JAVA (avancé)

https://arbimo.github.io/insa-4ir-advanced-prog/

Remise a niveau

pour faire une constante :

```
final int unNombreConstant = 3;
```

code minimale :

```
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello World !");  // how to fill a tuto
    }
}
```

le "static" correspond a une variable globale a toutes les instanciation de class, a utiliser par principe sur les fonctions.

- public : visible pour tous et par conséquent le moins restrictif ;
- protected (protégé) : visible pour le package et l'ensemble de ses sous-classes ;
- package-protected (protégé par paquet) : généralement visible uniquement par le package dans lequel il se trouve (paramètres par défaut). Ne pas mettre de mot clé déclenche ce niveau de contrôle ;
- private (privé) : accessible uniquement dans le contexte dans lequel les variables sont définies (à l'intérieur de la classe dans laquelle il est situé).
- default : visible depuis le package

créer une liste :

```
List<String> nom = Arrays.asList("Baptiste", "Leandro", "Raph");
```

TD

TD1

thread :

```
private static class TickThread extends Thread {
    private final AtomicBoolean asToStop = new AtomicBoolean(false);

    public void stopThread() {
        this.asToStop.set(true);
    }

    @Override
    public void run() {
        log("Starting...");
        while(!this.asToStop.get()) {
            try {
                System.out.println("Tick");

                this.sleep(1000);

            } catch (InterruptedException e) {
                System.out.println("Tick interrupted");
            }
        }
    }
}
public static void main(String[] args) {
    TickThread t = new TickThread();
        t.setName("Tick-thread");
        t.start();
        t.stopThread();
}
```

de manière synchronisé avec timer :

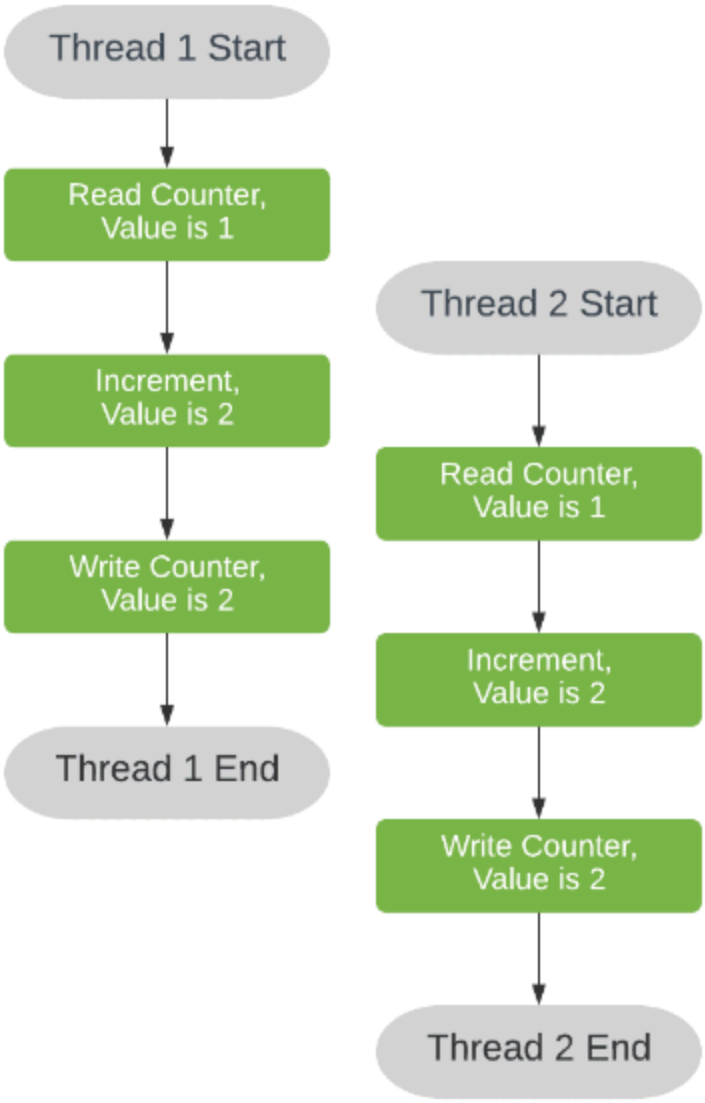
```
 public static class TimerThread extends Thread {

        public TimerThread(String name) {
                this.setName(name);
        }

        @Override
        public void run() {
                Timer timer = new Timer();
                timer.schedule(new TimerTask() {
                        @Override
                        public void run() {
                                log("tick " + seconds);
                                incrementSeconds();
```

```
                        }
                }, 0,1000);
        }
}
```

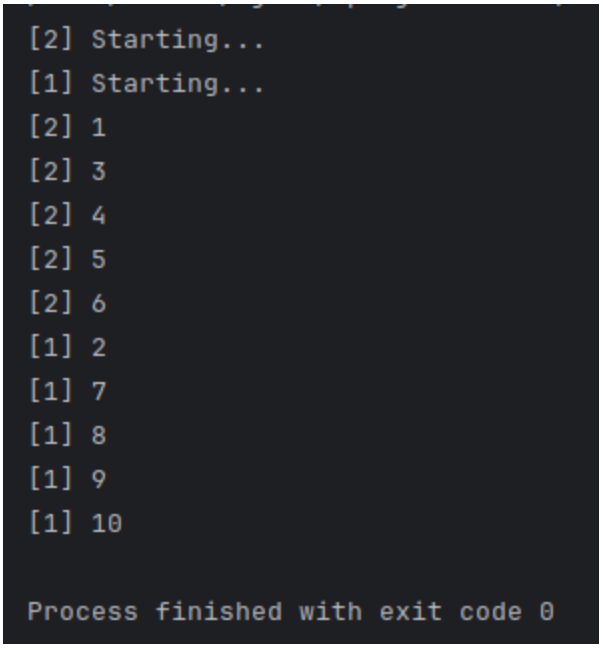
Race condition : https://www.baeldung.com/java-synchronized



t1 et t2 sont des thread qui ont une boucle qui incremente un compteur commun

```
counterThread t1 = new counterThread();
counterThread t2 = new counterThread();
t1.setName("1");
t1.start();
t2.setName("2");
t2.start();
```

résultat :



pour synchroniser un élément :

```
synchronized (this) {
        //instructions
}
```

pour synchroniser une méthode :

```
private synchronized int incr() {
    counter++;
    return counter;
}
```

TD2 : Network

permet d'emettre et recevoir des packet (sorte de ping amélioré) :

```
import java.io.IOException;
import java.net.*;

public class Net {

    public static void main(String[] args) throws IOException {
        DatagramSocket socket = new DatagramSocket(12345);
        boolean running = true;
        byte[] buf = new byte[256];
        System.out.println("Listening on port 12345...");
```

```
        while (running) {
            DatagramPacket inPacket  = new DatagramPacket(buf, buf.length);
            socket.receive(inPacket);
            String received = new String(inPacket.getData(), 0,
inPacket.getLength());
            System.out.println("Received: " + received);

            if (received.equals("end")) {
                running = false;
                continue;
            }

            InetAddress senderAddress = inPacket.getAddress();
            int senderPort = inPacket.getPort();
            DatagramPacket outPacket = new DatagramPacket(buf, buf.length,
senderAddress, senderPort);
            socket.send(outPacket);
        }
        socket.close();
        System.out.println("end");
    }
}
```

server

```
import java.io.IOException;
import java.net.*;

public class Net {
    public static void main(String[] args) throws IOException {
        DatagramSocket socket = new DatagramSocket(12345);
        boolean running = true;
        byte[] buf = new byte[256];
        System.out.println("Listening on port 12345...");

        while (running) {
            DatagramPacket inPacket  = new DatagramPacket(buf, buf.length);
            socket.receive(inPacket);
            String received = new String(inPacket.getData(), 0,
inPacket.getLength());

            if (received.equals("stop")) {
                running = false;
                continue;
```

```
            } else {
                System.out.println("Greetings, " + received);
            }
        }
        socket.close();
        System.out.println("end");
    }
}
```

client

```
import java.io.IOException;
import java.net.*;
import java.nio.charset.StandardCharsets;

public class Client {
    static byte[] buf = new byte[256];

    private static void setMsg(String msg) {
       buf = msg.getBytes();
    }

    public static void main(String[] args) throws IOException {
        DatagramSocket socket = new DatagramSocket(12346);
        boolean running = true;

        setMsg("Bjr");

        while (running) {
            DatagramPacket inPacket  = new DatagramPacket(buf, buf.length);
            InetAddress senderAddress = InetAddress.getByName("127.0.0.1");
            int senderPort = 12345;
            DatagramPacket outPacket = new DatagramPacket(buf, buf.length,
senderAddress, senderPort);
            socket.send(outPacket);
        }
        socket.close();

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

TCP

```java
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.*;
import java.io.IOException;

public class tcp {
    public static class EchoMultiServer {
        private ServerSocket serverSocket;

        public void start(int port) throws IOException {
            serverSocket = new ServerSocket(port);
            while (true) {
                new EchoClientHandler(serverSocket.accept()).start();
            }
        }

        public void stop() throws IOException {
            serverSocket.close();
        }

        private static class EchoClientHandler extends Thread {
            private Socket clientSocket;
            private PrintWriter out;
            private BufferedReader in;

            public EchoClientHandler(Socket socket) throws IOException {
                this.clientSocket = socket;
            }

            public void run() {
                try {
                    out = new PrintWriter(clientSocket.getOutputStream(),
true);
                    in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
                    String inputLine;
                    while ((inputLine = in.readLine()) != null) {
                        if (".".equals(inputLine)) {
                            out.println("bye");
                            break;
                        }
                        out.println(inputLine);
                        System.out.println(inputLine);
                    }
```

```
                    in.close();
                    out.close();
                    clientSocket.close();
                } catch (IOException e) {
                    throw new RuntimeException(e);
                }
            }
        }
    }

    public static void main(String[] args) throws IOException {
        EchoMultiServer server = new EchoMultiServer();
        server.start(25565);
    }
}
```

```
# dans bash et pas SH hein !
echo "bonjour" > /dev/tcp/127.0.0.1/25565
# ou
nc 127.0.0.1 25565
```

TD3

CM1 (langage JAVA)

Override des class par défaut

N'importe quelle classe est "extend" de la classe "Objet" (qui contient des classes tel que toString).

```
public class MyInt {
        public final int n;

        MyInteger(int n) {
                this.n = n;
        }

        @Override
        public string toString() {
                return this.n.toString();
        }
}

MyInt a = new MyInt(1);
```

```
MyInt b = new MyInt(2);

// without toString
System.out.println(a) // MyInt@8efb846
System.out.println(b) // MyInt@2a84aee7

// with toString
System.out.println(a) // 1
System.out.println(b) // 2
```

boxing

```
int n = 7;
print(n);              // impossible par défaut car 7 n'est pas un objet

// le compilateur fait ça en réalité :
int n = 7;
Integer nBoxed = new Integer(n);
print(nBoxed)
```

stdlib

- Object, String, Integer
- concurrency : Thread, locks, atomics
- collections : lists, sets, maps
- input/output : files
- networking : TCP, UDP
- GUI

set : une collection qui ne contient aucun élément dupliqué :

```
insert(T t)
contains(T t)
remove(T t)
```

HashSet<T> : une des implémentation du Set<T>

```
//MyInt peux être n'importe quoi, c'est générique
HashSet<MyInt> nawak = new HashSet<>();

nawak.insert(3);
nawak.insert(4);
```

```
if(nawak.contains(3)) {...} // True
nawak.remove(3);
if(nawak.contains(3)) {...} // False
```

CM2 (compilateur)

JVM : Java Virtual Machine

Compiler du JAVA

javac : permet la création d'un .class pour chacune des classes java, certains projets vont produire un .jar
-> chaque fichier de class contient du "**java bytecode**" qui est une sorte d'asm de haut niveau proche de java.

pour démarrer le programme :

```
#json est la classe à lancer et le . l'emplacement des classes
java json --class-path .
```

> "ClassNotFoundException" est une erreur qui indique qu'une classe n'a pas été trouvé.

Interpreteur et compilateur JIT

-> "**java bytecode**" est interprété
-> "**JIT**" vient compiler les fonctions qui sont régulièrement appelées (par ex à partir d'un certains nombres d'appel)

JIT
-> compile des fonctions de manière prédictive, si par exemple la fonction prend souvent des objets en entrée d'un certains types, cette fonction va être compilé dans ce cas et interprété si ce type change(selon la fréquence peut être dé-compilé puis recompilé).

Language	Pre-compiled	Interpreted	Interpreted + JIT Compiled
Java			X
C / C++	X		
Python		X	
Javascript			X
Rust	X		
Go	X		
Bash		X	

Garbage Collection

cf slides

CM3 (Processus de développement logiciel)

phases de création d'un logiciel : (outdated)

1. project gathering
2. planning
3. design
4. implementation
5. testing
6. deployment
7. maintenance
   -> projet monolithique, produit par une seule entreprise avec une version seulement à chaque fois (systèmes de release : une version par an par ex.).

aujourd'hui :

- la complexité a augmenté avec l'ajout des librairies (beaucoup plus d'open-source)
- amélioration de la distribution du logiciel : paper -> CD-ROM -> Network

-> On créer des modules qui font des choses spécifiques : micro-services(large-scale), librairies(medium scale), small packages, classes et fonctions...

processus de création aujourd'hui :

1. identifier les responsabilités de chaque modules (class/package/librairy) -> pas d'overlap

2. documenter le contrat du module (module's contract)
3. cacher les détails des implémentations : private/public

Gestion des erreurs

Solution de l'étudiant : (aka : "Make the Compiler SHUT UP Approach")

```
try {
        // code
} catch (IOException e) {
        //tkt frérot c'est kantique
}
```

3 possibilités :

1. Gérer le soucis localement (local handling) :

```
static String DEFAULT_CONFIG = "address=localhost; port=80";

static Config loadConfig() {
        String config = null;
        try {
                config = readFile("~/.conf");
        } catch (UnreadableFile e) {
                //
                System.out.println("Could not read config file "+e.filename
+
                "\n Caused by "+ e.cause);
                System.out.println("Loading default configuration");
                config = DEFAULT_CONFIG;
        }
        return new Config(config);
}
```

2. Gérer l'erreur ailleurs (error propagation)

```
try {
        return Files.readString(filePath);
} catch (IOException e) {
        // propagate error upwards
        throw new UnreadableFile(filename, e);
}
```

3. Pas de possibilités de résoudre l'erreur (crash)

```
public static void main(String[] args) {
        Config config = loadConfig();
        try {
                startHttpServer(config);
        } catch (Exception e) {
                // log error
                e.printStackTrace();
                System.err.println("Could not start HTTP Server");
                // Terminate program with error code
                System.exit(1);
        }
}
```

Tests

```
@Test
public void testAdd() {
        int result = add(4, 5);
        assert result == 9;
}
```

prints -> asserts

- améliore la qualité du programme
- sustainable development
- documentation
- confiance envers le code

ChatSystem

- chatter (clavarder en FR) sur le réseau de l'entreprise donc pas de serveur
- créer un MVP : faire les features clé
- faire un code maintenable et de bonne qualité : évalué là dessus

Organisation

- 2 personnes même groupe de TP
- conception UML et projet JAVA
- pas de communications entre les groupes : protocoles distincts

fonctionnalités

- chat décentralisé
- voir qui est connecté sur le réseaux : scanner
- échanger des messages
- avoir un historique du chat
- GUI

y'a une doc de spécification à respecter
-> euh faut compatible windows, mac, linux, android -> pas besoin de tout faire (car MVP) mais faut démontrer que ce soit possible (typiquement android ptdrr) !

Besoins techniques

- UDP et broadcast pour découvrir les contacts
- sessions TCP pour les messages
- JDBC et SQLlite pour l'historique des messages
- Swing pour l'interface graphique
- GIT, MAVEN, UNIT TESTING, CI et dependencies management

3 phases

phase 1 : contact discovery

-> jusqu'au 17novembre -> evaluation par les paires en phase2
-> 3TD UML (cas d'utilisation, sequence, class)
-> implémentation (4TP)

phase 2 :

-> présentation des solutions aux problemes : pour implémenter plus proprement
-> en gros correction du projet par le prof pour améliorer notre manière de coder
-> live coding : design pattern observer... (plein de capsule de 10min seulement sur chaque fonctionnalités)
-> evaluer le code d'un autre groupe
-> exam sous forme d'un QCM sur le cours du début de semestre sur machine : accès a tout (1/2h) mais faut être rapide. (en décembre)

phase 3 :

-> correction du contact discovery
-> créer l'application complete

-> 6 séances de TP
-> des séances d'UML
-> un peu après la fin du semestre on rend le projet complet
-> Faut rendre le projet + doc (readme) **qui explique comment deploy le projet aussi pour l'utilisateur finale** + test
-> rapport de conception à faire en UML

> on peux check sur le site une checklist des features a chaque release ! (a la fin du projet mais c'est mieux si on regarde à chaque fois)