Programming Assignment#2

< Designing a pid Manager >

20102122 정효안

From now on, I will implement a pid manager that manages the process identifiers (pids).

The overview of the report is as follows.

- Approach to the problem
- Overview of the implementation
- Design
- Results
- Conclusion

Approach to the problem

1. Make **PIDTester** that takes number of threads created, life time of a thread while the program is running and the life time of the program.

2. Make **PIDManagerClass** that implements PIDManager Interface.

   (There are getPID( ), getPIDWait( ), releasePID( ))

3. Make **MyThread**. By this and PIDTester, we can test both getPID() and getPIDWait().

Overview of the implementation

1. PIDTester

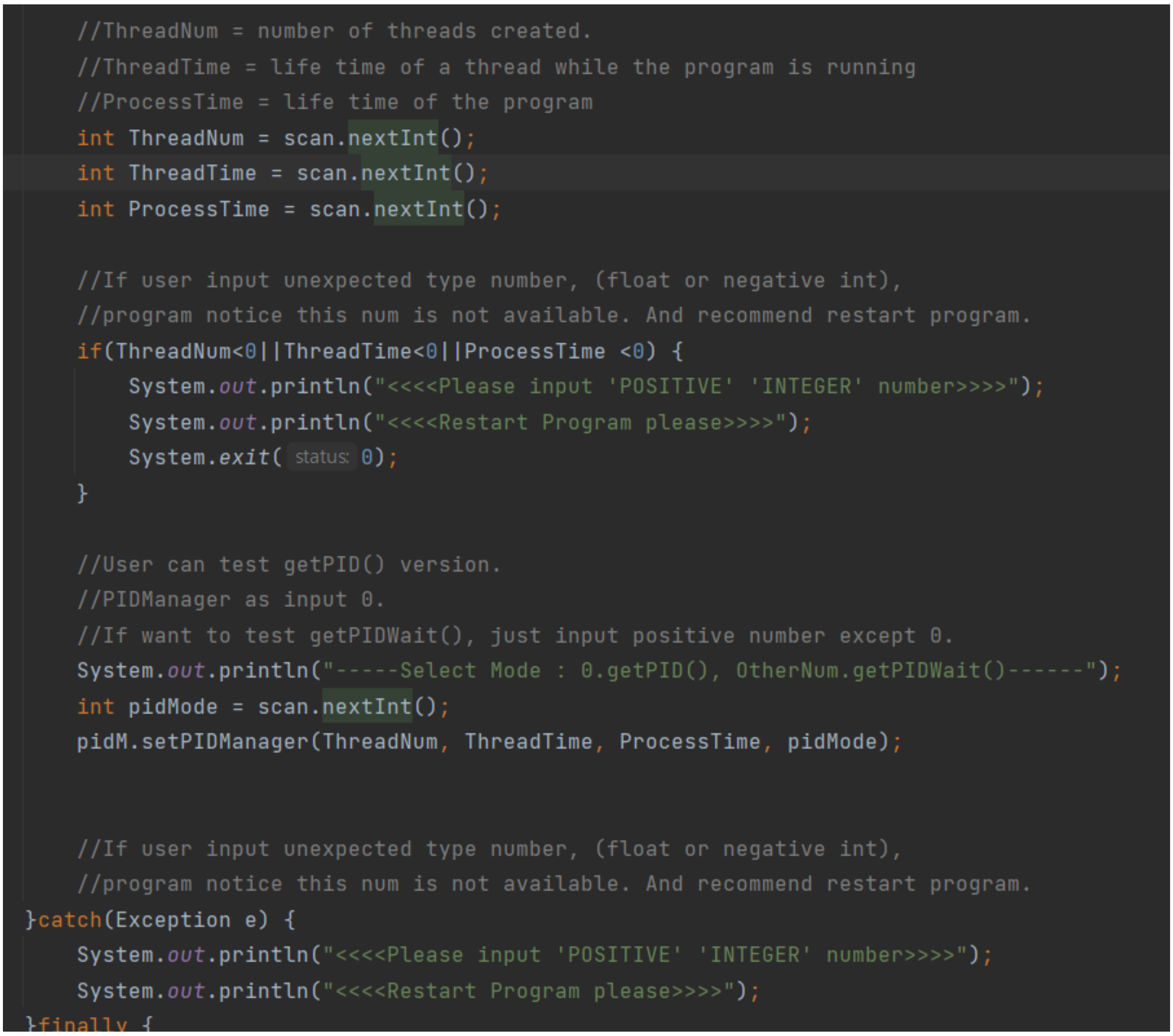
2. PIDManagerClass

3. MyThread

Design

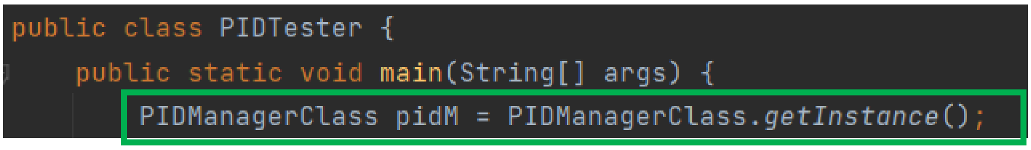
What I already had is a PIDManager Interface. Here is an explanation of the important parts of the code. (In the picture, I added explanation of code by comment.)

PIDTester

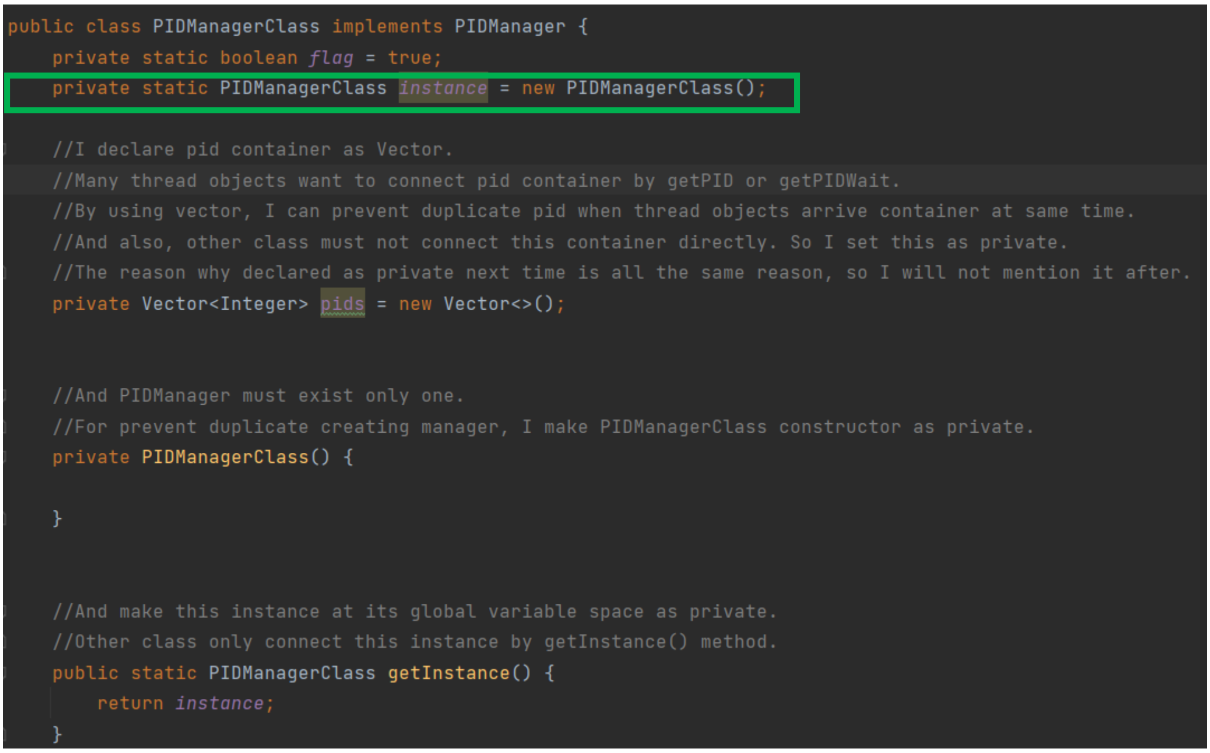


```
    //ThreadNum = number of threads created.
    //ThreadTime = life time of a thread while the program is running
    //ProcessTime = life time of the program
    int ThreadNum = scan.nextInt();
    int ThreadTime = scan.nextInt();
    int ProcessTime = scan.nextInt();

    //If user input unexpected type number, (float or negative int),
    //program notice this num is not available. And recommend restart program.
    if(ThreadNum<0||ThreadTime<0||ProcessTime <0) {
        System.out.println("<<<<Please input 'POSITIVE' 'INTEGER' number>>>>");
        System.out.println("<<<<Restart Program please>>>>");
        System.exit( status: 0);
    }

    //User can test getPID() version.
    //PIDManager as input 0.
    //If want to test getPIDWait(), just input positive number except 0.
    System.out.println("-----Select Mode : 0.getPID(), OtherNum.getPIDWait()------");
    int pidMode = scan.nextInt();
    pidM.setPIDManager(ThreadNum, ThreadTime, ProcessTime, pidMode);


    //If user input unexpected type number, (float or negative int),
    //program notice this num is not available. And recommend restart program.
}catch(Exception e) {
    System.out.println("<<<<Please input 'POSITIVE' 'INTEGER' number>>>>");
    System.out.println("<<<<Restart Program please>>>>");
}finally {
```

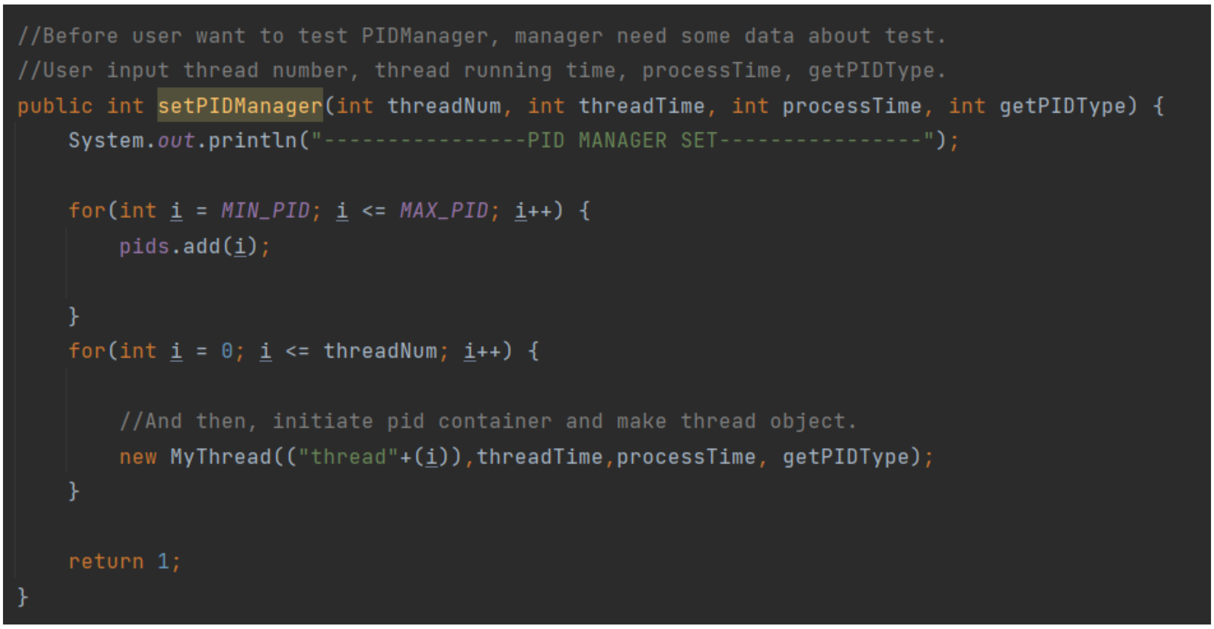
PIDManagerClass



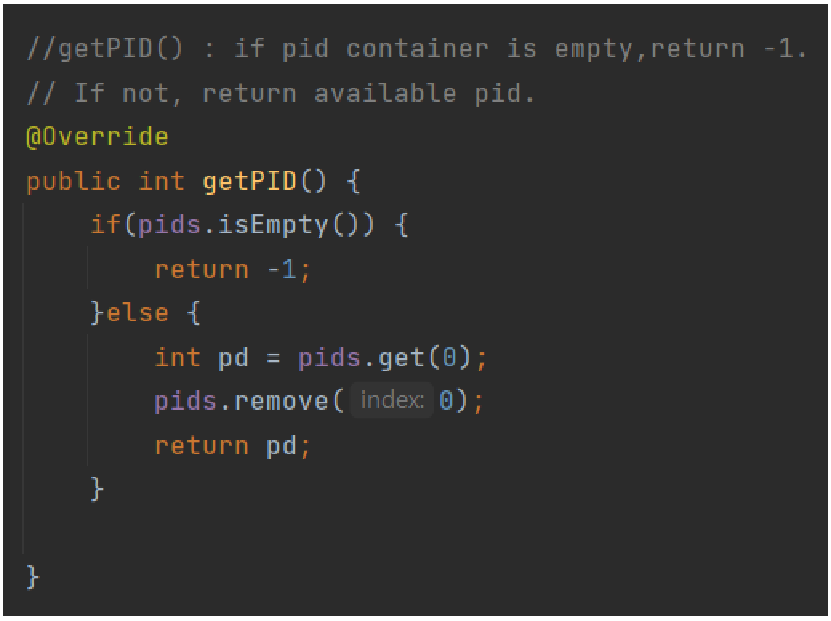
```java
public class PIDTester {
    public static void main(String[] args) {
        PIDManagerClass pidM = PIDManagerClass.getInstance();
```



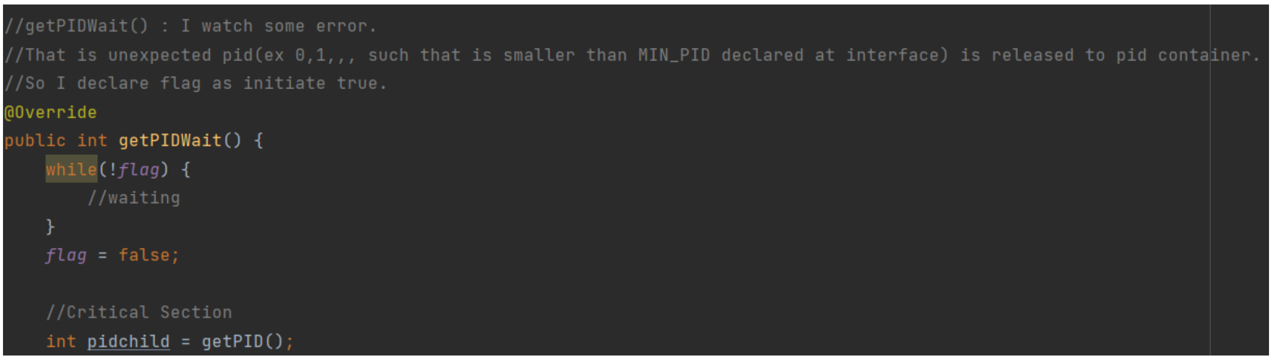
```java
public class PIDManagerClass implements PIDManager {
    private static boolean flag = true;
    private static PIDManagerClass instance = new PIDManagerClass();

    //I declare pid container as Vector.
    //Many thread objects want to connect pid container by getPID or getPIDWait.
    //By using vector, I can prevent duplicate pid when thread objects arrive container at same time.
    //And also, other class must not connect this container directly. So I set this as private.
    //The reason why declared as private next time is all the same reason, so I will not mention it after.
    private Vector<Integer> pids = new Vector<>();


    //And PIDManager must exist only one.
    //For prevent duplicate creating manager, I make PIDManagerClass constructor as private.
    private PIDManagerClass() {

    }


    //And make this instance at its global variable space as private.
    //Other class only connect this instance by getInstance() method.
    public static PIDManagerClass getInstance() {
        return instance;
    }
```



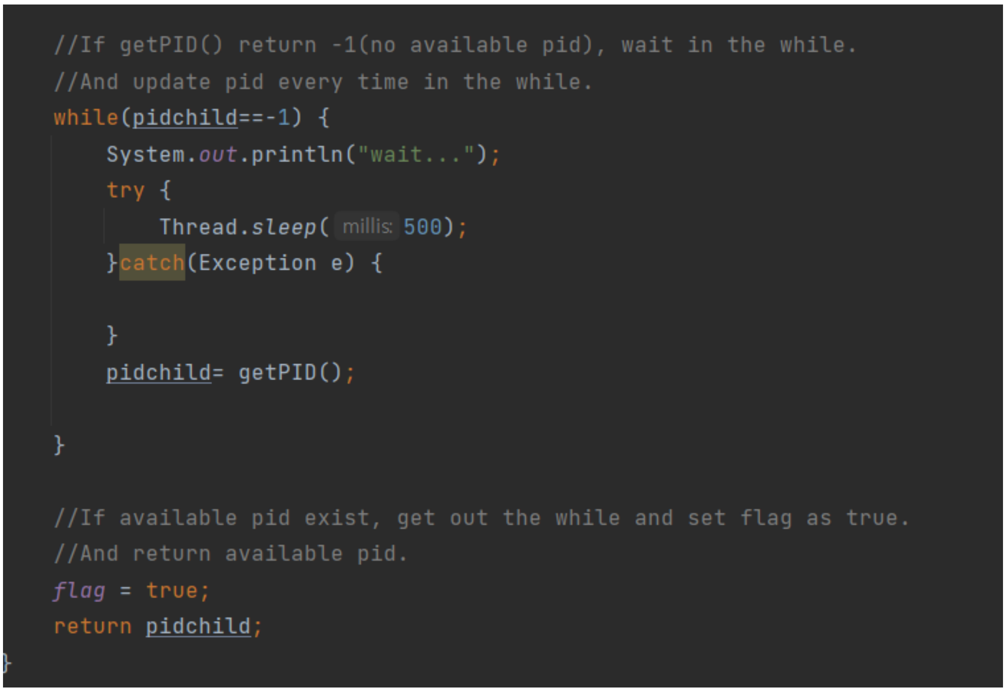
```java
//Before user want to test PIDManager, manager need some data about test.
//User input thread number, thread running time, processTime, getPIDType.
public int setPIDManager(int threadNum, int threadTime, int processTime, int getPIDType) {
    System.out.println("----------------PID MANAGER SET----------------");

    for(int i = MIN_PID; i <= MAX_PID; i++) {
        pids.add(i);

    }
    for(int i = 0; i <= threadNum; i++) {

        //And then, initiate pid container and make thread object.
        new MyThread(("thread"+(i)),threadTime,processTime, getPIDType);
    }

    return 1;
}
```



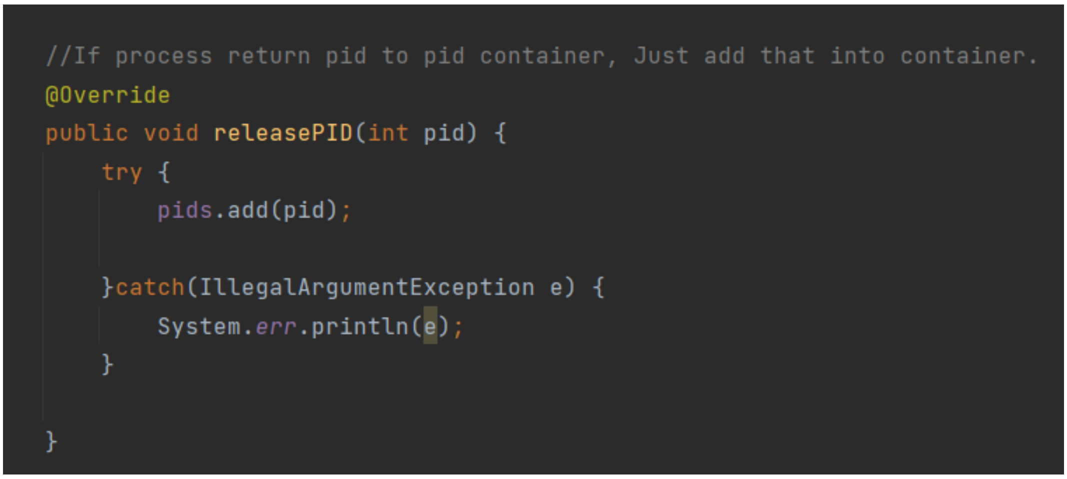
```java
//getPID() : if pid container is empty,return -1.
// If not, return available pid.
@Override
public int getPID() {
    if(pids.isEmpty()) {
        return -1;
    }else {
        int pd = pids.get(0);
        pids.remove( index: 0);
        return pd;
    }

}
```



```java
//getPIDWait() : I watch some error.
//That is unexpected pid(ex 0,1,,, such that is smaller than MIN_PID declared at interface) is released to pid container.
//So I declare flag as initiate true.
@Override
public int getPIDWait() {
    while(!flag) {
        //waiting
    }
    flag = false;

    //Critical Section
    int pidchild = getPID();
```

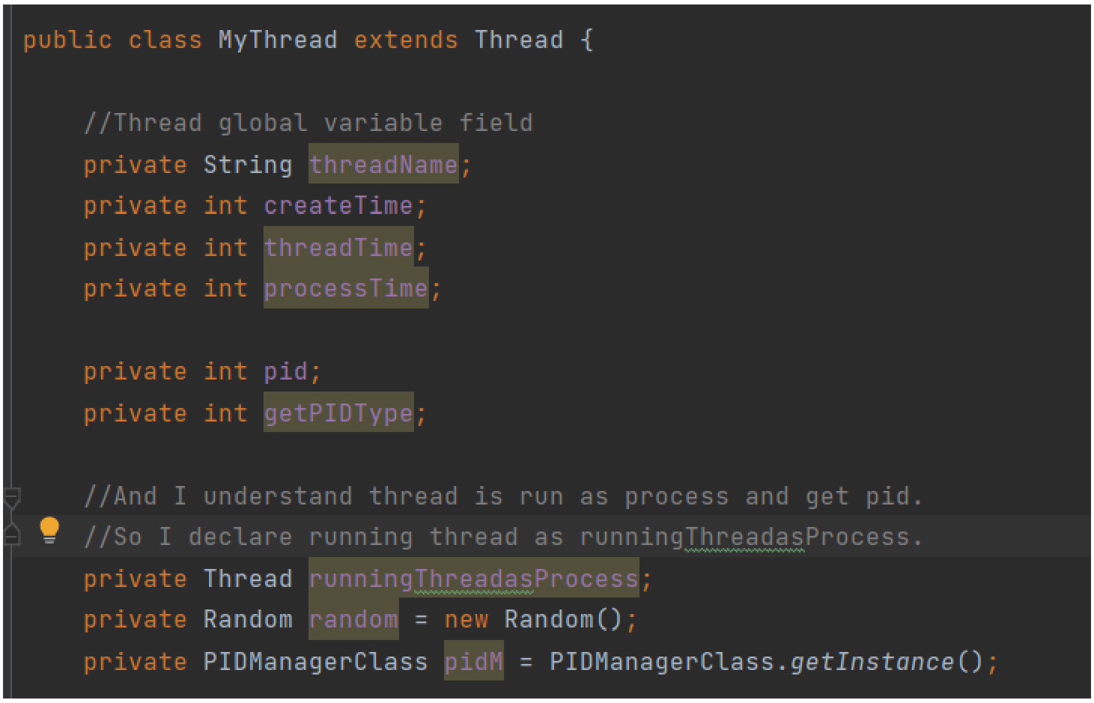


```java
    //If getPID() return -1(no available pid), wait in the while.
    //And update pid every time in the while.
    while(pidchild==-1) {
        System.out.println("wait...");
        try {
            Thread.sleep( millis: 500);
        }catch(Exception e) {

        }
        pidchild= getPID();

    }

    //If available pid exist, get out the while and set flag as true.
    //And return available pid.
    flag = true;
    return pidchild;
}
```

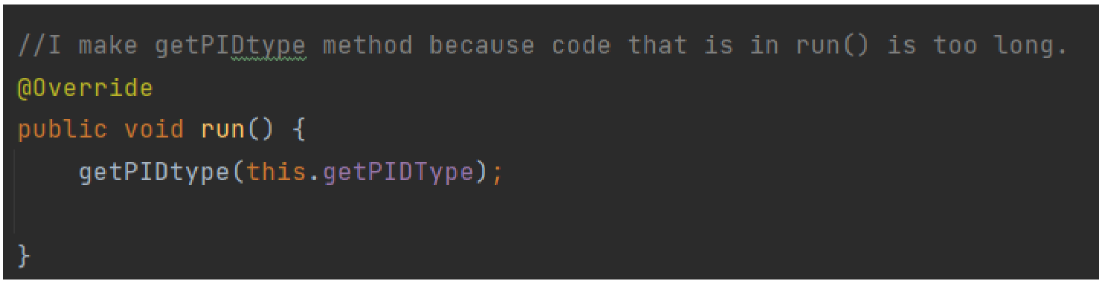


```
//If process return pid to pid container, Just add that into container.
@Override
public void releasePID(int pid) {
    try {
        pids.add(pid);

    }catch(IllegalArgumentException e) {
        System.err.println(e);
    }

}
```

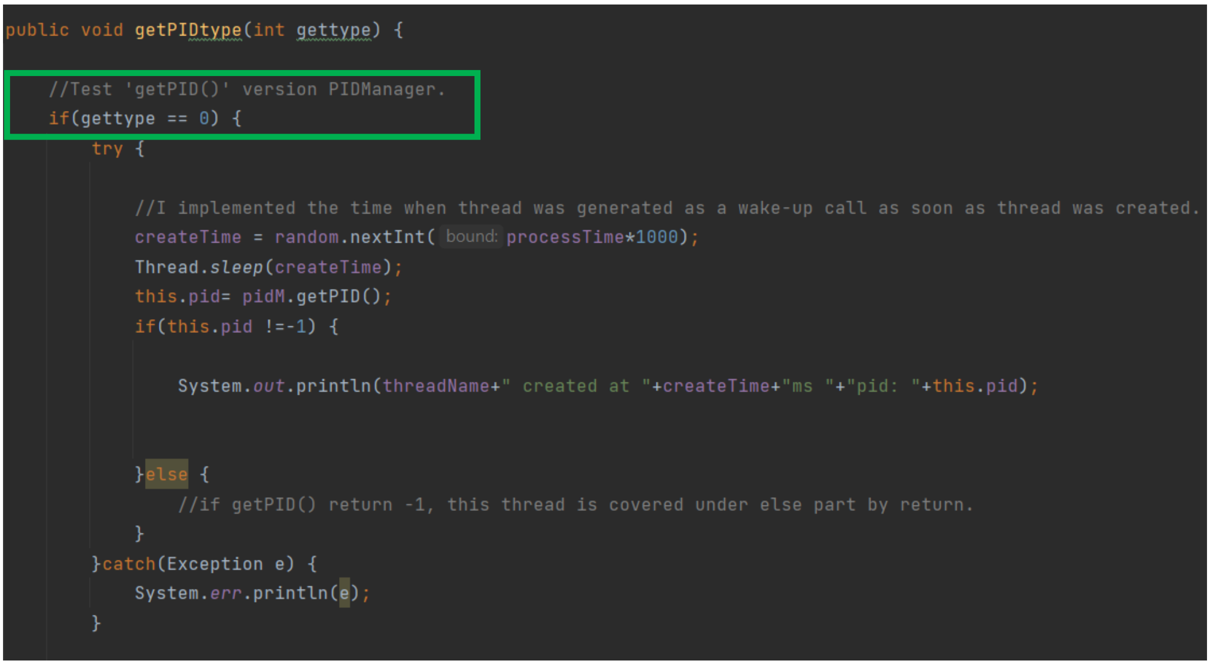
MyThread



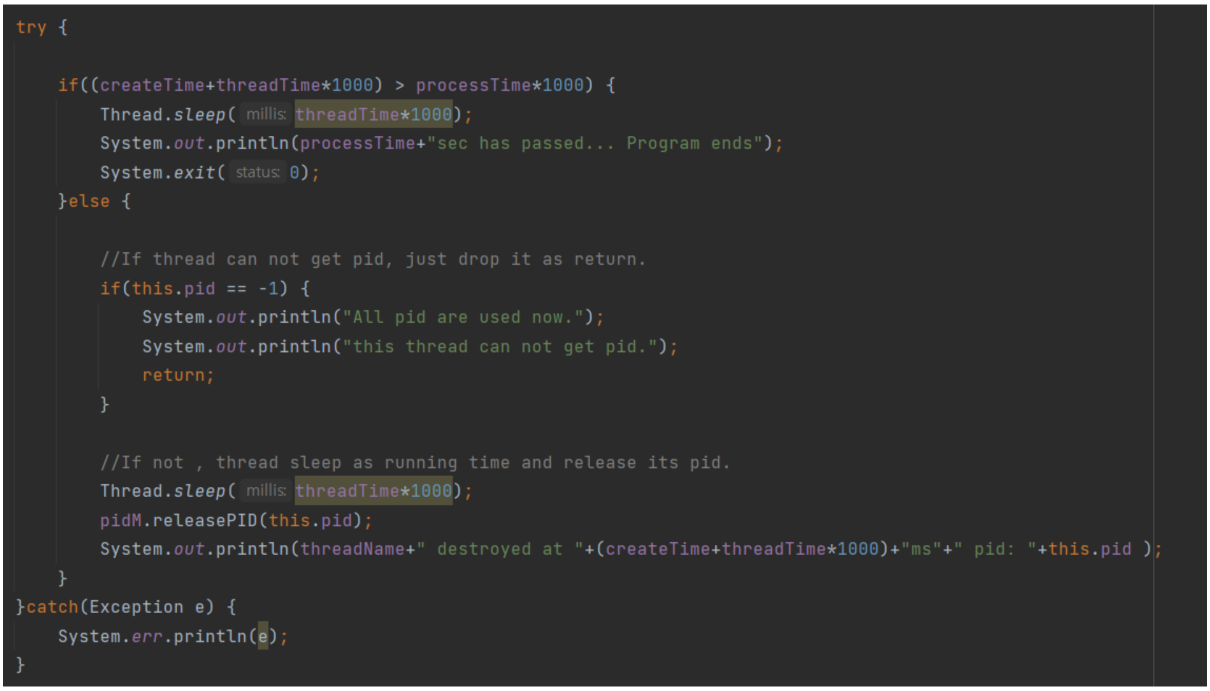
```
public class MyThread extends Thread {

    //Thread global variable field
    private String threadName;
    private int createTime;
    private int threadTime;
    private int processTime;

    private int pid;
    private int getPIDType;

    //And I understand thread is run as process and get pid.
    //So I declare running thread as runningThreadasProcess.
    private Thread runningThreadasProcess;
    private Random random = new Random();
    private PIDManagerClass pidM = PIDManagerClass.getInstance();
```



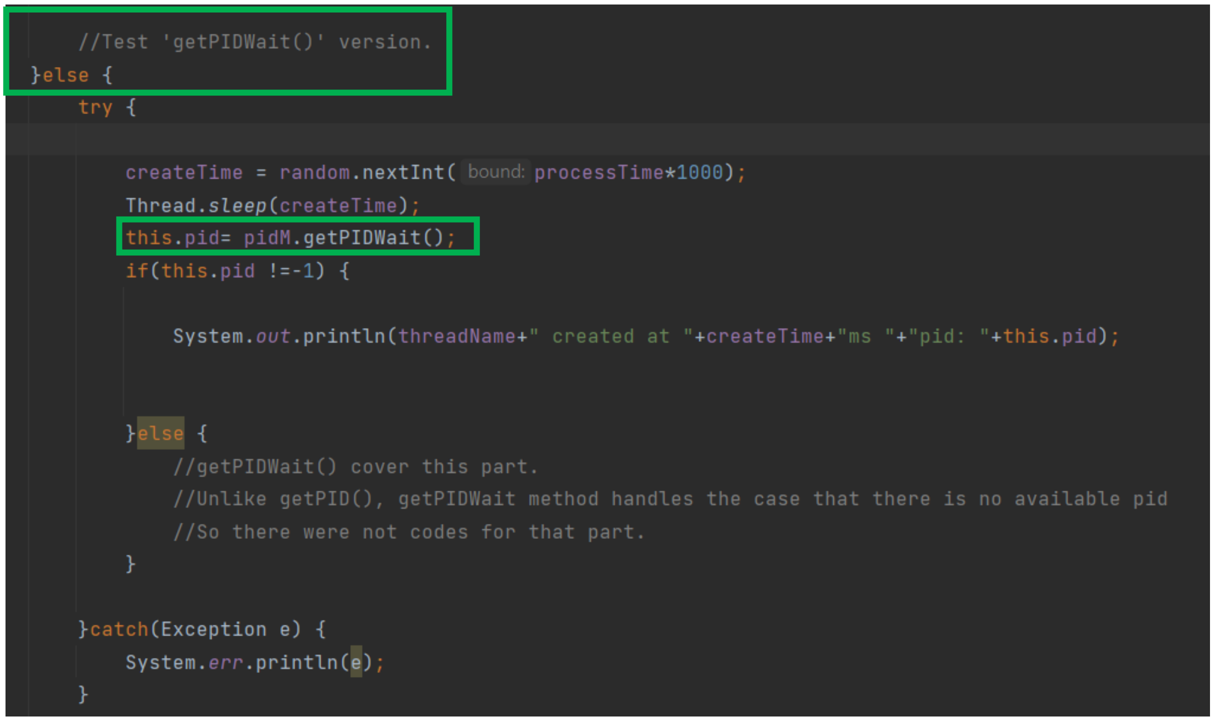
```
//I make getPIDtype method because code that is in run() is too long.
@Override
public void run() {
    getPIDtype(this.getPIDType);

}
```



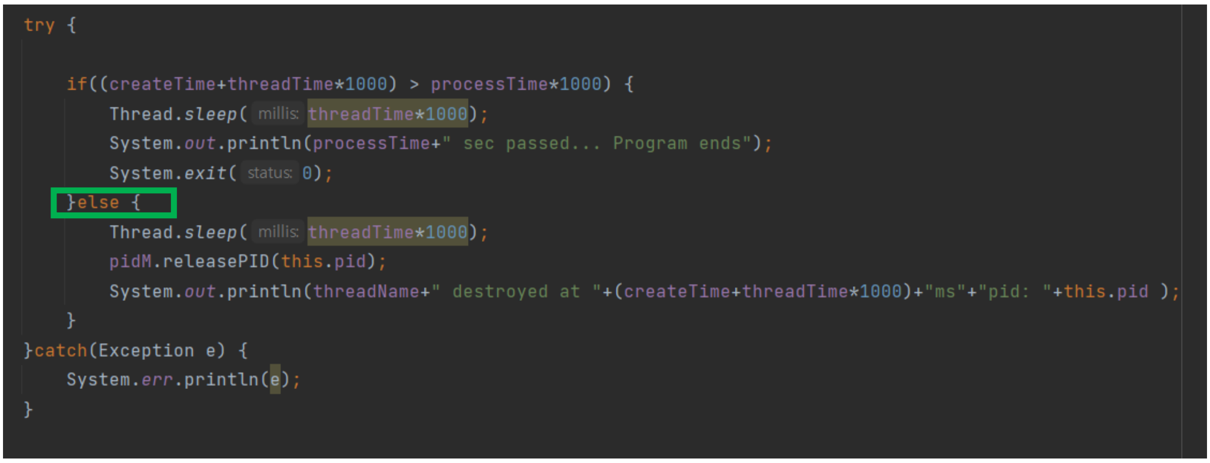
```
public void getPIDtype(int gettype) {

    //Test 'getPID()' version PIDManager.
    if(gettype == 0) {
        try {

            //I implemented the time when thread was generated as a wake-up call as soon as thread was created.
            createTime = random.nextInt( bound: processTime*1000);
            Thread.sleep(createTime);
            this.pid= pidM.getPID();
            if(this.pid !=-1) {

                System.out.println(threadName+" created at "+createTime+"ms "+"pid: "+this.pid);


            }else {
                //if getPID() return -1, this thread is covered under else part by return.
            }
        }catch(Exception e) {
            System.err.println(e);
        }
```



```
try {

    if((createTime+threadTime*1000) > processTime*1000) {
        Thread.sleep( millis: threadTime*1000);
        System.out.println(processTime+"sec has passed... Program ends");
        System.exit( status: 0);
    }else {

        //If thread can not get pid, just drop it as return.
        if(this.pid == -1) {
            System.out.println("All pid are used now.");
            System.out.println("this thread can not get pid.");
            return;
        }

        //If not , thread sleep as running time and release its pid.
        Thread.sleep( millis: threadTime*1000);
        pidM.releasePID(this.pid);
        System.out.println(threadName+" destroyed at "+(createTime+threadTime*1000)+"ms"+" pid: "+this.pid );
    }
}catch(Exception e) {
    System.err.println(e);
}
```



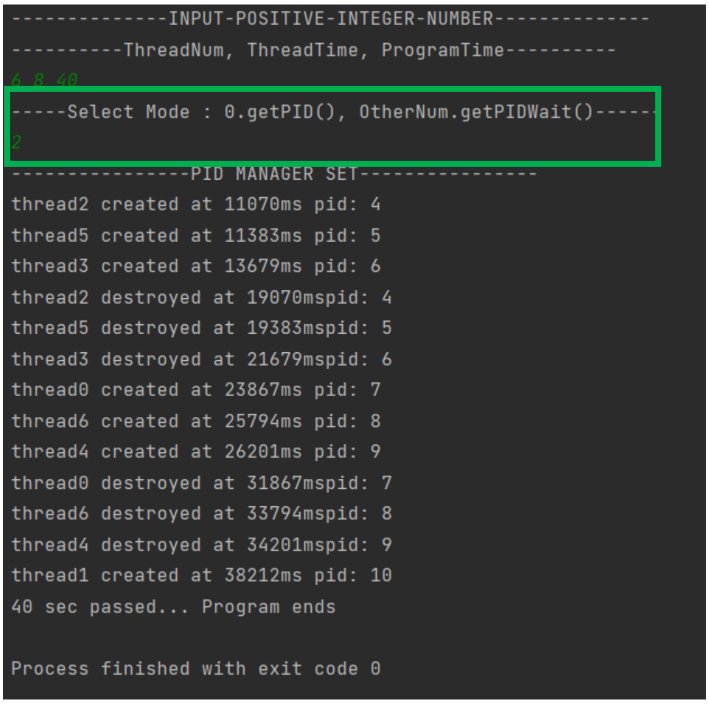
```
    //Test 'getPIDWait()' version.
}else {
    try {

        createTime = random.nextInt( bound: processTime*1000);
        Thread.sleep(createTime);
        this.pid= pidM.getPIDWait();
        if(this.pid !=-1) {

            System.out.println(threadName+" created at "+createTime+"ms "+"pid: "+this.pid);


        }else {
            //getPIDWait() cover this part.
            //Unlike getPID(), getPIDWait method handles the case that there is no available pid
            //So there were not codes for that part.
        }

    }catch(Exception e) {
        System.err.println(e);
    }
```



```
try {

    if((createTime+threadTime*1000) > processTime*1000) {
        Thread.sleep( millis: threadTime*1000);
        System.out.println(processTime+" sec passed... Program ends");
        System.exit( status: 0);
    }else {
        Thread.sleep( millis: threadTime*1000);
        pidM.releasePID(this.pid);
        System.out.println(threadName+" destroyed at "+(createTime+threadTime*1000)+"ms"+"pid: "+this.pid );
    }
}catch(Exception e) {
    System.err.println(e);
}
```

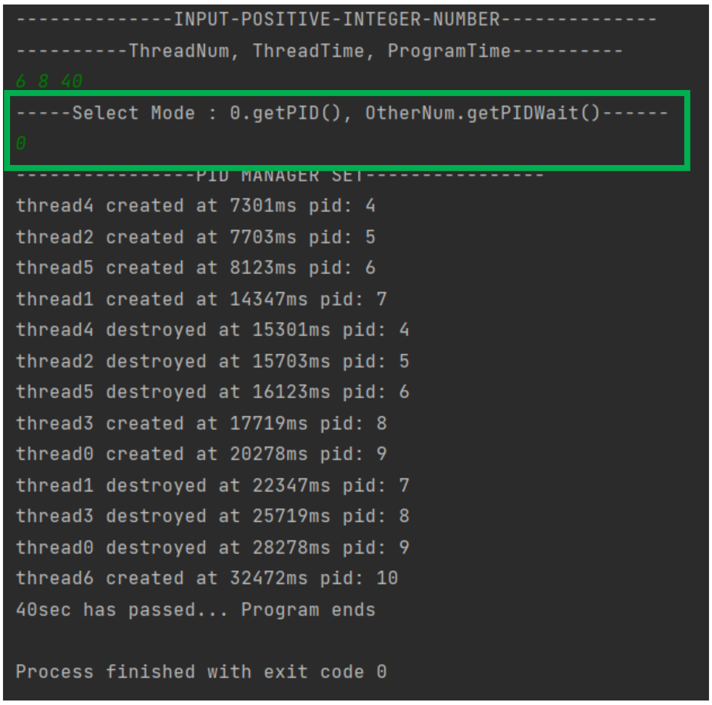
Result

Test **getPIDWait()** (by entering '2')



```
--------------INPUT-POSITIVE-INTEGER-NUMBER--------------
----------ThreadNum, ThreadTime, ProgramTime----------
6 8 40
-----Select Mode : 0.getPID(), OtherNum.getPIDWait()------
2
----------------PID MANAGER SET----------------
thread2 created at 11070ms pid: 4
thread5 created at 11383ms pid: 5
thread3 created at 13679ms pid: 6
thread2 destroyed at 19070mspid: 4
thread5 destroyed at 19383mspid: 5
thread3 destroyed at 21679mspid: 6
thread0 created at 23867ms pid: 7
thread6 created at 25794ms pid: 8
thread4 created at 26201ms pid: 9
thread0 destroyed at 31867mspid: 7
thread6 destroyed at 33794mspid: 8
thread4 destroyed at 34201mspid: 9
thread1 created at 38212ms pid: 10
40 sec passed... Program ends

Process finished with exit code 0
```

Test **getPID()** (by entering '0')



```
--------------INPUT-POSITIVE-INTEGER-NUMBER--------------
----------ThreadNum, ThreadTime, ProgramTime----------
6 8 40
-----Select Mode : 0.getPID(), OtherNum.getPIDWait()------
0
----------------PID MANAGER SET----------------
thread4 created at 7301ms pid: 4
thread2 created at 7703ms pid: 5
thread5 created at 8123ms pid: 6
thread1 created at 14347ms pid: 7
thread4 destroyed at 15301ms pid: 4
thread2 destroyed at 15703ms pid: 5
thread5 destroyed at 16123ms pid: 6
thread3 created at 17719ms pid: 8
thread0 created at 20278ms pid: 9
thread1 destroyed at 22347ms pid: 7
thread3 destroyed at 25719ms pid: 8
thread0 destroyed at 28278ms pid: 9
thread6 created at 32472ms pid: 10
40sec has passed... Program ends

Process finished with exit code 0
```

Conclusion

Through this project, I was able to understand the principles more deeply by directly implementing the pidmanager that I learned only by theory.