Thread Library (JAVA)

4.4 Thread Library

주로 쓰이는 스레드 관련 라이브러리

- POSIX Pthread (POSIX라는 유닉스 계열 OS의 기준을 충족하는 라이브러리, POSIX기준을 충족하면 유닉스 계열의 OS라고 인정받을 수 있다)
- Windows thread
- Java thread

자바를 예시로 사용하는 이유

- 자바는 스레드를 가정하고 설계된 언어이다
- 스레드 관련 제공된 메소드가 많다

자바에서 스레드 사용 방법

JAVA의 class와 interface

class

- 필드와 메서드 들을 모아놓은 집합체
- 필드에 특정 값을 넣어 놓을 수도 있고 메서드를 구현해 놓을 수 있다.

```
class MyClass1{
    public int sum(int a, int b){
        return a + b;
    }
}
class MyClass2 extends MyClass1{
    public int sum(int a, int b, int c){
        return a+b+c;
    }
    public int sub(int a, int b){
        return a-b;
    }
}
```

interface

- class와 달리 메서드를 구현 해 놓지 않는다.
- 메서드는 명시만 해두고 구현은 상속받는 자식 class에서 구현된다.

```
interface MyInterface{
    public int sum();
}
class MyClass implements Myinterface{
    public int sum(int a, int b){
        return a+b;
    }
}
```

Extend와 Implement차이

extend

- 기존 클래스를 상속받아 새로운 클래스를 생성, 메소드를 작성
- 부모 클래스의 구현된 메서드를 받아서 사용
- 부모 클래스는 하나만 상속 가능

implement

- interface라는 구현되지 않은
- 기존 interface 상속받아 구현되지 않은 메소드를 구현

- 부모 interface는 여러개가 될 수 있음

Thread와 Intrerface관계

```java
@FunctionalInterface
public interface Runnable {
    When an object implementing interface Runnable is used to create a thread, starting the thread causes the object's run method to be called in that separately executing thread.
    The general contract of the method run is that it may take any action whatsoever.
    See Also: Thread.run()
    public abstract void run();
}
```

```java
public class Thread implements Runnable {
    /* Make sure registerNatives is the first thing <clinit> does. */
    private static native void registerNatives();
    static {
        registerNatives();
    }
```

```java
    /* What will be run. */
    private Runnable target;
```

```java
    @Override
    public void run() {
        if (target != null) {
            target.run();
        }
    }
```

Runnable은 Functional Interface(구현하지 않은 메서드가 1개 뿐인 인터페이스, 해당 메서드는 앞에 abstract를 써서 명시해줌)

Runnable 안에 run()가 있지만 구현되어 있지 않음

Thread 클래스는 Runnable을 implements함

Thread는 target이라는 Runnbale타입 인스턴스를 생성하고 target에 있는 run()을 실행한다고만 정의 되어 있음

자바에서 스레드 사용 예시

방법1. Thread 클래스 extend

public void run() 오버라이드

extend되면 추가 상속이 안된다

방법2. Runnable 인터페이스 implement

public void run() 오버라이드

권장된 방법

방법3. 람다 표현식으로 Runnable 사용(JAVA 8부터 가능)

람다 표현식으로 Runnable 사용

클래스를 추가로 만들 필요가 없다

예제 코드

■ 방법 1: Thread 클래스 상속받기

```
class MyThread1 extends Thread {
    public void run() {
        try {
            while (true) {
                System.out.println("Hello, Thread!");
                Thread.sleep(500);
            }
        }
        catch (InterruptedException ie) {
            System.out.println("I'm interrupted");
        }
    }
```

무한반복문을 쓰는 이유는 여러 스레드를 사용하기 위함

```
public class ThreadExample1 {

    public static final void main(String[] args) {
        MyThread1 thread = new MyThread1();
        thread.start();
        System.out.println("Hello, My Child!");
    }
}
```

thread.satrt()로 extend한 클래스에서 오버라이드 된 run()실행

```
Hello, My Child!
Hello, Thread!
Hello, Thread!
Hello, Thread!
Hello, Thread!
Hello, Thread!
Hello, Thread!
Hello, Thread!
```

"Hello, My Child!"가 먼저 나오는 이유

- fork()의 과정과 비슷함
- thread.start()후에 아직 스레드 switching이 안일어남
- 현재 스레드 main()의 동작이 끝난 후 run()이 실행

▪ 방법 2: Runnable 인터페이스 구현하기

```
class MyThread2 implements Runnable {
    public void run() {
        try {
            while (true) {
                System.out.println("Hello, Runnable!");
                Thread.sleep(500);
            }
        }
        catch (InterruptedException ie) {
            System.out.println("I'm interrupted");
        }
    }
}

public class ThreadExample2 {

    public static final void main(String[] args) {
        Thread thread = new Thread(new MyThread2());
        thread.start();
        System.out.println("Hello, My Runnable Child!");
    }
}
```

방법1과 같은 방식이지만 extend가 아니고 implements로 Runnable의 run()을 구현한다

```
Hello, My Runnable Child!
Hello, Runnable!
Hello, Runnable!
Hello, Runnable!
Hello, Runnable!
Hello, Runnable!
Hello, Runnable!
Hello, Runnable!
```

- **방법 3: Runnable 람다 표현식 사용하기**

```
public class ThreadExample3 {
    public static final void main(String[] args) {
        Runnable task = () -> {
            try {
                while (true) {
                    System.out.println("Hello, Lambda Runnable!");
                    Thread.sleep(500);
                }
            }
            catch (InterruptedException ie) {
                System.out.println("I'm interrupted");
            }
        };
        Thread thread = new Thread(task);
        thread.start();
        System.out.println("Hello, My Lambda Child!");
    }
}
```

상속없이 Runnable인스턴스 task를 생성하고 안에 람다함수를 만들어 준다. 람다는 임시적으로 만든 함수 이기 때문에 이름도 없다.

람다로 만든 함수 task를 Thread에 넣어준다.

나머지는 다른 코드와 같다.

```
<terminated> ThreadExamples [Java Application] C:\Java\jdk
Hello, My Lambda Child!
Hello, Lambda Runnable!
Hello, Lambda Runnable!
Hello, Lambda Runnable!
Hello, Lambda Runnable!
Hello, Lambda Runnable!
Hello, Lambda Runnable!
```

스레드 대기: wait()와 비슷한 join()메소드

- **부모 쓰레드의 대기**: wait? join!

```
public class ThreadExample4 {
    public static final void main(String[] args) {
        Runnable task = () -> {
            for (int i = 0; i < 5; i++) {
                System.out.println("Hello, Lambda Runnable!");
            }
        };
        Thread thread = new Thread(task);
        thread.start();
        try {
            thread.join();
        }
        catch (InterruptedException ie) {
            System.out.println("Parent thread is interrupted");
        }
            System.out.println("Hello, My Joined Child!");
    }
}
```

방법3의 람다함수로 구현, 하지만 람다함수에는

위의 다른 예시들처럼 부모 스레드 완료 후 자식 스레드가 실행되지 않게 하는 방법

부모 스레드 실행중 자식 스레드 실행 후 나머지 부모 스레드 실행하는 방식

scope를 생각하면 이해가 편할듯

```
Hello, Lambda Runnable!
Hello, Lambda Runnable!
Hello, Lambda Runnable!
Hello, Lambda Runnable!
Hello, Lambda Runnable!
Hello, My Joined Child!
```

스레드 종료: stop()대신 interrupt()

- **쓰레드의 종료**: stop? interrupt!

```
public class ThreadExample5 {
    public static final void main(String[] args) throws InterruptedException {
        Runnable task = () -> {
            try {
                while (true) {
                    System.out.println("Hello, Lambda Runnable!");
                    Thread.sleep(100);
                }
            }
            catch (InterruptedException ie) {
                System.out.println("I'm interrupted");
            }
        };
        Thread thread = new Thread(task);
        thread.start();
        Thread.sleep(500);
        thread.interrupt();
        System.out.println("Hello, My Interrupted Child!");
    }
```

방법3으로 람다함수 구현

실행단계

1. thread.start()로 자식 스레드 시작
2. 자식 스레드는 println후 100ms 대기 ← 무한반복
3. 부모 스레드에 thread.sleep()로 500ms 대기
4. thread.interrupt()로 자식 스레드 중지

```
Hello, Lambda Runnable!
Hello, Lambda Runnable!
Hello, Lambda Runnable!
Hello, Lambda Runnable!
Hello, Lambda Runnable!
I'm interrupted
Hello, My Interrupted Child!
```

자식 스레드가 5번만 실행되는 이유

- 부모에서 500ms만큼 대기 후에 interrupt가 예약되어 있고 자식 스레드는 100ms 마다 println을 하기 때문에 500ms동안 실행할 수 있는 만큼 실행하고 interrupt로 종료가 된다.