Java프로그래밍

12강. 멀티 스레드 프로그래밍 (교재 11장)

컴퓨터과학과 김희천 교수

한국방송통신대학교

오늘의 학습목차

1. 프로세스와 스레드
2. Thread 클래스
3. 스레드의 상태
4. 스레드 동기화

1. 프로세스와 스레드

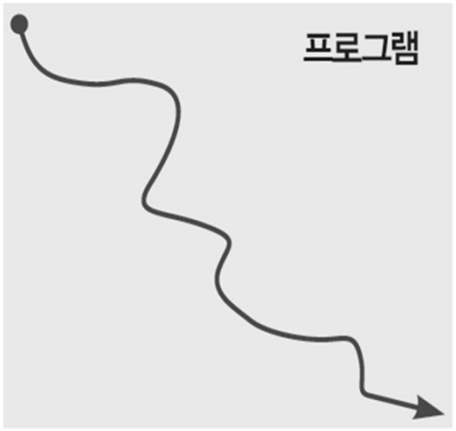
1) 스레드

◆ 프로세스와 스레드
- ✓ Java 프로그램은 하나의 프로세스로 만들어져 실행됨
- ✓ 프로세스는 실행 중인 프로그램
- ✓ 지금까지는 프로세스에서 하나의 스레드가 생성되고 main( ) 메소드가 호출되어 실행됨(단일 스레드)
- ✓ 스레드는 실행 중인 프로그램 내에 존재하는 소규모 실행 흐름
- ✓ 스레드는 경량 프로세스

◆ 멀티 스레드
- ✓ 하나의 프로세스 내부에서 여러 스레드가 만들어져 동시 실행될 수 있음

2) 멀티 스레드

- Java 프로그램은 하나의 스레드(main 스레드)로 시작됨
- main 스레드에서 자식 스레드를 만들어 시작시킬 수 있음
- 그러면 여러 스레드가 동시에 독립적으로 실행되고 종료됨



단일 스레드 프로그램

다중 스레드 프로그램

2. Thread 클래스

1) Thread 클래스

- 스레드의 생성과 관리를 위한 메소드를 제공
- 스레드 생성을 위해 Thread 유형의 객체가 필요함

생성자

- Thread( ), Thread(String name)
- Thread(Runnable target),
  Thread(Runnable target, String name)
  - ✓ Runnable 인터페이스를 구현하려면
    run( )을 구현해야 함

2) 스레드 생성과 실행

- Thread 유형의 객체 t를 생성
- t.start( )를 호출하면 스레드 실행이 시작됨
  - ✓ 이것은 run( ) 메소드를 호출함
- void run( ) 메소드에 스레드의 실행 코드가 있음
- run( )을 정의하는 두 가지 방법이 있음
  - ✓ 즉, 스레드를 실행시키는 두 가지 방법이 있음

3) 스레드 실행 방법1 – Thread 클래스의 상속

- Thread 클래스를 상속받는 클래스 A를 정의
  - ✓ 여기서 run( ) 메소드를 재정의
- A 유형의 객체를 생성하고 start( )를 호출함

```
class MyThread1 extends Thread {
   public void run( ) {
      for (int i = 0; i < 10; i++)
         System.out.println(getName( ));
   }
}
public class ThreadTest1 {
   public static void main(String args[ ]) {
      Thread t1 = new MyThread1( ); t1.start( );
      Thread t2 = new MyThread1( ); t2.start( );
      System.out.println("main");
   }
}
```

4) 스레드 실행 방법2 – Runnable 인터페이스를 구현

- Runnable 인터페이스를 구현하는 클래스 B를 정의
  - ✓ 여기서 run( ) 메소드를 구현
- B의 객체를 인자로 사용하여 Thread 유형의 객체를 생성하고 start()를 호출함

```
class MyThread2 implements Runnable {
   public void run( ) {
      for (int i = 0; i < 10; i++)
         System.out.println(Thread.currentThread( ).getName( ));
   }
}
public class ThreadTest2 {
   public static void main(String[ ] args) {
      Thread t1 = new Thread(new MyThread2( ), “ thd0 ” ); t1.start( );
      Thread t2 = new Thread(new MyThread2( ), “ thd1 ” ); t2.start( );
      System.out.println( “ main ” );
   }
}
```

5) 멀티 스레드의 실행

- 멀티 스레드 프로그램의 실행 결과는 예측할 수 없음
  - ✓ 실행 결과가 매번 다를 수 있음
- 각 스레드는 정해진 순서 없이 독립적으로 실행됨
- main 스레드는 다른 스레드를 시작시키나 다른 스레드의 실행과 무관하게 실행되고 종료됨

```
main
thd1
thd1
thd0
thd0
…
```

```
…
thd1
main
thd1
thd1
thd1
```

```
thd1
thd0
Thd0
main
thd0
…
```

3. 스레드의 상태

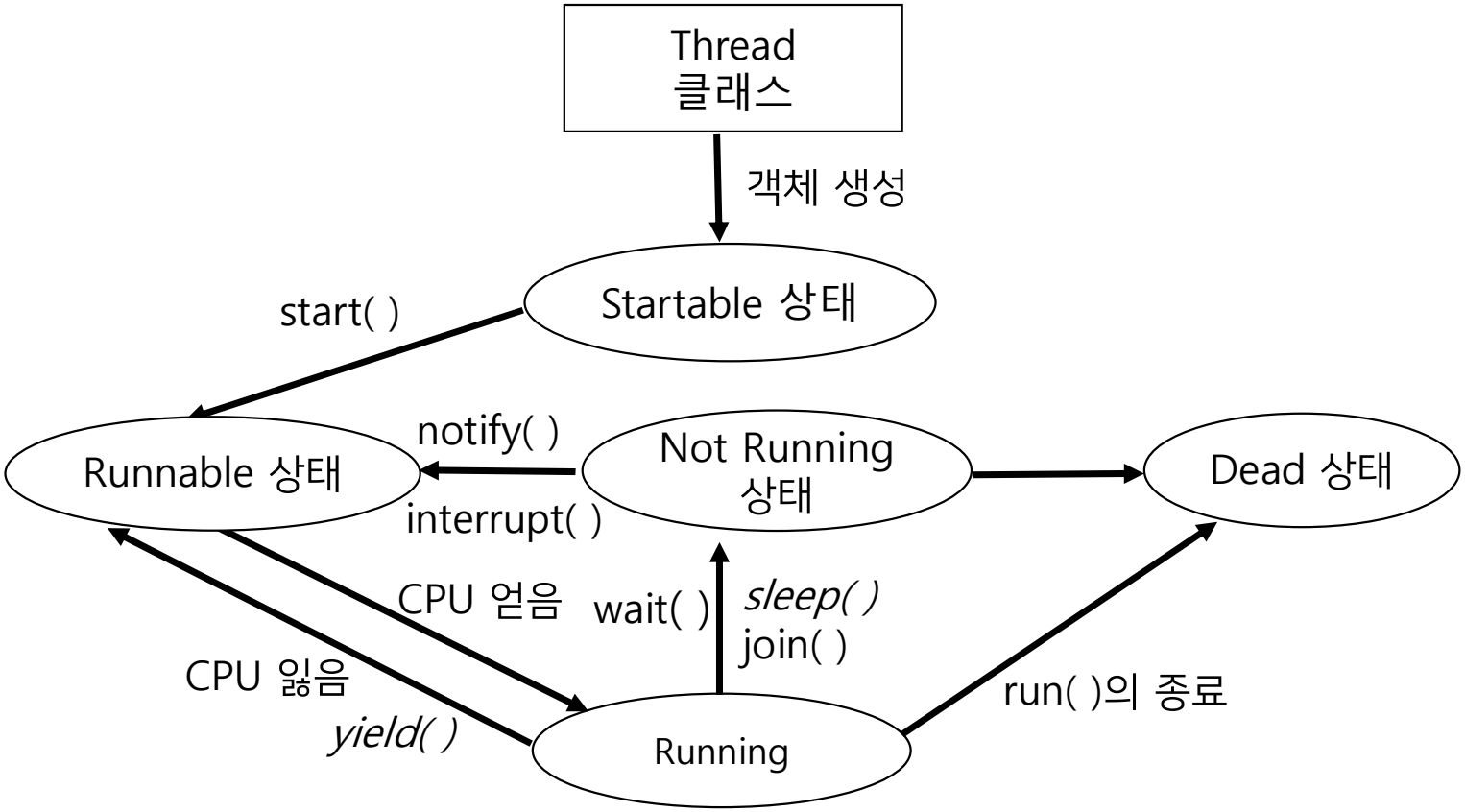
1) 스레드의 상태

- 보통 1개의 CPU를 사용하여 여러 스레드가 수행됨
- CPU를 얻어 실행되고 최종적으로 종료될 때까지 여러 상태 변화를 겪음

상태	설명
Startable	객체가 생성되었으나 start( )의 실행 전
Runnable	start( ) 메소드가 호출되었으나 CPU 획득 전
Running	CPU를 얻어 실행 중
Not Running	CPU를 잃고 중단된 상태 Blocked, Waiting, Timed_Waiting
Dead	run( ) 메소드가 종료된 상태

3. 스레드의 상태

2) 스레드의 상태 전이



3) 스레드의 상태 제어를 위한 메소드(1)

- void setPriority(int newPriority)
  - ✓ 스레드의 우선순위를 변경. 높은 우선순위를 가지는 스레드가 CPU를 얻을 확률이 높음
- static void sleep(long millis) throws InterruptedException
  - ✓ 현재 실행 중인 스레드가 정해진 시간 동안 실행을 멈추고 Not Running 상태로 들어감
- static void yield( )
  - ✓ 현재 실행중인 스레드가 잠시 실행을 멈추고 Runnable 상태로 들어감
  - ✓ CPU를 다른 스레드에게 양보하는 것

3) 스레드의 상태 제어를 위한 메소드(2)

- void join( ) throws InterruptedException
  - ✓ 스레드가 종료될 때까지 기다림
  - ✓ 현재 실행 중이었던 스레드는 Not Running 상태로 들어감
  - ✓ void join(long millis)는 최대 millis 시간 동안 기다림
  - ✓ 기다리는 중에 다른 스레드가 이 스레드를 깨워주면 InterruptedException을 받으면서 리턴됨
- void interrupt( )
  - ✓ 스레드를 인터럽트시킴
  - ✓ 스레드가 wait( ), join( ), sleep( )에 의해 중단된 상태였다면 그 상태에서 깨어나 Runnable 상태가 됨

4) 스레드 상태 제어를 위한 Object 클래스의 메소드

- void wait( ) throws InterruptedException
- void wait(long millis) throws InterruptedException
  - ✓ 객체를 처리 중인 스레드를 정해진 시간 동안 중지시킴
  - ✓ 다른 스레드가 해당 객체에 대해 notify( ) 메소드를 실행시켜 주면 이 스레드가 깨어날 수 있음
  - ✓ 이 메소드는 synchronized 메소드의 내부에서만 호출 가능
- void notify( )
  - ✓ wait( )를 호출하여 중단된 스레드를 깨워줌
  - ✓ 이 메소드는 synchronized 메소드의 내부에서만 호출 가능

5) 스레드의 상태 제어 예1

```
class MyThread1 extends Thread {
   public void run( ) {
      for (int i = 0; i < 1000; i++) {
         System.out.println(getName( ));
         Thread.yield( );
      }
   }
}
public class JoinTest2 {
    public static void main(String args[ ]) throws InterruptedException {
      Thread t1 = new MyThread1( ); t1.start( );
      Thread t2 = new MyThread1( ); t2.start( );
      t1.join( ); t2.join( );
      System.out.println("main");
   }
}
```

5) 스레드의 상태 제어 예2

```
class MyThread extends Thread {
   Thread thdNext = null;
   public MyThread(String szName)
              { super(szName); }
   public void run() {
      for(int i = 0;i < 100;i++) {
          try {
                Thread.sleep(1000000);
          } catch(InterruptedException e) {
              System.out.print(getName( ) + " ");
              if(thdNext.isAlive( ))
                    thdNext.interrupt( );
          }
      }
    }
   public void setNextThread(Thread t) {
            thdNext = t;
   }
}
```

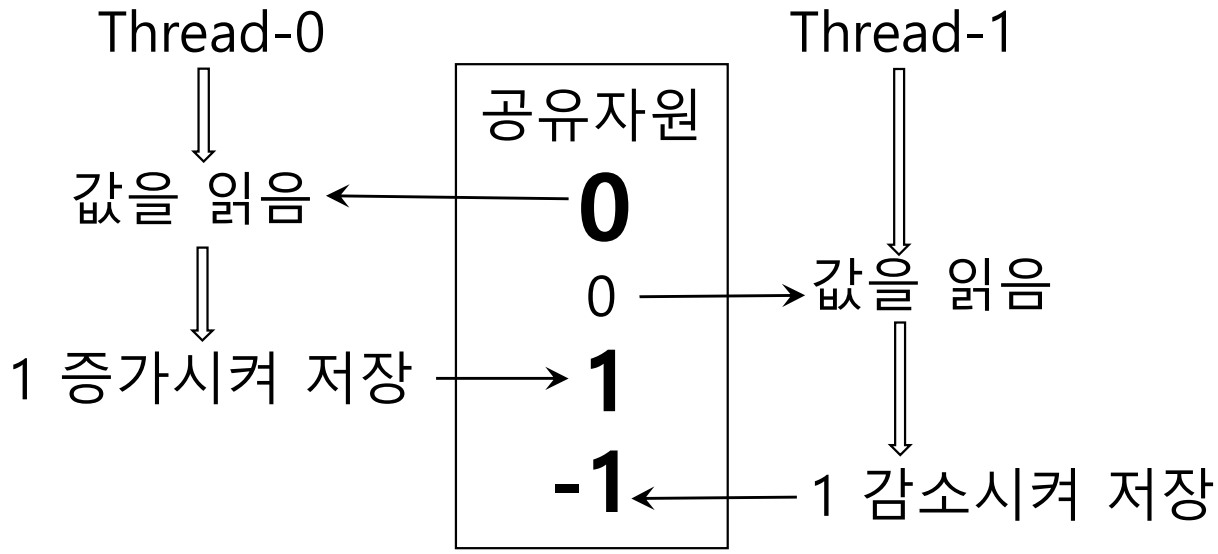
```
public class Test6 {
  public static void main(String args[]) {
     MyThread my_thread1 = new MyThread("thd1");
     MyThread my_thread2 = new MyThread("thd2");
     MyThread my_thread3 = new MyThread("thd3");
     my_thread1.setNextThread(my_thread2);
     my_thread2.setNextThread(my_thread3);
     my_thread3.setNextThread(my_thread1);
     my_thread1.start(); my_thread2.start();
     my_thread3.start( );
     try {
         my_thread1.interrupt();
         my_thread1.join(); my_thread2.join();
         my_thread3.join();
      } catch (InterruptedException e) {
        System.out.println(e); }
      System.out.println("main");
  }
}
```

4. 스레드 동기화

4. 스레드 동기화

1) 스레드 간의 간섭

◆ 여러 개의 스레드들이 하나의 공유 객체에 동시 접근하는 경우 일관성이 깨짐



2) 스레드 간의 간섭 예

```
class Counter {
    private int c = 0;
    public void increment( ) { c++; }
    public void decrement( ) { c--; }
    public int value( ) { return c; }
}

class MyThread3 implements Runnable {
    Counter c;
    public MyThread3(Counter c) {
       this.c = c;
    }
    public void run( ) {
       for(int i = 0;i < 100000;i++)
           c.increment( );
    }
}
```

```
class MyThread4 implements Runnable {
   Counter c;
   public MyThread4(Counter c) { this. c= c; }
   public void run( ) {
      for(int i = 0;i < 100000;i++)
          c.decrement( );
   }
}

public class ThreadTest3 {
   public static void main(String args[ ])  throws
InterruptedException {

     Counter c = new Counter( );
     Thread t1 = new Thread(new MyThread3(c));
     Thread t2 = new Thread(new MyThread4(c));
     t1.start( ); t2.start( );
     t1.join( ); t2.join( );
     System.out.println(c.value( ));
   }
}
```

3) 스레드 동기화

- 서로 다른 스레드들이 공유 자원을 다룰 때,
  일관성을 유지하도록 하는 것
- 한 번에 오직 한 개의 스레드만이 해당 공유 객체에
  접근하도록 동기화 함

방법

- 상호 배제 원칙
- 키워드 synchronized
  - ✓ 동기화 메소드 또는 동기화 블록을 제공
  - ✓ 공유 자원을 수정할 때, 다른 스레드에서
    같은 코드를 수행할 수 없게 함

4) synchronized 메소드

- 한번에 하나의 스레드에 의해서만 실행 가능
- synchronized 메소드를 실행하려면 메소드를 호출한 객체에 대한 lock을 얻어야 함
  - ✓ 다른 스레드는 동일 객체에 대해 synchronized 메소드를 실행할 수 없게 됨
  - ✓ public synchronized void func( ) { ... }
- 일부 블록만 동기화하는 것도 가능함
  - ✓ synchronized (객체) { ... }
  - ✓ 객체는 공유자원으로 대개 this

5) synchronized 메소드 사용 예

```
class Counter {
    private int c = 0;
    public synchronized void increment( ) { c++; }
    public synchronized void decrement( ) { c--; }
    public int value( ) { return c; }
}
```

```
class Counter {
    private int c = 0;
    public void increment( ) {
       synchronized(this) { c++; }
    }
    public void decrement( ) {
       synchronized(this) { c--; }
    }
    public int value( ) { return c; }
}
```

Java프로그래밍

다음시간안내

13강. AWT 컨트롤 클래스 (교재 12장)