LA RÉFLEXION EN JAVA

1. Définition :

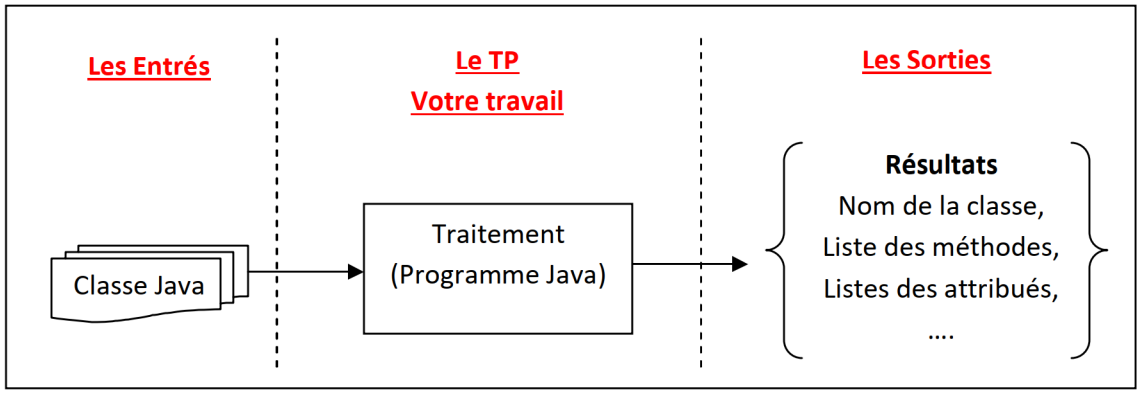
La réflexivité, aussi appelée introspection, consiste à découvrir de façon dynamique des informations relatives à une classe ou à un objet. C'est notamment utilisé au niveau de la machine virtuelle Java lors de l'exécution du programme.

2. Mise en œuvre :

Le package ***java.lang.reflect*** est consacré à la réflexion. La structure d'un objet Java est caractérisée par une classe qui contient des constructeurs, des méthodes et des champs. Les caractéristiques de l'objet sont représentés de manière formelle par une classe du package de réflexion.

Classe du package	Représente
Class	Une classe.
Method	Une méthode d'une classe
Constructor	Constructeur d'une classe
Field	Un champ d'une classe

Principe:



Exemple :

```
public class TPMaster{
    private int x;
    public TPMaster (){ }
    public void afficher(String str)   {System.out.println(str);}    }
```

La structure d'un l'objet obj (TPMaster obj= new TPMaster()) est la suivante :

1 : Un **Class** représentant la classe TPMaster.
2 : Un **Field** représentant le champ « x ».
3 : Un **Constructor** représentant le constructeur de la classe
4 : Un **Method** représentant la méthode « afficher »

Pour accéder à toute la structure d'un objet, il faut utiliser la classe « *Class* ». Ci-dessous quelques méthodes qu'on peut utiliser :

- Pour récupérer la classe d'une instance d'un objet, il faut utiliser la méthode ***getClass*** sur l'objet (Class classe=obj.**getClass**();).
- Pour récupérer une méthode, il faut utiliser ***getMethod***().
- Pour récupérer un constructeur, il faut utiliser ***getConstructor***().
- Pour récupérer un champ, il faut utiliser ***getField***().
- Pour récupérer un constructeur, le principe reste le même que pour les méthodes sauf qu'au lieu d'utiliser ***getMethod***, on utilise ***getConstructor***.

3. Exemple d'application :

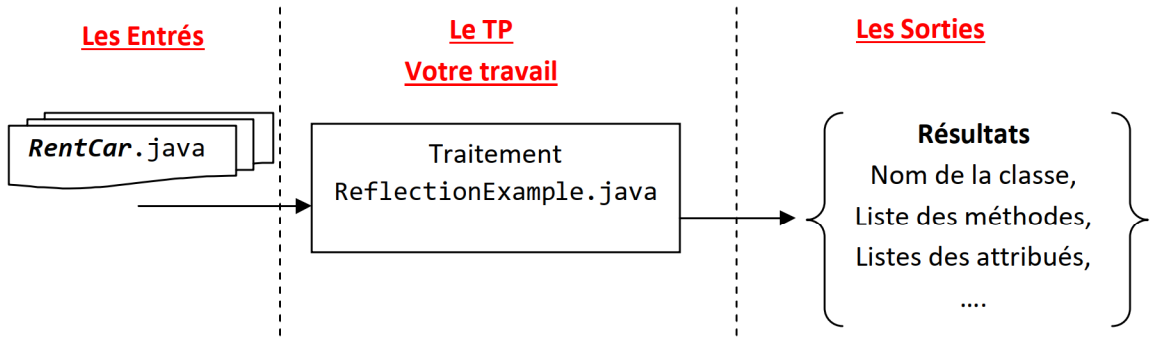
Considérons la classe Java suivante :
(Source : https://examples.javacodegeeks.com/core-java/reflection/java-reflection-example/)

```
package Pack;
import java.lang.reflect.*;
import java.util.Arrays;
```

```
class RentCar {
          private int rate;
          private String type;
          public int price;
          public RentCar(int length) {
               if (length < 455) {type = "small"; rate = 35;}
               else if ((length >= 455) && (length < 495)) {
                   type = "mid-sized";rate = 45;}
               else if (length >= 495) {type = "large";rate = 55; }
           }
          public int getRate() {return rate;}
          public String getType() {return type;}
          public void setRate(int rate) {this.rate = rate;}
          public void setType(String type) {this.type = type;}
          public void computeRentalCost(int numDays) {
          price = numDays * rate;
          System.out.println("The cost of your rental car is " + price + " euros");
}}
```

Considérons également la classe ***ReflectionExample*** qui permet d'extraire les informations suivantes à partir de la classe ***RentCar*** :

- Le nom de la clase.
- Le nom simple de la classe (sans package).
- Le nom de package
- Le/les constructeur(s) de la classe
- Un constructeur avec un argument spécifique.
- Initialisation d'un objet de la classe.
- Les méthodes de la classe.
- Les méthodes déclarées dans la classe.
- Une méthode avec un nom et des paramètres.
- Le type de retour des méthodes.
- Les paramètres des méthodes.
- Les attributs publics de la classe.
- Le nom et le type des attributs publics de la classe.
- La valeur d'un attribut public.
- La valeur de l'attribut privé.



```
public class ReflectionExample  {
public static void main(String[] args)  {
```

1. Obtenir le nom de l'objet de la classe si on connait le nom de la calsse

```
Class  rental  =  RentCar.class;
try {
```

2. Récupérer le nom de la classe

```
String  rentalClassPackage  =  rental.getName();
System.out.println("Class  Name  is:  " +  rentalClassPackage);
```

3. Récupérer le nom simple de la classe (sans package)

```
String  rentalClassNoPackage  =  rental.getSimpleName();
System.out.println("Class  Name  without  package  is:  "  +
rentalClassNoPackage);
```

4. Récupérer le nom de package

```
Package  rentalPackage  =  rental.getPackage();
System.out.println("Package  Name  is:  " +  rentalPackage);
```

5. Récupérer le/les constructeur(s) de la classe

```
Constructor[]  constructors  =  rental.getConstructors();
System.out.println("Constructors  are:  "+  Arrays.toString(constructors));
```

6. Récupérer un constructeur avec un argument spécifique.

```
Constructor  constructor  =  rental.getConstructor(Integer.TYPE);
```

7. Initialisation d'un obejt de la classe RentCar

```
RentCar  rent  =  (RentCar)  constructor.newInstance(455);
```

8. Récupérer les méthodes de la classe (avec les méthodes de la superclasse).
   superclasse de RentCar dans ce cas est la class java.lang.Object

```
Method[]  allmethods  =  rental.getMethods();
System.out.println("Methods  are:  " +  Arrays.toString(allmethods));
for (Method  method  :  allmethods)  {System.out.println("method  =  " +
method.getName());}
```

9. Récupérer les méthodes déclarées dans la classe (sans inherited methods)

```
Method[]  declaredMethods  =  rental.getDeclaredMethods();
System.out.println("Declared  Methods  are:  "+
Arrays.toString(declaredMethods));
for (Method  dmethod  :  declaredMethods)  {System.out.println("method  =  " +
dmethod.getName());}
```

10. Récupérer une méthode avec un nom et des paramètres

```
Method  oneMethod  =  rental.getMethod("computeRentalCost",new Class[]  {
Integer.TYPE  });
System.out.println("Method  is:  " +  oneMethod);
```

11. Appel de la méthode computeRentalCost avec un paramètre de type int

```
oneMethod.invoke(rent, 4);
```

12. Récupérer les parametres de computeRentalCost

```
Class[]  parameterTypes  =  oneMethod.getParameterTypes();
System.out.println("Parameter  types  of  computeRentalCost()  are:  "+
Arrays.toString(parameterTypes));
```

13. Récupérer le type de retour de computeRentalCost

```
Class returnType = oneMethod.getReturnType();
System.out.println("Return type is: " + returnType);
```

14. Récupérer les attributs publics de la classe RentCar

```
Field[] fields = rental.getFields();
System.out.println("Public Fields are: ");
for (Field oneField : fields) {
```

15. Récupérer le nom d'un attribut public

```
    Field field = rental.getField(oneField.getName());
    String fieldname = field.getName();
    System.out.println("Fieldname is: " + fieldname);
```

16. Récupérer le type d'un attribut public

```
    Object fieldType = field.getType();
    System.out.println("Type of field " + fieldname + " is: "+ fieldType);
```

17. Récupérer la valeur d'un attribut public

```
    Object value = field.get(rent);
System.out.println("Value of field " + fieldname + " is: " + value);
}
```

18. Pour accéder à des attribut privé de la classe, on utilise la méthode getDeclaredField().

```
Field privateField = RentCar.class.getDeclaredField("type");
String name = privateField.getName();
System.out.println("One private Fieldname is: " + name);
```

19. Rendre l'instance de l'attribut privé accessible (uniquement pour la réflexion, pas pour le code normal).

```
privateField.setAccessible(true);
```

20. Récupérer la valeur de l'attribut privé

```
String fieldValue = (String) privateField.get(rent);
System.out.println("fieldValue = " + fieldValue);}
```

21. La gestion des exceptions

```
 catch (NoSuchFieldException e) {e.printStackTrace();}
 catch (NoSuchMethodException e) {e.printStackTrace();}
 catch (IllegalArgumentException e) {e.printStackTrace();}
 catch (IllegalAccessException e) {e.printStackTrace();}
 catch (InstantiationException e) {e.printStackTrace();}
 catch (InvocationTargetException e) {e.printStackTrace();}
 catch (Exception e) {e.printStackTrace();}
}}
```

Résultat de l'exécution:

```
1. Class Name is: Pack.RentCar
2. Class Name without package is: RentCar
3. Package Name is: package Pack
4. Constructors are: [public Pack.RentCar(int)]
5. Methods are: [public java.lang.String Pack.RentCar.getType(), public void
   Pack.RentCar.computeRentalCost(int), public int Pack.RentCar.getRate(),
   public void Pack.RentCar.setType(java.lang.String), public void
```

```
   Pack.RentCar.setRate(int), public final void java.lang.Object.wait() throws
   java.lang.InterruptedException, public final void
   java.lang.Object.wait(long,int) throws java.lang.InterruptedException,
   public final native void java.lang.Object.wait(long) throws
   java.lang.InterruptedException, public boolean
   java.lang.Object.equals(java.lang.Object), public java.lang.String
   java.lang.Object.toString(), public native int java.lang.Object.hashCode(),
   public final native java.lang.Class java.lang.Object.getClass(), public
   final native void java.lang.Object.notify(), public final native void
   java.lang.Object.notifyAll()]
6. method = getType
7. method = computeRentalCost
8. method = getRate
9. method = setType
10. method = setRate
11. method = wait
12. method = wait
13. method = wait
14. method = equals
15. method = toString
16. method = hashCode
17. method = getClass
18. method = notify
19. method = notifyAll
20. Declared Methods are: [public java.lang.String Pack.RentCar.getType(),
   public void Pack.RentCar.computeRentalCost(int), public int
   Pack.RentCar.getRate(), public void Pack.RentCar.setType(java.lang.String),
   public void Pack.RentCar.setRate(int)]
21. method = getType
22. method = computeRentalCost
23. method = getRate
24. method = setType
25. method = setRate
26. Method is: public void Pack.RentCar.computeRentalCost(int)
27. The cost of your rental car is 180 euros
28. Parameter types of computeRentalCost() are: [int]
29. Return type is: void
30. Public Fields are:
31. Fieldname is: price
32. Type of field price is: int
33. Value of field price is: 180
34. One private Fieldname is: type
35. fieldValue = mid-sized
```

Source : https://examples.javacodegeeks.com/core-java/reflection/java-reflection-example/