21차 수업 복습 - Thread

2022.01.19

<경마장 시뮬레이션>
1. Thread를 가동시킬 Horse클래스 작성
- implements Runnable >> 스레드를 구동시킬 클래스에 반드시 작성
(- 혹은 구동시킬 클래스에 extends Thread를 써줘도 됨)
- private final static Random generator = new Random(); >> Math.random();과 같이 랜덤을 생성하는 또 다른 방법

```
public class Horse implements Runnable {
    private String horseName;
    private int waitingTime;

    private final static Random generator = new Random();
```

- generator.nextInt(bound:7777); >> nextInt(7777)안의 범주는 0~7776 즉, 7777개
Math.random();을 사용했을 때 range의 역할
10~7777를 표현하고싶을 땐
nextInt(7768) +10

```
public Horse (String name){
    horseName = name;
    waitingTime = generator.nextInt( bound: 7777);
}
```

- run() >> 스레드가 구동시킬 매서드
alt+insert -> interface 누르면 생성됨

- try~catch >> 자동으로 생성되긴 하지만 여기선 직접 작성

```java
// run() --> Thread가 구동시킬 메서드
@Override
public void run() {
    try{
        Thread.sleep(waitingTime);
    }catch (InterruptedException e){
        e.printStackTrace();
    }

    System.out.println(horseName + "가(이) 경주를 완료ㅎ
}
```

2. Thread가 가동될 RacingContestExample클래스 생성

- new Thread(new Horse("")); >> 스레드화 시킬 클래스를 객체화 시켜줌
  스레드가 구동되려면 new Thread()내부에 객체화시킬 클래스(여기선 Horse)를 넣어주면 됨
  그 클래스는 반드시 implements Runnable이 돼있어야함

```java
public class RacingContestExample {
    public static void main(String[] args) {
        Thread t1 = new Thread(new Horse( name: "적토마"));
        Thread t2 = new Thread(new Horse( name: "질풍마"));
        Thread t3 = new Thread(new Horse( name: "뇌전마"));
```

- start(); >> 만들어진 스레드 객체는 start()를 통해 구현됨. run()을 구동시킴으로써.

```java
t1.start();
t2.start();
t3.start();
```

<은행 시뮬레이션>
- lock()과 unlock()을 쓰기 전 먼저 "critical section"이란 걸 짚고 넘어가자면,
critical section이란 특정 자원을 동시에 접근하는 상황이 발생할 수 있는 모든 영역에 붙이는 이름이다. 두 개 이상의 스레드가 있을 때 그 스레드들이 공통으로 다루는 '전역(static) 변수'라고 볼 수 있다. 이 변수는 스레드가 두 개 이상있을 때 변경될 가능성이 있기 때문에 이를 보호하기 위해 방어벽 역할을 하는 lock()이 필요한 것

1. 2개의 스레드의 제어권을 통제하기 위한 NormalBank클래스 생성
(context switching역할을 함)
-Lock >> 각 thread의 제어권이 넘어가기 전에 critical section을 보호하기 위한 역할 생성
- lock = new ReentrantLock();

```
public class NormalBank {
    private static BigInteger money;
    private Lock lock;

  public NormalBank(){
      money = new BigInteger( val: "10000000000");
      lock = new ReentrantLock();
  }
```

- lock.lock(); >> 방어벽 활성화
- try~catch~finally >> finally는 catch에 걸리든 별 이상이 없든 무조건 실행하는 역할
맨 마지막에 놓는다
- lock.unlock(); >> 방어벽 해제
- get쓰는 이유 >> money가 private이기 때문

```
    public static BigInteger getMoney() {
        return money;
    }
}
```

2. 예금인지 출금인지 판별해줄 종업원 역할의 Worker클래스 생성(여기서run();생성)

```java
public class Worker implements Runnable{
    private NormalBank bank;

    private boolean depositOrWithdraw;
    private int count;
```

```java
    public Worker(NormalBank bank, boolean dow, int count){
        this.bank = bank;
        depositOrWithdraw = dow;
        this.count = count;
    }

    @Override
    public void run() {
        for(int i = 0 ; i < count ; i++){
            if(depositOrWithdraw){
                bank.deposit();
            } else {
                bank.withdraw();
            }
        }
    }
}
```

3. 스레드를 실행할 NormalBankSituationExample클래스 생성
- 스레드 객체 안에 run()이 들어있는 Worker클래스 객체화
- t1 >> 예금 처리하는 종업원(Worker클래스) 뽑아서 100번 일 시킴
- t2 >> 출금 처리하는 종업원(Worker클래스) 뽑아서 100번 일 시킴

```
public class NormalBankSituationExample {
    public static void main(String[] args) throws InterruptedException {
        NormalBank bank = new NormalBank();

        Thread t1 = new Thread(new Worker(bank, dow: true, count: 100));
        Thread t2 = new Thread(new Worker(bank,  dow: false,  count: 100));

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

- join(); >> 이걸 쓰면 main프로세는 스레드가 끝날 때까지 대기 한다
- NormalBank.getMoney(); >>money는 static변수이기 때문에 반환하려면
  클래스명.get() 의 형식으로 호출해야함(참조변수 사용x)

```
t1.join();
t2.join();

System.out.println("최종 결과는 : " + NormalBank.getMoney());
```