Object - Oriented Programming

LAB #12_1. Threads

Thread

- 하나의 프로세스 내부에서 독립적으로 실행되는 작업 단위

- 프로세스 내부에 thread가 한 개 존재하면, 단일 스레드 (Single Thread),
  두 개 이상 존재하면, 다중 스레드 (Multi Thread)

- 하나의 스레드 내부에서의 작업 처리 단계는 순차적

Threads in Java

- JVM 에 의해 하나의 프로세스가 발생하고,
  main() 안의 스레드 인스턴스의 메소드를 포함한 실행문들이 thread이다.

- Main() 이외의 다른 thread를 생성하려면 Thread 클래스를 상속하거나
  Runnable 인터페이스를 implements 하면 된다.

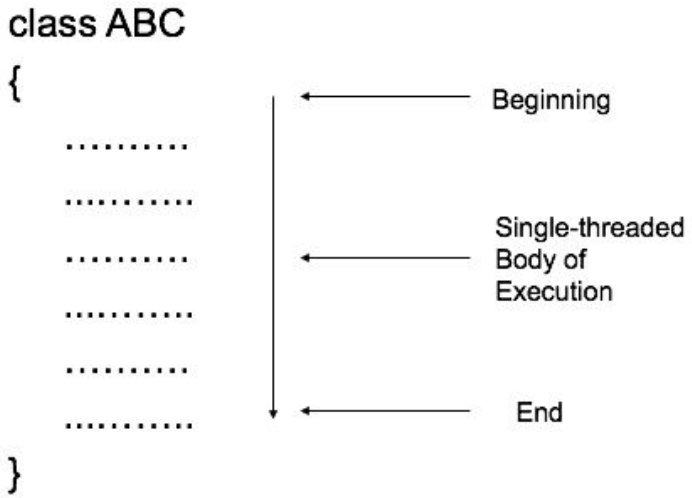
Threads & Multithreading

- Multi threading?
  프로세스(프로그램이 자원을 할당받고 실행되면 프로세스가 됨)가
  두 개 이상의 서브 프로세스로 나누어져서
  동시에 병렬로 구현될 수 있는 프로그래밍 개념이다.

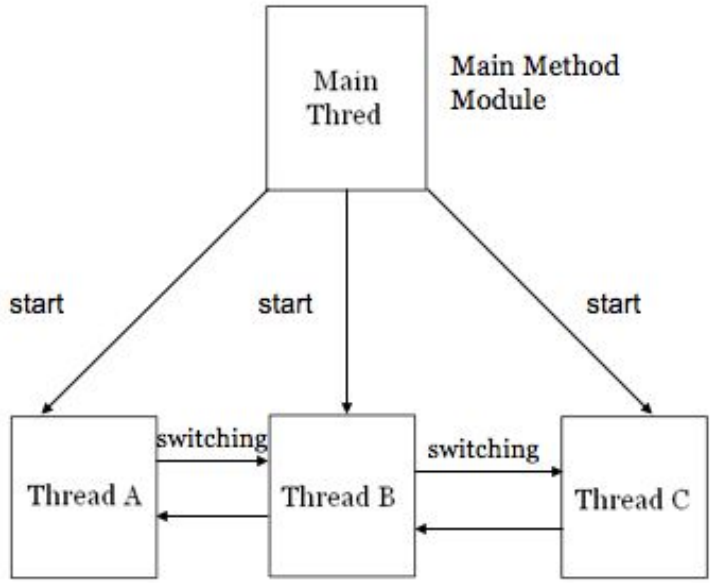
- Thread 사용의 이점
  - 병렬 처리 가능
  - CPU의 유휴시간 활용 가능
  - 우선순위에 따른 작업 순서 지정 가능

Threads & Multithreading

Single Thread



MultiThread



Threads & Multithreading

Single Thread

```
public class SingleThread {
    public static void main(String[] args) {
        try {
            System.out.println("Start");

            System.out.println("A");
            Thread.sleep(3000);
            System.out.println("End A");

            System.out.println("B");
            Thread.sleep(3000);
            System.out.println("End B");

            System.out.println("C");
            Thread.sleep(3000);
            System.out.println("End C");

            System.out.println("End");
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

MultiThread

```
public class MultiThread extends Thread {

    public void run() {
        System.out.println(this.getName());
        try {
            sleep(3000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("End " + this.getName());

    }

    public static void main(String[] args) {
        System.out.println("Start");

        MultiThread a = new MultiThread();
        a.setName("A");
        MultiThread b = new MultiThread();
        b.setName("B");
        MultiThread c = new MultiThread();
        c.setName("C");


        a.start();
        b.start();
        c.start();

        System.out.println("End");

    }
}
```

Threads in Java

- Java는 Thread 클래스와 Runnable 인터페이스를 제공함.
  - 따라서 thread를 사용할 수 있는 방법이 두가지이다.
    - Thread 클래스 확장

      ```
      public class MyThreadClass extends Thread{.. }
      ```

    - Runnable 인터페이스 구현

      ```
      public class MyThreadClass implements Runnable{.. }
      ```

- Thread 클래스 및 Runnable 인터페이스에는 run() 메소드가 있다.
  - public void run() 메소드는 순차 프로그램의 main 메소드처럼 작동한다.
    -> 스레드 시작 진입점
  - Thread가 필요한 동작을 수행할 수 있도록 run() 메소드를 override하거나 implement해야함

Threads in Java

extends Thread

```
public class ThreadTest extends Thread {

    public void run() {
        System.out.println("Thread run.");
    }

    public static void main(String[] args) {
        ThreadTest t1 = new ThreadTest();
        ThreadTest t2 = new ThreadTest();

        t1.start();
        t2.start();

    }
}
```

implements Runnable

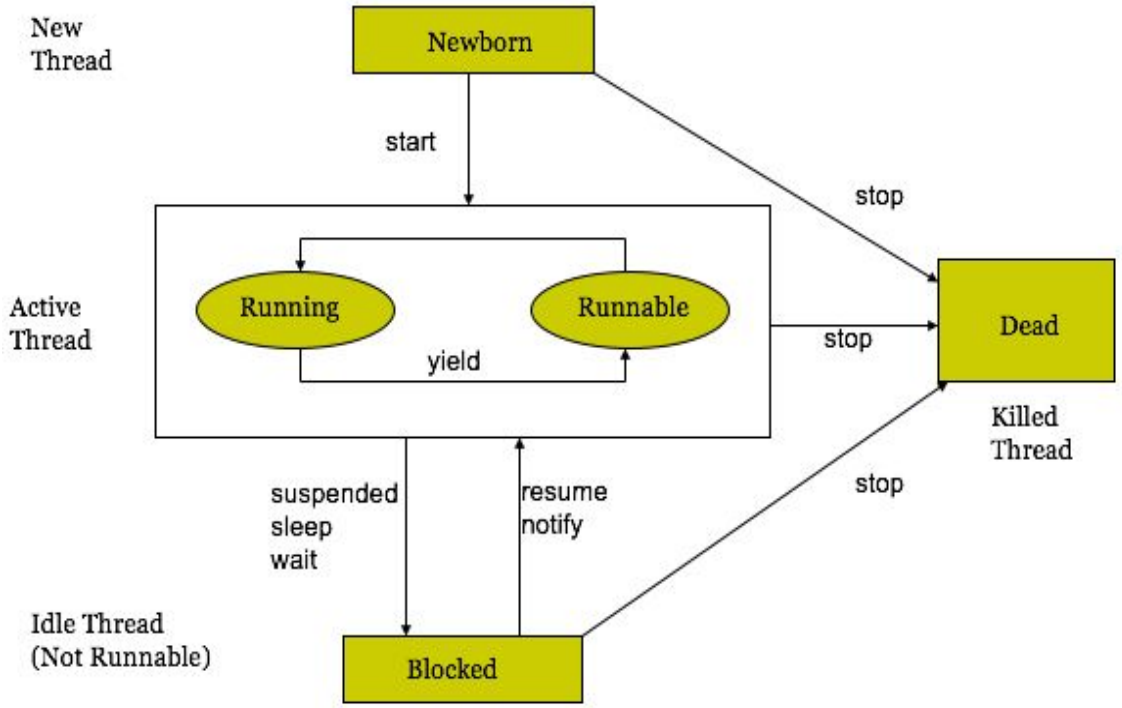
```
public class ThreadTest implements Runnable {

    public void run() {
        System.out.println("Thread run.");
    }

    public static void main(String[] args) {
        Thread t1 = new Thread(new ThreadTest());
        Thread t2 = new Thread(new ThreadTest());

        t1.start();
        t2.start();
    }
}
```

Thread Lifecycle



Thread의 상태 관련 method

New (Newborn)	start()
Runnable	yield()
Running	~~suspend()~~, wait(), sleep()
Blocked	~~resume()~~, notify()
Terminated (Dead)	~~stop()~~

Thread Lifecycle

- Runnable (준비 상태)
  - CPU를 점유하여 실행하기 위해 대기하고 있는 상태 (Running 이전 상태)
  - 코드 상에서 start() 메소드를 호출하면 run() 메소드에 설정된 스레드가 Runnable 상태 진입
- Running (실행 상태)
  - CPU를 점유하여 실행중인 상태
  - Runnable 상태의 여러 thread중 우선 순위를 가진 thread가 결정되면 JVM이 자동으로 run() 메소드를 호출하여 thread가 Running 상태 진입

Thread Lifecycle

- Dead (종료 상태)
  - Running 상태에서 thread가 모두 실행된 후 완료 상태
  - 또는 완료되진 않았지만 임의로 kill 된 상태
- Blocked (지연 상태)
  - CPU 점유권을 상실한 상태
  - wait() 메소드에 의해 Blocked 상태가 된 thread는 notify() 메소드가 호출되면 Runnable 상태로 전환됨
  - sleep() 메소드에 의해 Blocked 상태가 된 thread는 지정된 시간이 지나면 Runnable 상태로 전환됨
  - I/O에 관련된 메소드에 의해 I/O Blocked 상태가 된 thread는 I/O 관련 메소드의 실행이 끝나면 Runnable 상태로 전환됨

Thread Example

- Producer Consumer
  - 난수를 생성하는 producer thread
  - 난수를 사용하는 consumer thread

```
package multithread;

public class Producer extends Thread {

    public Producer(){ }

    public void produce(){

        for(int i=0; i<10; i++){
            prodConTest.NUMBER = Math.random()*100;
            System.out.println("Producer: "+prodConTest.NUMBER );
        }
    }

    public void run(){
        produce();
    }

}
```

```
package multithread;

public class Consumer extends Thread {

    public Consumer(){  }

    public void consume(){
        for (int i=10; i>=0; i--){
                System.out.println("Consumer: "+prodConTest.NUMBER);
        }
    }

    public void run(){
        consume();
    }
}
```

Thread Example

- Main 메소드가 producer, consumer. 두 개의 thread를 생성.
  - producer thread
  - consumer thread

```
package multithread;

public class prodConTest {
    static double NUMBER;

    public static void main(String[] args){

        Producer producer = new Producer();
        Consumer consumer = new Consumer();
        producer.start();
        consumer.start();

    }
}
```

```
Consumer: 0.0
Consumer: 82.26335058969953
Producer: 82.26335058969953
Consumer: 82.26335058969953
Producer: 23.827262626261348
Producer: 32.16717390573557
Consumer: 23.827262626261348
Producer: 91.34480443971357
Consumer: 91.34480443971357
Producer: 62.256633263452656
Consumer: 62.256633263452656
Producer: 72.88430578958373
Consumer: 72.88430578958373
Producer: 86.29986539827502
Consumer: 86.29986539827502
Producer: 25.044011441522727
Consumer: 25.044011441522727
Producer: 83.09676926554043
Consumer: 83.09676926554043
Producer: 60.64252656797524
Consumer: 60.64252656797524
```

Synchronization

- Java에서 Synchronization(동기화)는
  다중 thread에 의한 공유 자원에 대한 접근을 제어함.
  - Thread 간섭 방지
  - 일관성 문제 방지
  - 데이터 변형 방지

- Multithreading은 비동기 동작을 사용한다.
- 필요할 경우 동시성을 강제로 사용한다.

  -> synchronized 키워드를 사용

Synchronization

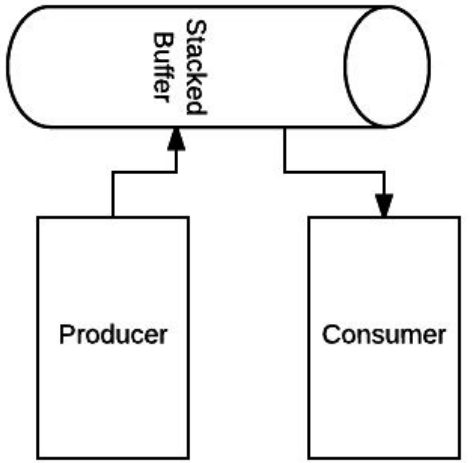
- 동작
  - Thread가 동기화 된 메소드를 실행하면 자동으로 해당 객체에 대한 lock을 획득
  - 메소드가 끝나면 lock이 해제됨
  - 동시간에 하나의 thread만 lock을 가질 수 있다.

- 동기화는 한번에 하나의 thread만 객체에 대한 작업을 수행할 수 있도록 한다.
- 여러 thread가 객체에 접근하려 할 때, 동기화는 일관성을 유지하는데 도움이 된다.

Synchronized Producer Consumer

- 이 버전의 producer consumer에는 단일 스택 버퍼가 있다.
- Producer가 버퍼에 데이터를 추가하고 버퍼가 full이 되면 대기한다.
- Consumer는 버퍼에서 데이터를 가져오고 버퍼가 empty가 되면 대기한다.

- 이를 동기화하기 위해 필요한 것
  - Synchronized - 메소드를 동기화한다.
  - Wait() - 스레드를 blocked 상태로 전환한다.
  - Notify() / NotifyAll()- 스레드를 active 상태로 전환한다.



Synchronized Producer Consumer

```
package multithread1;

public class Buffer {

    private int loc = 0;
    private double[] data;

    public Buffer(int size){
        data = new double[size];
    }

    public int getSize(){return data.length;}

    public synchronized void add(double toAdd) throws InterruptedException{
        if(loc >= data.length){
            System.out.println("Buffer is full.");
            wait();
        }
        System.out.println("Adding item "+toAdd);
        System.out.flush();
        data[loc++] = toAdd;
        notifyAll();
    }

    public synchronized double remove() throws InterruptedException{
        if (loc <= 0){
            System.out.println("Buffer is empty.");
            wait();
        }
            double hold = data[--loc];
            data[loc] = 0.0;
            System.out.println("Removing item "+hold);
            System.out.flush();
            notifyAll();
            return hold;
    }

    public synchronized String toString(){
        String toReturn = "";
        for(int i=0; i<data.length; i++){
            toReturn += String.format("%2.2f", data[i])+" ";
        }
        return toReturn;
    }
}
```

Synchronized Producer Consumer

```java
package multithread1;

public class Producer extends Thread {

    private int pNum;
    private final Buffer buffer;

    public Producer(Buffer buffer){
        this.buffer = buffer;
    }

    public void produce() throws InterruptedException{
        for(int i=0; i<buffer.getSize(); i++){
            buffer.add(Math.random()*100);
        }
    }

    public void run(){
        try {
            produce();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

}
```

```java
package multithread1;

public class Consumer extends Thread {
    private int pNum;
    private Buffer buffer;

    public Consumer(Buffer buffer){
        this.buffer = buffer;
    }

    public void consume() throws InterruptedException{
        for (int i=buffer.getSize(); i>=0; i--){
            buffer.remove();
        }
    }

    public void run(){
        try {
            consume();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

i > 0

Synchronized Producer Consumer

```
package multithread2;

public class prodConTest {

    public static void main(String[] args){

        Buffer buff =  new Buffer(10);

        Producer producer = new Producer(buff);
        Consumer consumer = new Consumer(buff);
        producer.start();
        consumer.start();



    }
}
```

```
Buffer is empty.
Adding item 77.8472326959853
Removing item 77.8472326959853
Buffer is empty.
Adding item 26.92239748059311
Adding item 20.580438594134364
Removing item 20.580438594134364
Removing item 26.92239748059311
Buffer is empty.
Adding item 80.29504857263319
Adding item 40.42235604592893
Adding item 1.8605782544117155
Adding item 8.794070081571315
Adding item 12.065152909808718
Adding item 79.15242043184627
Adding item 57.59898790728746
Removing item 57.59898790728746
Removing item 79.15242043184627
Removing item 12.065152909808718
Removing item 8.794070081571315
Removing item 1.8605782544117155
Removing item 40.42235604592893
Removing item 80.29504857263319
Buffer is empty.
```

Other Methods

Method	Meaning
Thread currentThread()	returns a reference to the current thread
Void sleep(long msec)	causes the current thread to wait for msec milliseconds
String getName()	returns the name of the thread.
Int getPriority()	returns the priority of the thread
Boolean isAlive()	returns true if this thread has been started and has not Yet died. Otherwise, returns false.
Void join()	causes the caller to wait until this thread dies.
Void run()	comprises the body of the thread. This method is overridden by subclasses.
Void setName(String s)	sets the name of this thread to s.
Void setPriority(int p)	sets the priority of this thread to p.

Thread Priorities

- Java에서는 각 thread에 우선순위가 지정되며, 실행 순서에 영향을 준다.
- setPriority() method를 사용하여 thread의 우선순위를 설정한다.

```
ThreadName.setPriority(intNumber);
```

- intNumber는 1에서 10사이의 값을 가질 수 있다.
- 정의되어 있는 우선순위 상수
  - MIN_PRIORITY : 1
  - NORM_PRIORITY : 5
  - MAX_PRIORITY : 10
- Default는 NORM_PRIORITY 이다.

실습 과제

- Class Buffer
  - private int loc (0으로 초기화)
    private double[] data
  - 크기의 정수를 인자로 받는 생성자

    -> data를 입력 받은 정수 크기의 double 배열로 초기화
  - int getSize() -> data 배열의 길이를 return 하는 Method
  - synchronized void add(double toAdd) throws InterruptedException
    - Data 배열이 모두 찼을 경우, Buffer is full 을 출력하고 thread를 wait 상태로 만든다.
    - Adding item : toAdd 를 출력하고 data배열에 toAdd를 넣고, loc을 증가시킨다.
  - synchronized double remove() throws InterruptedException
    - Loc이 0 일 경우, Buffer is empty를 출력하고 thread를 wait 상태로 만든다.
    - Removing item : data[loc-1] 를 출력하고, data[loc-1]을 0.0으로 설정한다.
    - loc을 1 감소시키고 data[loc]을 반환한다.

실습 과제

- Class ProdConSelfTest
  - private Buffer buffer
    private Producer producer
    private Consumer consumer
  - 생성자
    - Buffer 객체 생성 및 data 크기를 15로 설정
    - Producer 객체 생성 (buffer 활용)
    - Consumer 객체 생성 (buffer 활용)
  - private class Producer extends Thread (inner class)
    - private final Buffer buffer
    - Buffer를 인자로 받는 public 생성자 - 매개변수 Buffer로 전역 변수 buffer 초기화
    - public void produce() throws InterruptedException
      -> buffer의 add를 호출하여 무작위로 생성된 값(0~100 실수)을 Buffer 배열에 추가
      (배열의 크기만큼 반복)
    - public void run()
      -> produce() 호출

실습 과제

- private class Consumer extends Thread (inner class)
  - private final Buffer buffer
  - Buffer를 인자로 받는 public 생성자
  - public void consume() throws InterruptedException

    -> buffer의 remove를 호출하여 배열의 element를 하나씩 삭제한다. (배열의 크기만큼 반복)
  - public void run()

    -> consume() 호출
- void startThread()
  - producer thread를 start
  - consumer thread를 start

실습 과제

- Class ProdConTest
  - Main method 작성 후 결과 확인

```
public class ProdConTest {

    public static void main(String[] args) {

        ProdConSelfTest a = new ProdConSelfTest();
        a.startThread();
    }

}
```

```
Buffer is empty
Adding item 20.277999988964414
Adding item 24.868429932293
Adding item 63.16531211999455
Adding item 35.97666084701879
Adding item 81.11407395711483
Adding item 49.263554501067176
Removing item 49.263554501067176
Removing item 81.11407395711483
Removing item 35.97666084701879
Removing item 63.16531211999455
Removing item 24.868429932293
Removing item 20.277999988964414
Adding item 18.12436751399764
Removing item 18.12436751399764
Buffer is empty
Adding item 14.046889068665147
Adding item 28.802441549554313
Adding item 19.901957111559774
Adding item 6.120833972874006
Adding item 15.502592947064443
Adding item 91.76835315277727
Adding item 88.76713405275113
Adding item 98.54090622400345
Removing item 98.54090622400345
Removing item 88.76713405275113
Removing item 91.76835315277727
Removing item 15.502592947064443
Removing item 6.120833972874006
Removing item 19.901957111559774
Removing item 28.802441549554313
Removing item 14.046889068665147
Buffer is empty
```

실습 과제 제출

- ProdConSelfTest, ProdConTest, Buffer 클래스를 .java 파일로 제출