

## TP1 : Trace.py

```
def trace(f):
    def f_interne(*n):
        a = f.__qualname__
        print('--> ' + a + '(' + str(*n) + ')')
        resultat = f(*n)
        print('<--' + str(resultat))
        return(resultat)

    return f_interne
```

```
@trace
def fact(n):
    if n <= 1:
        return 1
    else:
        return n * fact(n - 1)
```

```
@trace
def est_pair(n):
    return n == 0 or est_impair(n - 1)
```

```
@trace
def est_impair(n):
    return n > 0 and est_pair(n - 1)
```

```
def main():
    x = 3
    print(f'fact({x}) =', fact(3))

    print(f'{x} est', 'pair' if est_pair(x) else
'impair')
```

```
if __name__ == '__main__':
    main()
```

## TP1 : fn\_bavard.py

```
import functools
def fn_bavard(f):
    @functools.wraps(f)
    def f_interne(*p, **k):
        print('debut de f_interne()')
        f(*p, **k)
        print('fin de f_interne()')
    print('dans fn_bavard')
    return f_interne
```

```
@fn_bavard
def exemple(x, y='ok'):
    print('exemple:', y, x)

print('Appel à exemple')
exemple('?')
print(exemple.__qualname__)
```

Un **décorateur** est une fonction qui modifie le comportement d'autres fonctions.

Les **décorateurs** sont utiles lorsque l'on veut ajouter du même code à plusieurs fonctions existantes.

## TP2 : Introduction

L'inversion de contrôle (inversion of control, IoC) est un patron d'architecture commun à tous les frameworks (ou cadre de développement et d'exécution). Il fonctionne selon le principe que

le flot d'exécution d'un logiciel n'est plus sous le contrôle direct de l'application elle-même mais du framework ou de la couche logicielle sous-jacente. Un proxy (on parle aussi de procuration ou mandataire) est un objet qui est fourni à l'utilisateur à la place de l'objet réel qu'il attend.

Le proxy sera donc un intermédiaire entre l'utilisateur et l'objet réel. Un proxy ajoute souvent des propriétés non fonctionnelles comme par exemple

## TP2 : Analyseur.java

```
import java.io.*;
import java.util.*;

/** Analyser des données d'un fichier, une donnée par ligne avec 4 informations
 * séparées par des blancs : x, y, ordre (ignore), valeur.
 */
public class Analyseur {
    /** Conserve la somme des valeurs associées à une position. */
    private Map<Position, Double> cumul;

    /** Construire un analyseur vide. */
    public Analyseur() {
        cumul = new HashMap<>();
    }

    public void update(Map<Position, Double> tmp) {
        tmp.forEach((key, value) -> {
            cumul.put(key, valeur(key) + value);
        });
    }

    /** Charger l'analyseur avec les données du fichier "donnees.java". */
    public void charger(String filename) throws MalformedURLException {
        try (BufferedReader in = new BufferedReader(new FileReader(filename))) {
            String ligne = null;
            Map<Position, Double> tmp = new HashMap<>();
            while ((ligne = in.readLine()) != null) {
                String[] mots;
                if (filename.contains("csv")) {
                    mots = ligne.split("(?=(?:[^\"]*" + "[^\"]*" + "[^\"]*" + "$)", -1);
                } else {
                    mots = ligne.split("\\s+");
                }
                assert mots.length == 4; // 4 mots sur chaque ligne
                int x = Integer.parseInt(mots[0]);
                int y = Integer.parseInt(mots[1]);
                if (x < 0 || y < 0) {
                    tmp = new HashMap<>();
                    throw new MalformedURLException(filename);
                }
                Position p = new Position(x, y);
                double valeur;
                if (filename.contains("-f2.txt") || filename.contains("-f2.csv")) {
                    valeur = Double.parseDouble(mots[4]);
                }
            }
        }
    }
}
```

## TP2 : Analyseur.java (Suite)

```
        } else {
            valeur = Double.parseDouble(mots[3]);
        }
        tmp.put(p, valeur(p) + valeur);
        // p.setY(p.getY() + 1);    // p.y += 1;
    }
    update(tmp);
} catch (IOException e) {
    throw new RuntimeException(e);
}
}

/** Obtenir la valeur associée à une position. */
public double valeur(Position position) {
    Double valeur = cumul.get(position);
    return valeur == null ? 0.0 : valeur;
}

/** Obtenir toutes les données. */
public Map<Position, Double> donnees() {
    return Collections.unmodifiableMap(this.cumul);
}

/** Afficher les données. */
public static void main(String[] args) {
    try {
        int i;
        Analyseur a = new Analyseur();
        if (args.length == 0) {
            System.out.println("Usage: Analyseur filename..*");
            System.exit(1);
        }
        for (i = 0 ; i < args.length ; i++) {
            a.charger(args[i]);
        }
        System.out.println(a.donnees());
        System.out.println("Nombres de positions : " +
                           a.donnees().size());
    } catch (MalformedURLException e) { }
}

}
```

## TP2 : Analyseur.py

```
import sys
import csv
class Analyseur:

    ''' Conserver des statistiques sur un jeu de données. '''
    def __init__(self):
        self.__cumuls = {}

    def charger(self, filename):
        with open(filename, 'r') as entree:
            for ligne in entree:
                if ".csv" in filename:
                    mots = list(csv.reader(entree))
                else:
                    mots = ligne.split()
                    x = int(mots[0])
                    y = int(mots[1])
                    p = (x, y)
                    if "-f2.txt" in filename or "-f2.csv" in filename:
                        v = float(mots[4])
                    else:
                        v = float(mots[-1])
                    self.__cumuls[p] = self.cumul(p) + v

    def cumul(self, position):
        return self.__cumuls.get(position, 0)

    @property
    def cumuls(self):
        return dict(self.__cumuls)

    def __str__(self):
        return str(len(self.__cumuls)) + ' données ' + ', '.join(
            '[(x={},y={}) : cumul={}]'.format(p[0], p[1], cumul)
            for p, cumul in self.__cumuls.items())

def main():
    a = Analyseur()
    for i in range(1, len(sys.argv)):
        a.charger(sys.argv[i])
    print('Statistiques :', a)
    print('Cumuls :', a.cumuls)
    print('Nombre de positions :', len(a.cumuls))

if __name__ == '__main__':
    if (len(sys.argv) == 1):
        print("Usage: python3 analyseur.py filename..*")
        exit(1)
    main()
```

### TP3 : List.java

```
public class UnmodifiableList {
    public static void main(String[] args) {
        List<Integer> list = new ArrayList<Integer>();
        System.out.println("Create an empty list : " + list);
        list.addAll(List.of(2, 3, 5, 7));
        System.out.println("Fill the list : " + list);
        list.remove(list.size() - 1);
        System.out.println("Remove 7 from the list : " + list);

        ***** Immutable List *****
        List<Integer> immutablelist = Collections.unmodifiableList(list);
        System.out.println("Create an immutablelist : " + immutablelist);

        try {
            System.out.println("Trying to add 10 to the unmodifiablelist");
            immutablelist.add(10);
        } catch (UnsupportedOperationException e) {
            System.out.println("Exception thrown : immutablelist is in " +
                "read-only, editing is Unsupported.");
        }

        try {
            System.out.println("Trying to remove 10 to the unmodifiablelist");
            immutablelist.remove(immutablelist.size() - 1);
        } catch (UnsupportedOperationException e) {
            System.out.println("Exception thrown : immutablelist is in " +
                "read-only, removing is unsupported.");
        }

        ***** proxy List *****
        String[] methodNames = {"add", "remove"};
        ProtectionHandler proxy = new ProtectionHandler(list, methodNames);
        System.out.println("Create a proxy object of list : " + list);

        if (rand.nextInt(1000) % 2 == 0) {
            proxy.invoke(list, ArrayList.class.getMethod("add", new Class[]{Object.class}), new
                Integer[]{list.size() - 1});
        }

        else {
            proxy.invoke(list, List.class.getMethod("remove", new Class[]{int.class}), new
                Integer[] {list.size() - 1});
        }
    }
}
```

### TP3 : Introduction

Java propose une classe *Proxy* qui s'appuie sur l'introspection pour définir des proxys : tous les appels de méthodes seront traités par un objet réalisant l'interface *InvocationHandler* qui réifie l'appel d'une méthode sous la forme d'une méthode *invoke* qui prend en paramètre la méthode appelée et ses paramètres effectifs. Un paramètre supplémentaire permet d'obtenir le proxy à l'origine de l'appel. La création effective du proxy se fait grâce à la classe *Proxy*. Elle permet de créer dynamiquement un proxy grâce à sa fabrique statique *newProxyInstance* qui prend en paramètre le chargeur de classe à utiliser (on prend celui qui a permis de charger la classe de l'objet qui nous intéresse, par exemple *List.class.getClassLoader()*), les interfaces que ce proxy réalisera (objet de type *Class*) et l'instance de *InvocationHandler* à utiliser pour traiter les appels de méthodes.

### TP3 : ProtectionHandler.java

```
import java.lang.reflect.InvocationHandler;
import java.lang.reflect.Method;
import java.util.*;

/**
 * ProtectionHandler implements InvocationHandler
 * @author Hamza Mouddene
 * @version 1.0
 */
public class ProtectionHandler implements InvocationHandler {
    /** Attributes of ProtectionHandler */
    private Object object;
    private String[] methodNames;

    /**
     * Constructio of ProtectionHandler.
     * @param object we will handle.
     * @param methodNames we will disable for object.
     */
    public ProtectionHandler(Object object, String[] methodNames) {
        this.object = object;
        this.methodNames = methodNames;
    }

    @Override
    public Object invoke(Object arg0, Method arg1, Object[] arg2) throws Throwable {
        if (Arrays.asList(this.methodNames).contains(arg1.getName())) {
            System.out.println("The proxy doesn't support " + arg1.getName() + "
                                method.");
            throw new UnsupportedOperationException();
        }
        return arg1.invoke(this.object, arg2);
    }
}
```

### TP3 : MapProxy.java

```
import java.util.*;

public class MapProxy {

    public static void main(String args[]) throws NoSuchMethodException,
    SecurityException,
        Throwable {
        Random rand = new Random();
        Map<String, Integer> map = new HashMap<String, Integer>();
        System.out.println("Create an empty map : " + map);
        map.put("a", 2);
        map.put("b", 3);
        map.put("c", 5);
        map.put("d", 7);
        System.out.println("Fill the map : " + map);
        map.remove("d");
        System.out.println("Remove the key d (value = 7), map : " + map);

        String[] methodNames = {"put", "clear"};
        ProtectionHandler proxy = new ProtectionHandler(map,
        methodNames);
        System.out.println("Create a proxy object of map : " + map);

        if (rand.nextInt(1000) % 2 == 0) {
            proxy.invoke(map, Map.class.getMethod("put", new Class[]
            {Object.class,                                Object.class}),          new
            Object[]{"c", 5});
        }

        else {
            proxy.invoke(map, Map.class.getMethod("clear", new Class[]{}),
            new Integer[]{});
        }

    }
}
```

### TP3 : Proxy.py

```
class C:  
    pass
```

```
class Proxy:
```

```
    def __init__(self, object, method_names):  
        self.object = object  
        self.method_names = method_names
```

```
    # Instance method
```

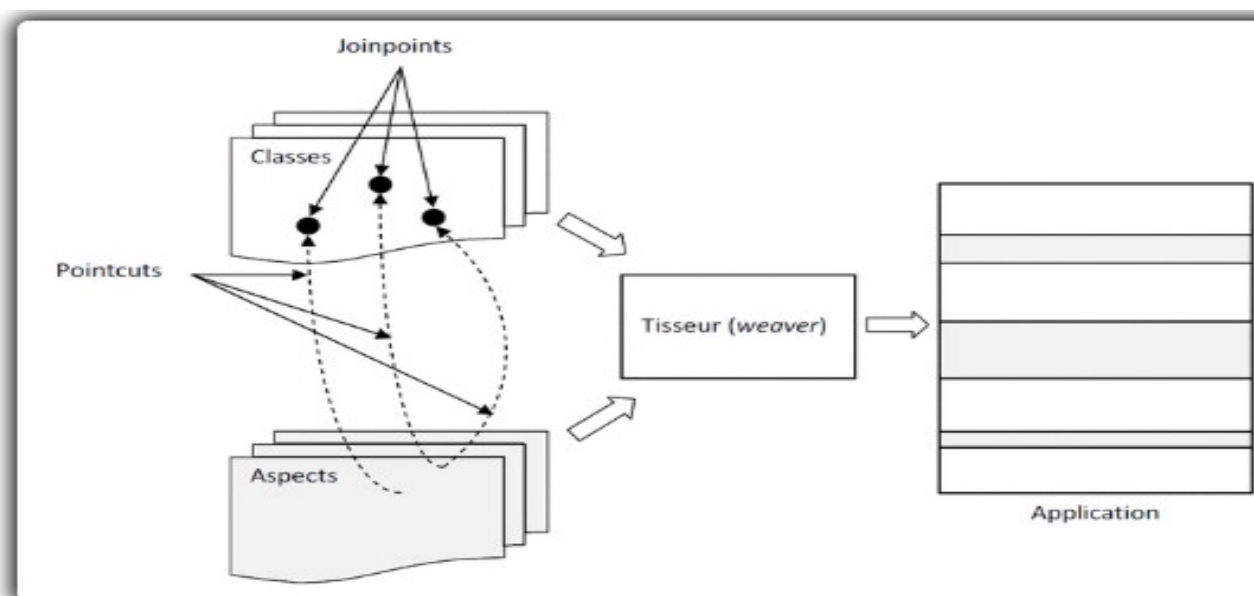
```
    def invoke(self, arg0, arg1, arg2):  
        if (arg1 in self.method_names):  
            raise AttributeError("The proxy doesn't support %s  
method.",  
                                arg1.__name__)  
        return getattr(C, arg1 + arg2)(arg0)
```

```
c = C()  
proxy = Proxy(c, dir(c))  
c.x = 5  
assert c.x == 5  
proxy.invoke(c, 'hasattr', "(c, 'x')")  
assert hasattr(c, 'x')  
assert getattr(c, 'x') == 5  
assert not hasattr(c, "y")  
setattr(c, 'y', 7)  
assert hasattr(c, "y")  
assert getattr(c, 'y') == 7  
assert c.y == 7  
assert vars(c) == {'x': 5, 'y': 7}  
delattr(c, 'x')  
assert not hasattr(c, 'x')  
assert vars(c) == {'y': 7}
```



## TP4 : Programmation par Aspect

Aspect : est une entité logicielle qui capture une fonctionnalité transversale à une application.	Point de jonction (joinpoint) : est un point dans le flot de contrôle d'un programme dans lequel un ou plusieurs aspects peuvent être appliqués.
Coupe (pointcut) : sélectionne un ensemble de points de jonction.	Code advice (advice code) : est un bloc de code définissant le comportement d'un aspect. AspectJ offre trois types de codes advice  : <i>before</i> , <i>after</i> et <i>around</i> .
Mécanisme d'introduction : est un mécanisme d'extension permettant d'introduire de nouveaux éléments structuraux au code d'une application.	Tissage (weaving) : est le processus qui prend en entrée un ensemble d'aspects et une application de base et fournit en sortie une application dont le comportement et la structure sont étendus par les aspects.



*Tissage des aspects*

## TP4 : Programmation par Aspect

### \*\*\*\*\*Pointcut\*\*\*\*\*

```
pointcut publicMethod():
call(void CompteCourant.*(double)) ||
call(void CompteCourant.debiter(double)) ||
call(void CompteCourant.editerReleve()) ||
call(String CompteCourant.toString()) ||
call (String CompteSimple.getNumero()) ||
call (double CompteSimple.getSolde()) ||
call (Personne CompteSimple.getTitulaire()) ||
call(void CompteSimple.crediter(double)) ||
call(void CompteSimple.debiter(double)) ||
call(String CompteSimple.toString())
/**
pointcut publicMethod():
call(public CompteCourant.*(..)) ||
call (public CompteSimple.*(..))
*/
```

### \*\*\*\*\*advice\*\*\*\*\*

```
before() : publicMethod() {
    System.out.println(thisJoinPoint.getSignature() + " " + t
        hisJoinPoint.getArgs());
}
```

### \*\*\*\*\*Aspect\*\*\*\*\*

```
aspect DebitEleve {
    public static LIMITE = 450;

    before(double val) : debiter: (call(void *.debiter(double))) && args(val) {
        call (void DebitEleve.func(val))
    }

    public void func(double somme) {
        If (somme > LIMITE) {
            System.out.println("send a notification in process ...")
        }
    }
}
```

## TP4 : Observer pattern

```
public class Subject {*****
    private List<Observer> observers = new ArrayList<Observer>();
    private int state;
    public int getState() {
        return state;
    }
    public void setState(int state) {
        this.state = state;
        notifyAllObservers();
    }
    public void attach(Observer observer){
        observers.add(observer);
    }
    public void notifyAllObservers(){
        for (Observer observer : observers) {
            observer.update();
        }
    }
}*****

public abstract class Observer {
    protected Subject subject;
    public abstract void update();
}*****

public class BinaryObserver extends Observer{
    public BinaryObserver(Subject subject){
        this.subject = subject;
        this.subject.attach(this);
    }
    @Override
    public void update() {
        System.out.println( "Binary String: " + Integer.toBinaryString(
            subject.getState() ) );
    }
}*****
```

```

public class OctalObserver extends Observer{
    public OctalObserver(Subject subject){
        this.subject = subject;
        this.subject.attach(this);
    }
    @Override
    public void update() {
        System.out.println( "Octal String: " +
            Integer.toOctalString( subject.getState() ) );
    }
}*****
public class HexaObserver extends Observer{
    public HexaObserver(Subject subject){
        this.subject = subject;
        this.subject.attach(this);
    }
    @Override
    public void update() {
        System.out.println( "Hex String: " +
            Integer.toHexString( subject.getState() ).toUpperCase() );
    }
}*****
public class ObserverPatternDemo {
    public static void main(String[] args) {
        Subject subject = new Subject();
        new HexaObserver(subject);
        new OctalObserver(subject);
        new BinaryObserver(subject);
        System.out.println("First state change: 15");
        subject.setState(15);
        System.out.println("Second state change: 10");
        subject.setState(10);
    }
}

```

```

Output : ***First : state change: 15 \n Hex String: F \n Octal String:
17 \n Binary String: 1111 \n Second state change: 10 \n Hex String:
A \n Octal String: 12 \n Binary String: 1010

```