<쓰레드>

쓰레드: 작업(Task)을 처리하는 단위

쓰는 이유: 하나의 프로그램에서 여러 작업을 할 수 있기 때문에 사용-> 멀티태스킹

쓰레드의 특징: 하나의 코어(작업공간)을 불규칙하게 번갈아가면서 사용함.

(스케쥴링이라고 특정 규칙이 있긴한데 예측하긴 어려움)

<ThreadTest.java>

0. 요약

- 쓰레드를 사용하는 다양한 방법 설명.
- 결국 다 같은 내용임

1. First (쓰레드를 상속받은 클래스)

```
First f1 = new First();
First f2 = new First();

f1.start();
f2.start();
```

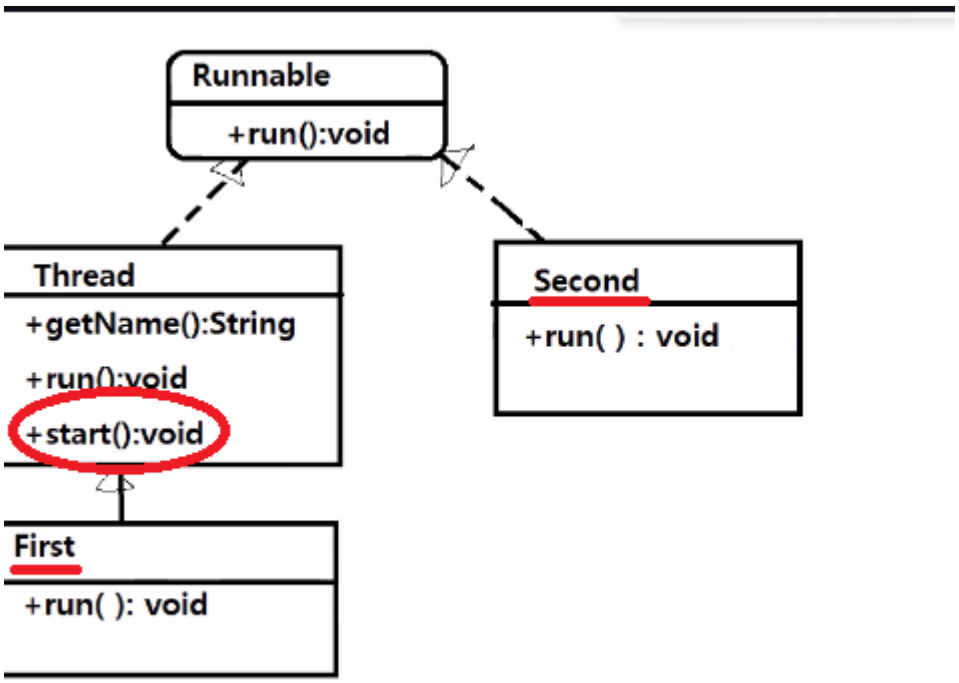
First클래스의 기능: 1부터 100까지 출력

```
Thread-0:83
Thread-0:84
Thread-1:38
Thread-0:85
Thread-0:86
Thread-1:39
Thread-1:40
Thread-1:41
```

- F1과 F2이 불규칙하게 번갈아가며 실행

2. Second (Runnable을 상속받은 클래스, start() 사용불가)

```
class Second implements Runnable {
```



- 클래스 Second는 run() (실행)은 가능하지만 대기상태로 올려두는 start()가 없음.

```
Second s1 = new Second();
Thread t1 = new Thread(s1);
t1.start();
```

- 쓰레드 형변환을 통해 사용

```java
@Override
public void run() {
    SimpleDateFormat sdf = new SimpleDateFormat("HH:mm:ss");

    for (int i = 1; i <= 10; i++) {
        System.out.println(sdf.format(new Date()));
        long mills = 1000; // 1ms = 1/1000초
        try {
            Thread.sleep(mills);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

}
```

<Second 코드>

Sleep(mills)로 쓰레드 1초간 일시정지 (1000ms = 1초)

-> 09:05:20
09:05:21
09:05:22
09:05:23
09:05:24

1초마다 실행을 하는 것이 아니라 1초쉬고 자기의 순번을 기다렸다 실행

대기손님이 많으면 1초쉬고 몇 초 더 기다렸다 실행할 수도 있음

3. Third, Fourth, Fifth (쓰레드의 역할을 이해시키기 위한 코드)

- Third의 역할: 캡션 PLAY 100번
- Fourth의 역할: Sound PLAY 100번
- Fifth의 역할: 동영상 PLAY 100번

- 각 클래스는 한 프로그램의 여러 기능(작업) 이라고 생각하면 됨.
- 실행결과는 매 실행마다 랜덤하게 바뀜

```
new Thread(new Third ()).start();
new Thread(new Fourth()).start();
new Thread(new Fifth ()).start();
```

- new Thread로 쓰레스를 생성해서 실행하는 방법
- 단점: 쓰레드의 상태 변경 불가 -> 별로 좋은 방법은 아니다.

4. 클래스없이 쓰레드 생성

```
new Thread(new Runnable() {
    public void run() {
        for (int i = 0; i < 10; i++)
            System.out.println("익명 클래스");
    }
}).start();

new Thread(()->{
    System.out.println("람다식형태의 쓰레드");
}).start();
```

- 클래스를 만들지않고 쓰레드를 만들어서 실행
- 장점: 간단한 기능이고 재사용을 안할 경우 코드가 간결해짐.

<ShareTest.java>

핵심내용: 동기화, 임계영역

수업 목적: 멀티쓰레드 프로그램에서는 하나의 자원(balance)을 여러 쓰레드가 공유해 서로에게 영향을 줄 수 있어서 한 쓰레드가 진행중인 작업을 다른 쓰레드가 간섭할 수 없도록 쓰레드를 동기화 시키는 방법을 알아보는 것

```
class Share {
    int balance;

    public void push() {
        for (int i = 0; i < 100; i++)
            balance++;
    }

    public void pop() {
        for (int i = 0; i < 100; i++)
            balance--;
    }
}
```

<Class Share>

- 필드 balance를 메서드 push, pop을 통해 증감작업.
- 클래스 Push, Pop은 하나의 객체 Share s의 Push와 Pop을 쓰레드로 사용하기 위해서 만듬

```
Share s = new Share();

Push ps = new Push(s);
Pop  po = new Pop(s);

ps.start();
po.start();
```

- Push ps와 Pop po는 객체 s만 참조.
- 각 클래스의 객체는 start(대기상태에 올려놓음) 후 run(실행)시 s의 push와 pop을 실행

```
before push:7 after push:8
before push:8 after push:9
before push:9before pop:2 after pop:9
 after push:10
before push:9 after push:10
```

<실행결과 중 일부분>

Push또는 pop실행중 다른 쓰레드가 작업공간을 점거

-> 쓰레드가 작업하는 동안 다른 쓰레드가 작업공간을 사용하지못하게하는 작업이 필요

이 작업을 동기화라고함.

[쓰는법]

```
synchronized (this) {
    System.out.print("before push:" + balance);
    balance++;
    System.out.println(" after push:" + balance);
}
```

- 중괄호 안에 있는 코드에 락을 걸어 다른 쓰레드가 임계영역(공유자원을 사용하는 공간)을 사용할 수 없도록 함.

```
synchronized public void pop() {
```

- 메서드에 락을 거는 방법

[일시정지, 일시정지 해제 후 시작]

```
this.wait();
this.notify();
```

- Wait은 try/catch로 해줘야 함.
- 이 부분은 깊은 내용이니깐 몰라도 됨.