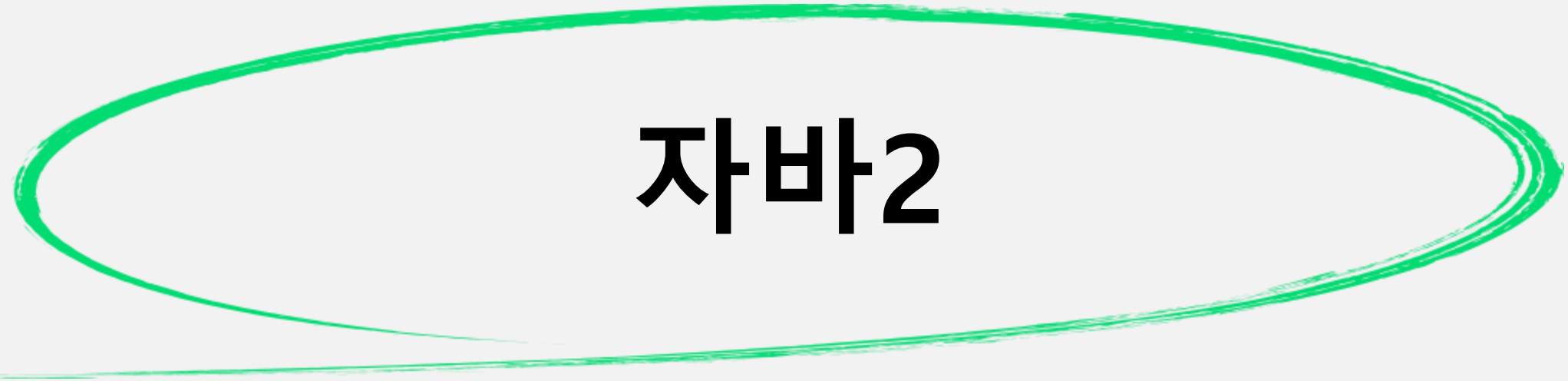


자바2

(스레드)

Chapter 09
스레드

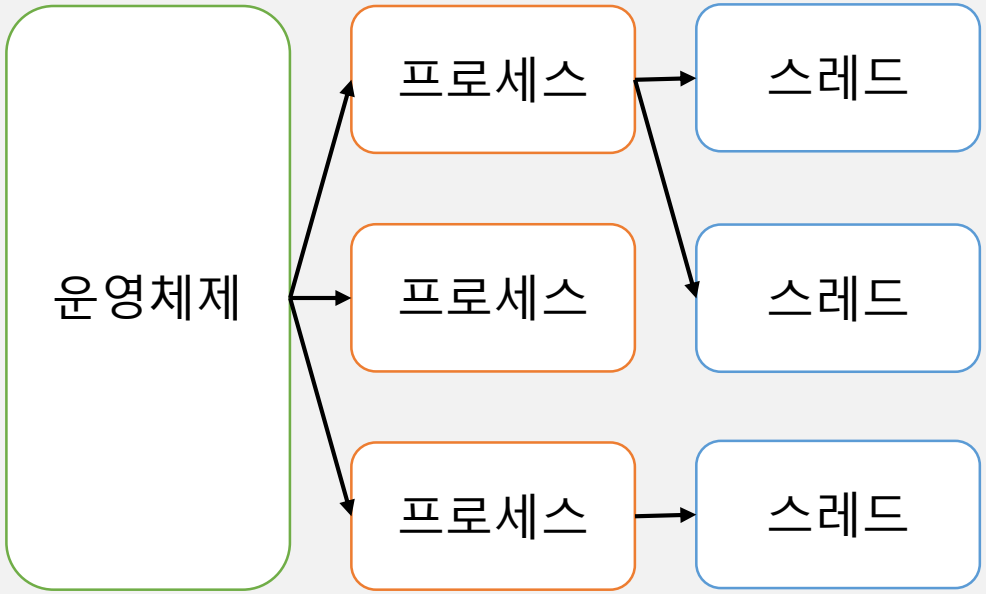
Chapter09의 학습목표

- 자바의 스레드 발생 방법에 대해 학습한다

스레드

스레드

- 하나의 프로세스 안에서 두 가지 이상의 일을 하도록 도와주는 것으로, 하나의 실행 흐름을 의미
- 멀티 태스킹(multi-tasking) : 여러 개의 어플리케이션을 동시에 실행하여서 컴퓨터 시스템의 성능을 높이는 기법
- 멀티 스레딩(multi-threading) : 병렬 작업의 아이디어를 하나의 어플리케이션 안으로 가져온 것.
  하나의 프로세스에서 여러 개의 스레드가 병행적으로 처리되는 것



스레드

프로세스와 스레드

- 컴퓨터에는 프로세스(process)와 스레드(thread) 라는 2가지 실행 단위가 있다
- 프로세스 : 자신만의 데이터를 가짐. 프로그램을 실행시켰을 때 메모리가 할당
  운영체제에서 실행중인 하나의 프로그램을 의미
- 멀티 프로세스(Multi-Process) : 두 개 이상의 프로세스가 실행되는 것
- 멀티 태스킹(Multi-Tasking) : 멀티 프로세스를 실행하여 일을 처리하는 것
- 스레드 : 동일한 데이터를 공유. 하나의 프로세스 안에 존재. 여러 개의 스레드가 모여서 하나의 프로세스를 이룸

프로세스가 실행되는 방식

1. 시간분할 방식 : 모든 프로세스에게 동일한 시간을 할당하고 골고루 실행
2. 선점형 방식 : 각각의 프로세스에게 우선순위를 부여하고 우선순위가 높은 순으로 실행되는 방식

✓ 웹 브라우저에서 웹 페이지를 보면서 동시에 파일을 다운로드할 수 있도록 한다.
✓ 워드 프로세서에서 문서를 편집하면서 동시에 인쇄한다.
✓ 게임 프로그램에서는 응답성을 높이기 위하여 많은 스레드를 사용한다.
✓ GUI에서는 마우스와 키보드 입력을 다른 스레드를 생성하여 처리한다.

스레드

스레드의 생성 1. Thread 클래스를 상속

- Thread 클래스를 상속받은 후에 run() 메소드를 재정의

```
class MyThread extends Thread {
    public void run() {
        for (int i = 10; i >= 0; i--)
            System.out.print(i + " ");
    }
}

public class ThreadTest {
    public static void main(String[] args) {
        Thread t = new MyThread();
        t.start();
    }
}
```

스레드

스레드의 생성 2. Runnable 인터페이스를 구현

1. Runnable 인터페이스를 구현한 클래스를 작성
2. run() 메소드를 작성
3. Thread 객체를 생성하고 이때 MyRunnable 객체를 인수로 전달
4. start()를 호출하여서 스레드를 시작.

```
class MyRunnable implements Runnable {
    public void run() {
        for (int i = 10; i >= 0; i--)
            System.out.print(i + " ");
    }
}

public class ThreadTest {
    public static void main(String[] args) {
        Thread t = new Thread(new MyRunnable());
        t.start();
    }
}
```

스레드

Thread클래스 상속 version + Runnable 인터페이스 구현 version 각각 실행

```
class MyThread extends Thread {
    public void run() {
        for (int i = 0; i < 10; i++) {
            System.out.println("스레드 진행중" + i);
        }
    }
}

class MyRunnable implements Runnable {
    public void run() {
        for (int i = 0; i < 10; i++) {
            System.out.println("러너블 진행중" + i);
        }
    }
}
```

```
public class ThreadTest {
    public static void main(String[] args) {
        MyThread t1 = new MyThread();
        MyRunnable t2 = new MyRunnable();
        Thread t = new Thread(t2);
        t1.start();
        t.start();
        for (int i = 0; i < 10; i++) {
            System.out.println("메인함수 진행중" + i);
        }
    }
}
```

스레드

Thread 클래스

메소드	설명
Thread()	매개 변수가 없는 기본 생성자
Thread(String *name*)	이름이 name인 Thread 객체를 생성한다
Thread(Runnable *target*, String *name*)	Runnable을 구현하는 객체로부터 스레드를 생성한다.
static int activeCount()	현재 활동중인 스레드의 개수를 반환한다.
String getName()	스레드의 이름을 반환
int getPriority()	스레드의 우선 순위를 반환
void interrupt()	현재의 스레드를 중단한다.
boolean isInterrupted()	현재의 스레드가 중단될 수 있는지를 검사
void setPriority(int *priority*)	스레드의 우선 순위를 지정한다.
void setName(String *name*)	스레드의 이름을 지정한다.
static void sleep(int *milliseconds*)	현재의 스레드를 지정된 시간만큼 재운다.
void run()	스레드가 시작될 때 이 메소드가 호출된다. 스레드가 하여야 하는 작업을 이 메소드 안에 위치시킨다.
void start()	스레드를 시작한다.
static void yield()	현재 스레드를 다른 스레드에 양보하게 만든다.

스레드

Thread의 우선 순위

- 스케쥴링(scheduling) : 하나의 작업을 여러 스레드가 나누어 쓰기 위해서 어떠한 원칙으로 어떤 순서로 스레드를 수행시킬 것 일가를 결정하는 작업
- 우선 순위(priority) : 자바에서의 스케쥴링 방법
- 스레드 생성과 동시에 Thread 클래스에 정의된 MIN_PRIORITY, MAX_PRIORITY 사이의 숫자인 우선순위를 배정받음
- 사용자는 각각의 스레드에 우선순위를 부여할 수 있음
- 인터럽트 (Interrupt) : 하나의 스레드가 실행하고 있는 작업을 중지하도록 하는 메커니즘으로, 스레드가 다른 스레드의 interrupt를 호출하면 해당 스레드가 중지됨

메소드명	설명
void setPriority(int newPriority)	우선순위를 결정
int getPriority()	우선순위를 반환
sleep(long millis)	밀리초 단위로 스레드를 쉬게 함 스레드가 수면 상태로 있는동안 인터럽트되면 InterruptedException(예외) 발생
join()	해당 스레드가 소멸될 때까지 기다리게 함
yield()	CPU를 다른 스레드에게 양보함 동일한 우선순위를 가지고 있는 다른 스레드를 실행시키고자 할 때 사용

스레드

Thread의 라이프 사이클

- 스레드의 상태가 변화하는 주기
- 스레드는 현재 상태에 따라 네 가지 상태로 분류 가능

상태	설명
new	키워드 new를 통해 인스턴스화된 상태 Runnable이 될 수 있는 상태이며 아직 대기열에 올라가지 않은 상태
Runnable	start()메소드가 호출되면 new상태에서 Runnable상태로 변경됨. 실행할 수 있는 상태로 대기 스케줄러에 의해 선택되면 run()메소드를 실행
Blocked	실행중인 스레드가 sleep(), join() 메소드를 호출하게 되면 Blocked 상태가 됨 스케줄러에 의해 선택받을 수 없는 상태
Dead	run()메소드의 실행을 모두 완료하면 Dead상태가 됨 할당받은 메모리와 정보가 삭제됨

스레드

Thread의 라이프 사이클 : sleep 예제 – Progressbar로 본 sleep

```
class MultiThread extends JPanel implements ActionListener{
    private JProgressBar progressBar1, progressBar2;
    private JButton startButton;
    //생성자
    public MultiThread() {
        startButton = new JButton("Start");
        startButton.addActionListener(this);
        progressBar1 = new JProgressBar(0, 1000);
        progressBar2 = new JProgressBar(0, 1000);

        progressBar1.setValue(0); //프로그래스바 value 설정
        progressBar1.setStringPainted(true); //문자열을 보이게 함
        progressBar2.setValue(0);
        progressBar2.setStringPainted(true);

        JPanel panel = new JPanel();
        panel.add(startButton);
        panel.add(progressBar1);
        panel.add(progressBar2);
        add(panel);
        setBorder(BorderFactory.createEmptyBorder(20,20,20,20));
    }
```

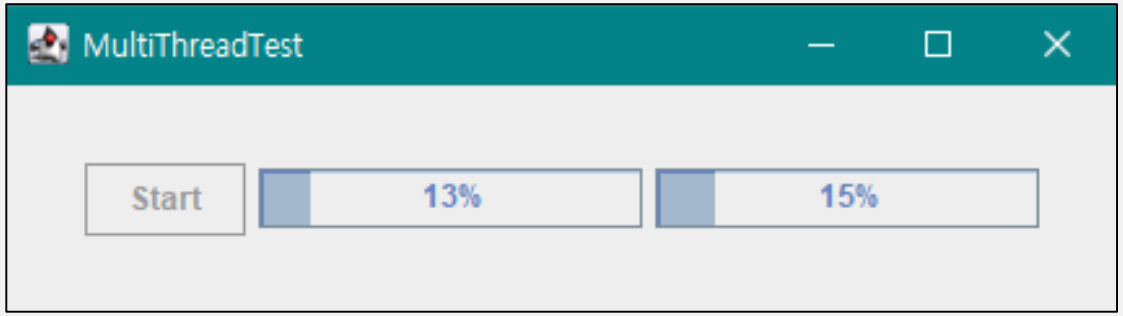
스레드

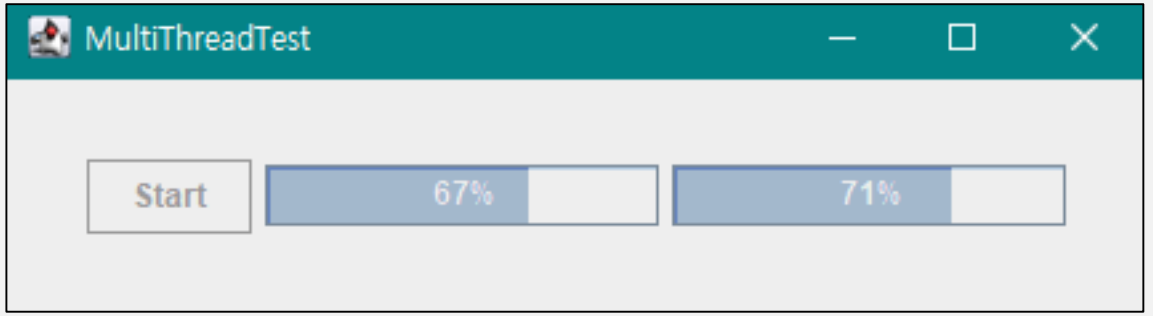
```
    @Override
    public void actionPerformed(ActionEvent e) {
        startButton.setEnabled(false);
        ProgressBarClass p1 = new ProgressBarClass(progressBar1, startButton);
        ProgressBarClass p2 = new ProgressBarClass(progressBar2, startButton);
        p1.start();
        p2.start();
    }
}
```

```
class ProgressBarClass extends Thread{
    private JProgressBar jpb;
    private JButton jb;
    public ProgressBarClass(JProgressBar jpb, JButton jb) {
        this.jpb = jpb;
        this.jb = jb;
    }
    @Override
    public void run() {
        int current = 0;
        while(current <= 1000) {
            current += Math.random() * 10;
            try {
                sleep(100);
            } catch (InterruptedException e) {
            }
            jpb.setValue(current);

        }
        jb.setEnabled(true);
    }
}
```

스레드

```
public class MultiThreadTest {
    public static void main(String[] args) {
        JFrame frame = new JFrame("MultiThreadTest");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        MultiThread pbd = new MultiThread();
        frame.add(pbd);
        frame.pack();
        frame.setVisible(true);
    }

}
```





스레드

Thread의 라이프 사이클 : sleep 예제

```
class MyThread extends Thread {
    public void run() {
        System.out.println("countdown-5 start");
        for (int i = 5; i >= 0; i--) {
            System.out.println(i);
            //0초인 경우에는 1초를 기다리지 않음
            if(i != 0) {
                try {
                    Thread.sleep(1000);
                }catch (InterruptedException ie) {
                    System.out.println(ie.toString());
                }
            }
        }
        System.out.println("종료");
    }
}
```

```
public class ThreadTest {
    public static void main(String[] args) {
        MyThread t1 = new MyThread();
        t1.start();
    }
}
```

```
countdown-5 start
5
4
3
2
1
0
종료
```

스레드

Thread의 라이프 사이클 : join 예제

```
class MyRunnable1 implements Runnable {
    public void run() {
        for (int i = 0; i < 10; i++) {
            System.out.println("t1" + i);
        }
        System.out.println("t1 완료");
    }
}

class MyRunnable2 implements Runnable {
    public void run() {
        for (int i = 0; i < 10; i++) {
            System.out.println("t2" + i);
        }
        System.out.println("t2 완료");
    }
}
```

```
MyRunnable1 r1 = new MyRunnable1();
MyRunnable2 r2 = new MyRunnable2();
Thread t1 = new Thread(r1);
Thread t2 = new Thread(r2);
t1.start();
try {
    //t1을 제외한 다른 스레드를 블록처리
    t1.join();
} catch (InterruptedException e) {
    System.out.println(e.toString());
}

t2.start();
try {
    //t1을 제외한 다른 스레드를 블록처리
    t2.join();
} catch (InterruptedException e) {
    System.out.println(e.toString());
}

for (int i = 0; i < 10; i++) {
    System.out.println("메인" + i);
}
```

스레드

Thread의 라이프 사이클 : yield 예제

```
class MyRunnable1 implements Runnable {
    public void run() {
        for (int i = 0; i < 30; i++) {
            System.out.print("O");
            Thread.yield();
        }
    }
}

class MyRunnable2 implements Runnable {
    public void run() {
        for (int i = 0; i < 30; i++) {
            System.out.print("X");
        }
    }
}
```

```
public class ThreadTest {
    public static void main(String[] args) {
        MyRunnable1 r1 = new MyRunnable1();
        MyRunnable2 r2 = new MyRunnable2();
        Thread t1 = new Thread(r1);
        Thread t2 = new Thread(r2);
        t1.start();
        t2.start();
    }
}
```

스레드

스레드 간섭 (Thread interference)

- 서로 다른 스레드에서 실행되는 두 개의 연산이 동일한 데이터에 적용되면서 서로 겹치는 현상

3개의 스레드가 동시에 티켓팅을 시도하는 경우

```
class Ticketing {
    int tiketNumber = 1;
    public void ticketing(){
        if(tiketNumber > 0) {
            System.out.println(Thread.currentThread().getName() + "가 티켓팅 성공");
            tiketNumber--;
        }
        else {
            System.out.println(Thread.currentThread().getName() + "가 티켓팅 실패");
        }
        System.out.println(Thread.currentThread().getName() + "가 티켓팅 시도 후 티켓수 :"
        + tiketNumber);
    }
}
```

스레드

스레드 간섭 (Thread interference)

```
class MyRunnable1 implements Runnable {
    Ticketing ticket = new Ticketing();
    public void run() {
        ticket.ticketing();
    }
}

public class ThreadTest {
    public static void main(String[] args) {
        MyRunnable1 r1 = new MyRunnable1();
        Thread t1 = new Thread(r1, "A");
        Thread t2 = new Thread(r1, "B");
        Thread t3 = new Thread(r1, "C");
        t1.start();
        t2.start();
        t3.start();
    }
}
```

```
A가 티켓팅 성공
C가 티켓팅 성공
B가 티켓팅 성공
B가 티켓팅 시도 후 티켓수 :-2
C가 티켓팅 시도 후 티켓수 :-1
A가 티켓팅 시도 후 티켓수 :0
```

스레드

스레드의 동기화 (Synchronization)

- 한 번에 하나의 스레드만이 공유 데이터를 접근할 수 있도록 제어하는 것
- 메소드의 선언부에 synchronized 키워드를 붙여 사용
- synchronized가 붙은 메소드는 자신에게 접근한 스레드에게 자신을 사용할 수 있는 권한인 lock을 부여함

```
public synchronized void 메소드명(){
    //...수행할 작업
}
```

```
public synchronized void ticketing(){....
```

```
A가 티켓팅 성공
A가 티켓팅 시도 후 티켓수 :0
C가 티켓팅 실패
C가 티켓팅 시도 후 티켓수 :0
B가 티켓팅 실패
B가 티켓팅 시도 후 티켓수 :0
```

스레드

wait()와 notify()

- 스레드를 대기시키는 wait() 메소드와 대기 중인 스레드를 깨우는 notify() 메소드를 통해 스레드를 제어 가능
- wait() : 스레드가 일단 lock을 반납하고 기다리게 하는 메소드
- notify() : 작업을 중단했던 스레드가 다시 lock을 얻을 수 있도록 깨우는 메소드

메소드명	설명
wait()	해당 스레드를 대기시킴 – notify()가 호출되어야 깨어남
wait(long timeout)	해당 스레드를 지정 시간만큼 대기시킴
notify()	해당 스레드를 깨움
notifyAll()	대기중인 스레드를 모두 깨움

스레드

wait()와 notify()

```
class Buffer {
    private int data;
    private boolean empty = true;
    public synchronized int get() {
        while (empty) {
            try {
                wait();
            } catch (InterruptedException e) {
            }
        }
        empty = true;
        notifyAll();
        return data;
    }
```

```
    public synchronized void put(int data) {
        while (!empty) {
            try {
                wait();
            } catch (InterruptedException e) {
            }
        }
        empty = false;
        this.data = data;
        notifyAll();
    }
}
```

스레드

wait()와 notify()

```
class Producer implements Runnable {
    private Buffer buffer;
    public Producer(Buffer buffer) {
        this.buffer= buffer;
    }
    public void run() {
        for (int i = 0; i < 10; i++) {
            buffer.put(i);
            System.out.println("생산자: " + i + "번 케익을 생산하였습니다.");
            try {
                Thread.sleep((int) (Math.random() * 100));
            } catch (InterruptedException e) {
            }
        }
    }
}
```

스레드

wait()와 notify()

```
class Consumer implements Runnable {
    private Buffer buffer;
    public Consumer(Buffer drop) {
        this.buffer= drop;
    }
    public void run() {
        for (int i = 0; i < 10; i++) {
            int data = buffer.get();
            System.out.println("소비자: " + data + "번 케익을 소비하였습니다.");
            try {
                Thread.sleep((int) (Math.random() * 100));
            } catch (InterruptedException e) {
            }
        }
    }
}
```

스레드

wait()와 notify()

```
public class ThreadTest {
    public static void main(String[] args) {
        Buffer buffer = new Buffer();
        new Thread(new Producer(buffer)).start();
        new Thread(new Consumer(buffer)).start();
    }
}
```

```
생산자: 0번 케익을 생산하였습니다.
소비자: 0번 케익을 소비하였습니다.
생산자: 1번 케익을 생산하였습니다.
소비자: 1번 케익을 소비하였습니다.
생산자: 2번 케익을 생산하였습니다.
소비자: 2번 케익을 소비하였습니다.
생산자: 3번 케익을 생산하였습니다.
소비자: 3번 케익을 소비하였습니다.
생산자: 4번 케익을 생산하였습니다.
소비자: 4번 케익을 소비하였습니다.
생산자: 5번 케익을 생산하였습니다.
소비자: 5번 케익을 소비하였습니다.
생산자: 6번 케익을 생산하였습니다.
소비자: 6번 케익을 소비하였습니다.
생산자: 7번 케익을 생산하였습니다.
소비자: 7번 케익을 소비하였습니다.
생산자: 8번 케익을 생산하였습니다.
소비자: 8번 케익을 소비하였습니다.
생산자: 9번 케익을 생산하였습니다.
소비자: 9번 케익을 소비하였습니다.
```

수고하셨습니다.