Thread

▼ 멀티 스레드란?

하나의 프로세스에서 두 가지 일을 동시에 진행할 수 있도록 지원하는 것

예) 메신저라는 하나의 프로세스에서 파일 전송과 채팅이 가능하게끔 프로그래밍하는 것

▼ Thread 클래스 직접 생성

- Runnable 구현 객체를 매개값으로 갖는 생성자 호출 및 인터페이스 사용

```
// 1번. new 스레드 객체 생성 후 Runnable 인터페이스 사용
Thread thread = new Thread(Runnable target);

class Task implements Runnalbe {
@Override
public void run() {
// 스레드 실행할 코드

    }
}

// 2번. 람다식 사용하여 스레드 클래스 생성
Thread clock = new Thread(() -> {
    while (true){
        System.out.println(sdf.format(new Date()));

        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    }
});
clock.stat();
```

▼ Thread 를 상속받아 생성한다

주의 : Thread 클래스의 run( ) 메소드는 빈깡통이므로

사용자의 목적에 맞게 run( ) 메소드를 재정의하여 사용해야한다.

```
package ch13_thread;

// 1. Thread를 상속 시킴
public class MyThread extends Thread{

    private int range;

    public MyThread(int range) {
        this.range = range;
    }

    // 2. Thread 클래스 내에 있는 run  메소드를
    //    Override 한다.(= 멀티스레드로 던질 일에 해당)
    @Override
    public void run() {
        for(int i = range; i < range+5; i++){
            System.out.println(i);
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        }

    }
}
```

▼ Thread 상속 받아 생성

```
// 1. Thread를 상속 시킴
public class MyThread extends Thread{

    private int range;

    public MyThread(int range) {
```

```
        this.range = range;
    }

    // 2. Thread 클래스 내에 있는 run  메소드를
    //    Override 한다.(= 멀티스레드로 던질 일에 해당)
    @Override
    public void run() {
        for(int i = range; i < range+5; i++){
            System.out.println(i);
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        }

    }
}
```

▼ 스레드 상태

	sleep(long millis)
일시 정지	join( )
	wait( )
실행	start( )

	interrupt( )	일시 정지 상태인 경우 InterruptedExceiption을 발생시켜 강제 실행 대기 상태 또는 강제 종료 해버림
깨우기	notify( ) notifyAll( )	wait( ) 메소드로 인해 일시 정지 상태인 스레드를 실행 대기 상태로 만든다.
실행 대기	yield( )	실행 상태에서 다른 스레드에게 실행을 양보하고 실행 대기 상태가 된다 (continue( ) ; 같은 느낌)

▼ 스레드 상태 (깨우기, 일시정지) : notify( ), wait( )

```
public synchronized void makeFishCake() {
count++;
System.out.println("붕어빵을 하나 구웠습니다. 수량: " + count);
```

```
// wait set에서 대기중인 임의의 스레드를 하나 깨운다.
notify();

// wait set에서 대기중인 모든 스레드를 깨운다.
// -> 실용성이 낮음
//        notifyAll();
}

if (num > count) {
        try {
            // wait()를 실행한 스레드는
            // 다른 스레드에서 notify()로 깨울때까지
            // 대기 상태에 놓인다.
            wait();
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }

        // 대기에서 풀려나면 다시 구매시도
        buyFishCake(num);

    } else {
        count -= num;
        System.out.println("붕어빵을 " + num + "개 구매하였습니
        System.out.println("남은 수량: " + count);
    }
}
```

▼ 스레드 상태 (일시정지) : Thread.sleep(밀리초), wait( )

```
public class Chef extends Thread{


@Override
public void run() {
    // FishCake 내의 makeFishCake() 를
    // 2초마다 반복 실행 예정
```

```
    FishCake fishCake = FishCake.getInstance(); // 202호
    for(int i = 0; i < 10; i++){

        fishCake.makeFishCake();

        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }

    }


}
}
```

▼ 스레드 동기화 : 접근제어자 synchronized 리턴타입 메소드( ) { }

멀티 스레드들이 하나의 객체를 공유해서 작업하는 경우,

변경 사항이 생기면 다른 스레드에서도 자동적으로 변경이 되어야하므로 객체의 동기화를 위하여 synchronized를 붙여 메소드를 작성한다.

```
public synchronized void makeFishCake() {
    count++;
    System.out.println("붕어빵을 하나 구웠습니다. 수량: " + coun

    // wait set에서 대기중인 임의의 스레드를 하나 깨운다.
    notify();

    // wait set에서 대기중인 모든 스레드를 깨운다.
    // -> 실용성이 낮음
//        notifyAll();
}

// 붕어빵 구매 (= Customer 클래스가 사용 예정)
// num은 손님이 구매하고자 하는 수량
// count가 메소드에 따라 변화되어야하므로 sychronized 추가
public synchronized void buyFishCake(int num) {
```

```
    // 붕어빵의 수량이 구매하고자 하는 개수보다 적은 경우.
    // 해당 쓰레드를 대기시킨다.
    if (num > count) {
        try {
            // wait()를 실행한 스레드는
            // 다른 스레드에서 notify()로 깨울때까지
            // 대기 상태에 놓인다.
            wait();
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }

        // 대기에서 풀려나면 다시 구매시도
        buyFishCake(num);

    } else {
        count -= num;
        System.out.println("붕어빵을 " + num + "개 구매하였습니
        System.out.println("남은 수량: " + count);
    }
}

}
```

FishCakeMain.java

FishCake.java

Customer.java

Chef.java