Les Options**

class NoSuchElementException extends Exception{

 private static final long serialVersionUID = 1234567L;

   public NoSuchElementException( ){

 super();

    }

public NoSuchElementException(  String s){

 super(s);

    }

}

class Optional{

    private Object value;

    public Optional(Object v){

        this.value=v;

    }

    Object getValue()throws NoSuchElementException{

             if(this.value==null)  {

                throw new  NoSuchElementException("Ici nullpointer exception ");

             }

        return this.value;

    }

}

class Main{

    public static void main(String []args){

        System.out.println("hello !");

        A a =new A();

        Optional op=new Optional(a.nom);

        try {Object s=op.getValue();

        }

        catch(NoSuchElementException npe){

         System.out.println(" Exception  capturée ici  ! : ");

         // npe.printStackTrace();

         npe.getMessage();

        }

    }

}

**Remarque : pour utiliser notre message d’erreur on applelle getMessge() ou eprintStackTree() ;**

printStackTrace( ) affiche l'exception et l'état de la pile d'exécution au moment de son appel. Voir figure suivante :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

**Le design factory et Optional**

class Optional{

    private Object value;

    private Optional(Object v){

        this.value=v;

    }

public static Optional empty(){

    return new Optional(null);

}

public static Optional of(Object v){

    if (v == null) throw new NullPointerException(" valeur null");

    return new Optional(v);

}

public Object getValue(){

    return value;

}

}

class Main{

    public static void main(String []args){

        System.out.println("hello !");

        A a =new A();

        Optional op1=Optional.empty();

        Optional op2=Optional.of("string");

        Optional op3=Optional.of(2000);

        Object o1=op2.getValue();

          System.out.println(o1.toString());

       // Optional op4=Optional.of(op1);

        //Optional op4a=Optional.of(a.nom);

//Optional op5=Optional.of(op1.getValue());

        }

    }

Ici Optional op5=Optional.of(op1.getValue())

et Optional op4a=Optional.of(a.nom);

Vont créer une erreur donné par

public static Optional of(Object v){

    if (v == null) throw new NullPointerException(" valeur null");

    return new Optional(v);

}

On pourra ajouter dans la classe Option la methode suivante.

public static Optional ofNullable(Object v){

    Optional o=v==null ? empty():of(v);

    return o;

}

Dans ce cas l’appelle de cette methode dans le main donne :

 Optional op6 =Optional.ofNullable(3);

 System.out.println("ofNullable ! : "+ op6.getValue());//ofNullable ! : 3

 Optional op7 =Optional.ofNullable(null);

System.out.println("ofNullable ! : "+ op7.getValue());//ofNullable ! : null