17강 멀티 스레드

목차

- 스레드의 개념
- 스레드의 상태
- 스레드의 동기화

프로세스 개념

- 실행중인 하나의 프로그램을 프로세스라고 부른다.
- 프로그램을 실행하면 운영체제로부터 필요한 자원을 받아서 필요한 코드를 실행하는 과정을 프로세스라고 한다.

멀티 프로세스의 개념

- 멀티프로세스는 동시에 여러가지 프로그램을 동작시키는 것을 의미한다.
- 예를 들면 게임을 하면서 동영상도 켜고 문서작업을 하면서 음악을 듣고 인터넷 검색도 하는 등 여러 프로그램들이 동시에 진행되는 것을 의미한다.

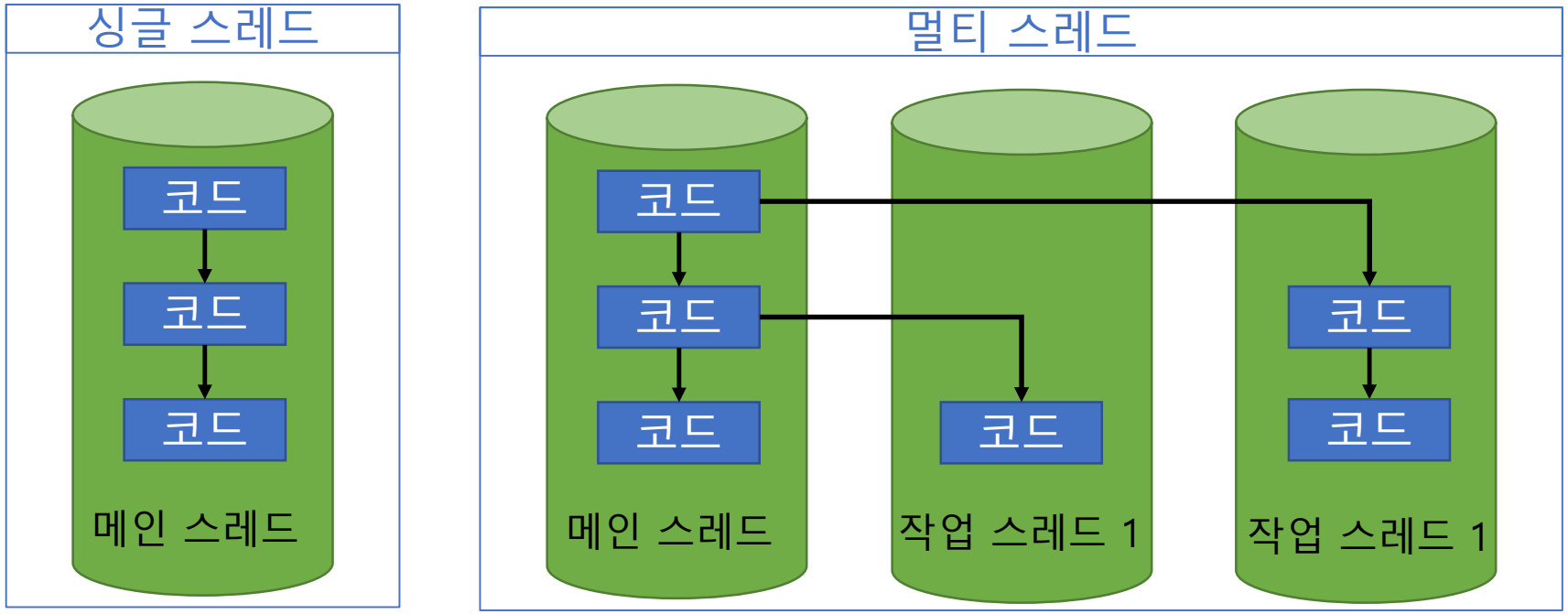
스레드 개념

- 스레드란 한 가닥의 실을 의미하는데 하나의 작업을 실행하기 위한 코드를 실처럼 엮어 놓았다고 해서 유래한 이름이다.
- 모든 프로그램은 기본적으로 하나의 프로세스에 하나의 스레드를 가진다.

- 자바도 마찬가지로 main메소드의 시작으로 프로그램이 시작되고 main메소드의 끝으로 프로그램이 끝난다.
- 이런 시작과 끝의 흐름이 하나의 스레드이고 이런 하나의 스레드로 프로그램이 완결되는 것을 싱글스레드라고 부른다.

멀티 스레드 개념

- 모든 자바 프로그램은 main메소드의 실행과 끝으로 이어지는 main 스레드를 가진다.
- 멀티 스레드는 main메소드의 실행이 진행중에 별도의 작업 메소드가 병렬로 처리가 되는 기술을 의미한다.
  - 자바는 멀티 스레드를 기본적으로 지원한다.



스레드 예제

- 메인 스레드만 사용하는 경우

- 그러나 위 for이 실행이 끝나고 이후에 비로소 아래 실행이 실행된다

```
public static void main(String[] args) {
    for(int i=0;i<5;i++) {
        System.out.println("딩");
        try {
            Thread.sleep(500);
        }catch(Exception e) {

        }
    }

    for(int i=0;i<5;i++) {
        System.out.println("동");
        try {
            Thread.sleep(500);
        }catch(Exception e) {

        }
    }

}
```

스레드 예제

- 멀티 스레드를 사용하는 경우
- Runnable 인터페이스를 구현한 클래스

```
public class DingDong01 implements Runnable{

    @Override
    public void run() {
        for(int i=0;i<5;i++) {
            System.out.println("동");
            try {
                Thread.sleep(500);
            }catch(Exception e) {

            }
        }
    }

}
```

```
public static void main(String[] args) {
    Runnable ddong = new DingDong01();
    Thread thread = new Thread(ddong);
    thread.start();//쓰레드를 실행하는 코드

    for(int i=0;i<5;i++) {
        System.out.println("딩");
        try {
            Thread.sleep(500);
        }catch(Exception e) {

        }
    }

}
```

```
딩
동
딩
동
딩
동
딩
동
딩
동
```

스레드 예제

- 멀티 스레드를 사용하는 경우
- 익명 구현 객체를 사용하는 경우

```
public static void main(String[] args) {
    Thread thread = new Thread(new Runnable() {
        @Override
        public void run() {
            for(int i=0;i<5;i++) {
                System.out.println("동");
                try {
                    Thread.sleep(500);
                }catch(Exception e) {

                }
            }
        }
    });
    thread.start();

    for(int i=0;i<5;i++) {
        System.out.println("딩");
        try {
            Thread.sleep(500);
        }catch(Exception e) {

        }
    }

}
```

스레드 예제

- 멀티 스레드를 사용하는 경우
- Thread 를 상속받은 객체를 통해서 쓰레드를 만드는 방법

```
public class DingDong02 extends Thread{
    @Override
    public void run() {
        for(int i=0;i<5;i++) {
            System.out.println("동");
            try {
                Thread.sleep(500);
            }catch(Exception e) {

            }
        }
    }

}
```

```
public static void main(String[] args) {
    Thread thread = new DingDong02();
    thread.start();

    for(int i=0;i<5;i++) {
        System.out.println("딩");
        try {
            Thread.sleep(500);
        }catch(Exception e) {

        }
    }
}
```

스레드 예제

- 멀티 스레드를 사용하는 경우
- 익명 자식 객체도 가능하다

```
public static void main(String[] args) {
    Thread thread = new Thread() {
        @Override
        public void run() {
            for(int i=0;i<5;i++) {
                System.out.println("동");
                try {
                    Thread.sleep(500);
                }catch(Exception e) {

                }
            }
        }

    };
    thread.start();

    for(int i=0;i<5;i++) {
        System.out.println("딩");
        try {
            Thread.sleep(500);
        }catch(Exception e) {

        }
    }
}
```

스레드의 이름

- 스레드 자체에는 이름을 필요하지 않지만 디버그시 어떤 스레드가 어떤 작업을 처리하는지 알아 둘 때 유용하다.

  - 이름설정
    thread.setName("이름");
  - 이름 알아보기
    thread.getName();

스레드의 우선순위

- 보통 각각의 스레드는 각각의CPU코어에서 담당해서 처리하게 된다.
- 그러나 코어의 개수보다 스레드가 많을 경우 어떤 순서에 의해서 스레드를 처리할 것인가가 문제가 되는데
- 이런 스레드 처리 방식을 스레드 스케쥴링이라고 한다.

스레드의 우선순위

- 스케쥴링 알고리즘은 2가지 방식을 동원한다.
  - 우선순위 방식 : 스레드마다 코드상에 우선순위를 부여한 다음 우선순위가 높은 스레드부터 처리하게 한다.
  - 시분할 방식 : 각 스레드 마다 시간을 할당해서 정해진 시간만큼 동작하고 시간이 지나면 일시 중지하고 다른 스레드로 실행 권한을 넘기는 방식이다.

  - 그러나 두번째에 해당하는 시분할 방식은 JVM에 의해서 결정되기 때문에 개발자가 코드로 제어할 수 없다.
  - 개발자가 코드로 제어할 수 있는 부분은 우선순위를 부여하는 방식이다.
    - 우선순위는 숫자로 표기되는데 1부터 10까지 부여된다.
    - 1은 가장 우선순위가 낮고 10이 가장 우선 순위가 높다.

스레드의 우선순위

- 예제

```
public class CalcThread extends Thread{
    public CalcThread(String name) {
        setName(name);
    }

    @Override
    public void run() {
        for(int i=1;i<=2000000000;i++) {// 쓰레드가 실행할 내용

        }
        System.out.println(getName());
    }

}
```

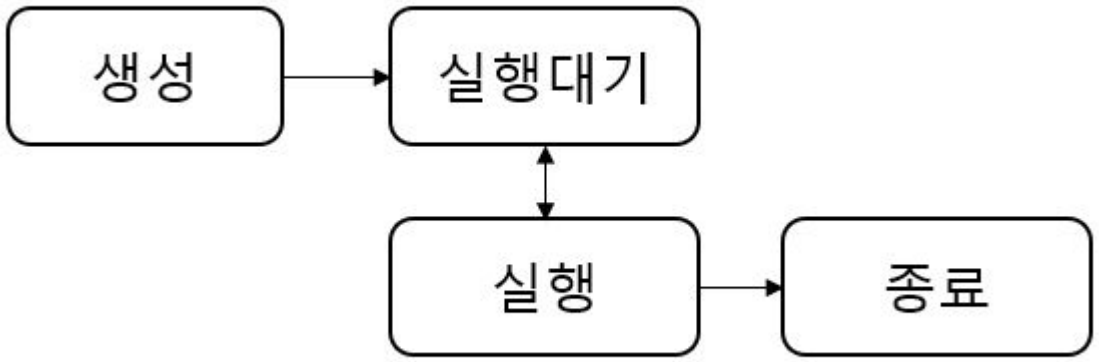
```
public static void main(String[] args) {
    for(int i=1;i<=10;i++) {
        Thread thread = new CalcThread("Thread"+i);
        if(i!=10) {
            thread.setPriority(1);
        }else {
            thread.setPriority(10);
        }
        thread.start();
    }

}
```

스레드의 상태

- 스레드는 실행중 이라고 해서 반드시 현재 실행중인 것은 아니다.
- 일시 중단이 될 때도 있고, 실행을 완료하는 경우도 있을 것이다.
- 이런 상태는 4가지가 존재한다.
  - 생성 : start()명령을 통해서 스레드를 실행한다.
  - 실행대기 : 다만 스레드를 실행한다 해서 바로 실행되는 것이 아닌 실행 대기로 들어간다. 스케쥴링이 이루어 지지 않거나 후순위를 기다리는 대기열이다.
  - 실행 : 스케줄링에 의해 실제 실행단계로 들어가면 스레드가 작동한다.
  - 종료 : 실행대기와 실행을 번갈아 가면 run()메소드를 실행하다가 메소드가 종료되면 스레드가 실행을 멈춘다.

스레드의 상태

- 예외적으로 실행에서 일시정지 상태로 가기도 하는데 일시정지는 스레드를 실행시킬 수 없는 상태이다.
- 일시정지에서 다시 실행하면 실행대기 상태로 가게 된다.



스레드의 상태

- 스레드의 우선순위나 시분할 스케쥴러에 의해
- 실행상태와 실행 대기 상태를 전환하게 된다.

- 그 중에 개발자의 코드에 따라 스레드의 실행을 잠시 멈추는 일시정지 상태를 만들 수도 있다

- 그리고 스레드의 모든 흐름이 마무리 되면 스레드의 종료 단계에 들어가게 되면서 스레드가 끝나게 된다.

- 스레드의 상태를 제어하는 각종 메소드들이 있다,

스레드의 상태

- 스레드의 상태를 제어하는 몇몇 메소드가 있다
  - sleep(); 주어진 시간동안 일시정지
  - yield(); 다른 스레드에게 실행을 양보
  - join(); 다른 스레드가 종료될 때 까지 기다림

  - wait()와 notify()는 동기화를 다룬 후에 확인하자

스레드의 상태

- 예제 : sleep();

```
public static void main(String[] args) {
    for(int i=0;i<10;i++) {
        System.out.println((i+1)+"번째 출력");
            try {
                Thread.sleep(2000);  //다음 실행까지 2초간의 대기 시간을 준다.
            } catch (InterruptedException e) {
            }
    }
}
```

스레드의 상태

- 예제 : yield();

```
public static void main(String[] args) {
    ThreadA threadA = new ThreadA();
    ThreadB threadB = new ThreadB();
    System.out.println("A,B 쓰레드 시작");
    threadA.start();   //threadA,threadB 모두 실행
    threadB.start();
    try {
        Thread.sleep(200);
    } catch (InterruptedException e) {

    }
    threadA.work = false;  //threadB 만 실행
    System.out.println("쓰레드 A 작업 종료");

    try {
        Thread.sleep(200);
    } catch (InterruptedException e) {

    }
    threadA.work = true;  //threadA,threadB 모두 실행

    try {
        Thread.sleep(200);
    } catch (InterruptedException e) {

    }
    threadA.stop=true;   //threadA,threadB 모두 종료
    threadB.stop=true;
}
```

```
public class ThreadA extends Thread{
    public boolean stop = false;
    public boolean work = true;

    public void run() {
        while(!stop) {   //stop이 true가 되면 while문 종료
            if(work) {
                System.out.println("ThreadA 작업 내용");
            }else {
                Thread.yield();// work가 false가 되면 다른 쓰레드에게 양보
            }
        }
        System.out.println("ThreadA 종료");
    }
}
```

```
public class ThreadB extends Thread{
    public boolean stop = false;
    public boolean work = true;

    public void run() {
        while(!stop) {   //stop이 true가 되면 while문 종료
            if(work) {
                System.out.println("ThreadB 작업 내용");
            }else {
                Thread.yield();// work가 false가 되면 다른 쓰레드에게 양보
            }
        }
        System.out.println("ThreadB 종료");
    }
}
```

스레드의 상태

- 예제 : join();

```
public static void main(String[] args) {
    SumThread sumThread = new SumThread();
    sumThread.start();

    try {
        sumThread.join();//main메소드 일시정지
    } catch (InterruptedException e) {
    }


    System.out.println("1~100까지의 합 : "+sumThread.getSum());
}
```

```
public class SumThread  extends Thread{
    private long sum;

    public long getSum() {
        return sum;
    }

    public void setSum(long sum) {
        this.sum = sum;
    }

    @Override
    public void run() {
        for(int i=1;i<=100;i++) {
            sum+=i;
        }
    }
}
```

스레드의 동기화

- 멀티 스레드란 기본적으로 각각의 흐름을 독립적으로 실행하는 것을 의미한다.
- 그러나 한 스레드가 사용중인 객체에 다른 스레드가 사용해야 하는 상황도 있을 수 있다.
- 그런 경우 데이터 변조로 인해서 원하는 결과가 아닌 엉뚱한 결과를 얻게 될 수 있다.

스레드의 동기화

- 예제:

```
public class User1 extends Thread{
    private Calc cal;

    public void setCal(Calc cal) {
        this.setName("User1");
        this.cal = cal;
    }

    @Override
    public void run() {
        cal.setMemory(100);
    }
}
```

```
public class User2 extends Thread{
    private Calc cal;

    public void setCal(Calc cal) {
        this.setName("User2");
        this.cal = cal;
    }

    @Override
    public void run() {
        cal.setMemory(50);
    }
}
```

스레드의 동기화

- 예제:

```
public static void main(String[] args) {
    Calc cal = new Calc();

    User1 user1 = new User1();
    user1.setCal(cal);
    user1.start();

    User2 user2 = new User2();
    user2.setCal(cal);
    user2.start();
}
```

```
public class Calc { //공유될 객체
    private int memory;

    public int getMemory() {
        return memory;
    }

    public void setMemory(int memory) {
        this.memory = memory;
        try {
            Thread.sleep(2000);
        }catch(Exception e){

        }
        System.out.println(Thread.currentThread().getName() + ":"+ this.memory);
    }
}
```

```
User2:50
User1:50
```

스레드의 동기화

- 위 예제를 보면

user1가 입력했던 100이 사라지고 user2가 입력한 50만 남는 것을 볼 수 있다

- 이처럼 두개 이상의 스레드가 하나의 데이터에 접근할 경우 코드가 꼬이면서 두 쓰레드 중에 하나의 결과만 기록되는 결과를 만들어 낼 수 있다
- 이런 현상을 막기 위해 동기화를 해야 하는데
- 간단히 설명하자면 하나의 스레드가 사용 중일 때는 다른 쓰레드가 사용할 수 없도록 잠금을 거는 행위를 말한다.

스레드의 동기화

- 멀티 스레드 프로그램에서 단 하나의 쓰레드만 사용할 수 있는 코드의 영역을 임계영역이라고 부른다.

- 자바에서는 임계영역을 설정하기 위해서 동기화 메소드/ 동기화 블록을 제공한다.

- 동기화 메소드를 만드는 방법은 synchronized 키워드를 붙이는 방법이다.
- 동기화 키워드 없는 메소드가 있다면 그냥 공유해서 사용가능 하다.

스레드의 동기화

- 동기화 예제

```
public class Calc { //공유될 객체
    private int memory;

    public int getMemory() {
        return memory;
    }

    public synchronized void setMemory(int memory) {
           // <= 이 메소드를 동기화 키워드를 붙여서 동기화 메소드로 사용한다,.
            this.memory = memory;
            try {
                Thread.sleep(2000);
            }catch(Exception e){

            }
            System.out.println(Thread.currentThread().getName() + ":"+ this.memory);
    }
}
```

```
User1:100
User2:50
```

스레드의 동기화

- 동기화 블록은 메소드 전체가 아닌 특정 블록을 잠그는 것을 말한다.

```
public void setMemory(int memory) {
    synchronized (this){
        // 공유 객체 Calc의 참조(잠금 대상)
        //실행코드가 동기화 블록에 들어가면 객체를 통으로 잠근다.
    this.memory = memory;
    try {
        Thread.sleep(2000);
    }catch(Exception e){

    }
    System.out.println(Thread.currentThread().getName() + ":"+ this.memory);
    }
}
```

스레드의 동기화

- 스레드를 번갈아 가면서 사용해야 하는 경우

- Object클래스의 메소드
  - wait(); 스레드를 일시정시상태로 만듦
  - notify(); 일시정지였던 스레드를 실행대기로 만듦

스레드의 동기화

- 예제:

```
public class ThreadA1 extends Thread{
    private WorkObject workObject;

    public ThreadA1(WorkObject workObject) {
        this.workObject=workObject;
    }

    @Override
    public void run() {
        for(int i=0;i<10;i++) {
            workObject.methodA1();
        }
        System.out.println("ThreadA1 작업 종료");
    }
}
```

```
public class ThreadB1 extends Thread{
    private WorkObject workObject;

    public ThreadB1(WorkObject workObject) {
        this.workObject=workObject;
    }

    @Override
    public void run() {
        for(int i=0;i<10;i++) {
            workObject.methodB1();
        }
        System.out.println("ThreadB1 작업 종료");
    }
}
```

스레드의 동기화

- 예제:

```
public static void main(String[] args) {
    WorkObject shareObject =new WorkObject();

    ThreadA1  threadA1 = new ThreadA1(shareObject);
    ThreadB1  threadB1 = new ThreadB1(shareObject);

    threadA1.start();
    threadB1.start();
}
```

```
public class WorkObject {
    public synchronized void methodA1() {
        System.out.println("ThreadA1의 methodA1() 작업 실행");
        notify();

            try {
                wait();
            } catch (InterruptedException e) {
            }
    }

    public synchronized void methodB1() {
        System.out.println("ThreadB1의 methodB1() 작업 실행");
        notify();

            try {
                wait();
            } catch (InterruptedException e) {
            }
    }
}
```