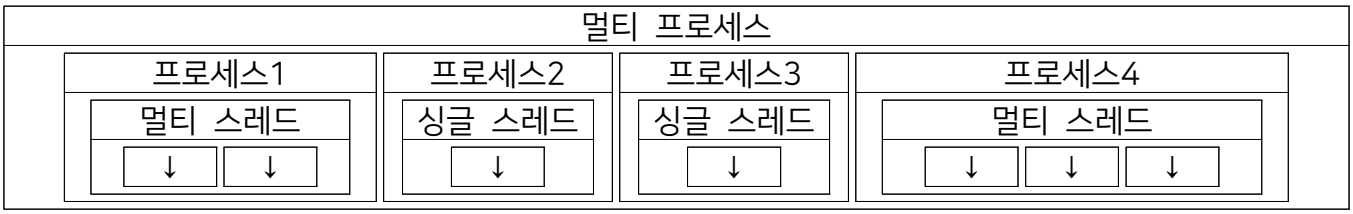
1. 멀티 스레드

- 애플리케이션 내부에서의 멀티 태스킹.



- 멀티 프로세스는 운영체제에서 할당받은 자신의 메모리를 가지고 실행하기 때문에 서로 독립적이다. 따라서 하나의 프로세스에서 오류가 발생해도 다른 프로세스에게 영향을 미치지 않는다. 하지만 멀티 스레드는 하나의 프로세스 내부에 생성되기 때문에 하나의 스레드가 예외를 발생시키면 프로세스 자체가 종료될 수 있어 다른 스레드에게 영향을 미치게 된다.

2. 메인 스레드

- 모든 자바 애플리케이션은 메인 스레드가 main() 메소드를 실행하면서 시작된다.
- 메인 스레드는 main() 메소드의 첫 코드부터 아래로 순차적으로 실행하고, main() 메소드의 마지막 코드를 실행하거나 return문을 만나면 실행이 종료된다.
- 메인 스레드는 필요에 따라 멀티 스레드를 생성해서 멀티 태스킹을 수행한다.
- 싱글 스레드 애플리케이션에서는 메인 스레드가 종료되면 프로세스도 종료된다. 하지만 멀티 스레드 애플리케이션에서는 실행 중인 스레드가 하나라도 있다면, 프로세스는 종료되지 않는다. 메인 스레드가 작업 스레드보다 먼저 종료되더라도 작업 스레드가 계속 실행 중이라면 프로세스는 종료되지 않는다.

3. Thread 클래스로부터 스레드 직접 생성

```
Thread thread = new Thread(Runnable target);
```

- java.lang.Thread 클래스로부터 작업 스레드 객체를 생성하려면 Runnable을 매개값으로 갖는 생성자를 호출해야 한다.
- Runnable은 작업 스레드가 실행할 수 있는 코드를 가지고 있는 객체이다. Runnable은 인터페이스 타입이기 때문에 구현 객체를 만들어 대입해야 한다. Runnable에는 run() 메소드 하나가 정의 되어 있는데, 구현 클래스는 run()을 재정의해서 작업 스레드가 실행할 코드를 작성해야 한다.

```
class Task implements Runnable{
   public void run(){
      스레드가 실행할 코드
   }
}


...
Runnable task = new Task();
Thread thread = new Thread(task);
```

```
...

...
Thread thread = new Thread(new Runnable(){
  public void run(){
    스레드가 실행할 코드;
  }
});
...



...
Thread thread = new Thread( () -> {
  스레드가 실행할 코드;
});
...



...
// Runnable 없이 Thread를 상속해서도 스레드를 사용할 수 있다.
public class WorkerThread extends Thread{
  @Override
  public void run(){
    스레드가 실행할 코드
  }
  Thread thread = new WokerThread();
}
...

...
Thread thread = new Thread(){
  public void run(){
    스레드가 실행할 코드;
  }
};
...
```

- 작업 스레드는 생성되는 즉시 실행되는 것이 아니라, start() 메소드를 실행해야 스레드가 시작된다.

```
thread.start();
```

4. 스레드의 이름

- 스레드는 이름을 가지고 있다.
- 이름이 큰 역할을 하는 건 아니지만 디버깅 할 때 어떤 스레드가 어떤 작업을 하는지 조사할 목적으로 쓰인다.
- 메인 스레드는 "main"이라는 이름을 가지고 있고, 우리가 직접 생성한 스레드는 자동적으로 "Thread-n"이라는 이름으로 설정된다.
- 스레드의 이름은 setter와 getter로 얻을 수 있다.

```
thread.setName("스레드 이름");
thread.getName();
```

- Thread.currentThread()를 사용하면 현재 코드를 실행하는 스레드 객체를 얻을 수 잇다.

5. 스레드 우선순위

- 스레드의 개수가 코어의 수보다 많을 경우, 스레드를 어떤 순서에 의해 동시성으로 실행할 것인가를 결정해야 하는데, 이것을 스레드 스케줄링이라 한다.
- 자바의 스레드 스케줄링은 우선순위 방식(Priority)과 순환할당(Round-Robin) 방식을 사용한다.
- 우선순위 방식은 우선순위가 높은 스레드가 실행 상태를 더 많이 가지도록 스케줄링 하는 것이다.
- 순환 할당 방식은 시간 할당량(Time Slice)을 정해서 하나의 스레드를 정해진 시간만큼 실행하고 다시 다른 스레드를 실행하는 방식을 말한다.
- 우선순위 방식은 스레드 객체에 우선순위 번호를 부여할 수 있기 때문에 개발자가 코드로 제어할 수 있다. 하지만 순환 할당 방식은 자바 가상 기계에 의해서 정해지기 때문에 코드로 제어할 수 없다.
- 우선순위는 Thread 클래스가 제공하는 setPriority() 메소드를 이요하면 된다.

```
thread.setPriority(우선순위);
```

- 우선순위는 1부터 10까지 부여되며 1이 가장 우선순위가 낮고 10이 가장 높다.
- 우선순위를 부여하지 않으면 기본적으로 5를 할당 받는다.
- 가독성을 위해 Thread 클래스의 클래스 상수를 사용할 수도 있다.

```
thread.setPriority(Thread.MAX_PRIORITY); // 10
thread.setPriority(Thread.NORM_PRIORITY); // 5
thread.setPriority(Thread.MIN_PRIORITY); // 1
```

6. 동기화 메소드와 동기화 블록

- 스레드가 사용 중인 객체를 다른 스레드가 변경할 수 없도록 하려면 스레드 작업이 끝날 때까지 객체에 잠금을 걸어서 다른 스레드가 사용할 수 없도록 해야한다.
- 멀티 스레드 프로그램에서 단 하나의 스레드만 실행할 수 잇는 코드 영역을 임계 영역(critical seciton)이라고 한다.
- 자바는 임계 영역을 지정하기 위해 동기화(synchronized) 메소드와 동기화 블록을 제공한다.
- 스레드가 객체 내부의 동기화 메소드 또는 블록에 들어가면 즉시 객체에 잠금을 걸어 다른 스레

드가 임계 영역 코드를 실행하지 못하도록 한다.

- 동기화 메소드는 메소드 전체 내용이 임계 영역이므로 스레드가 동기화 메소드를 실행하는 즉시 객체에는 잠금이 일어나고, 스레드가 동기화 메소드를 종료하면 잠금이 풀린다.

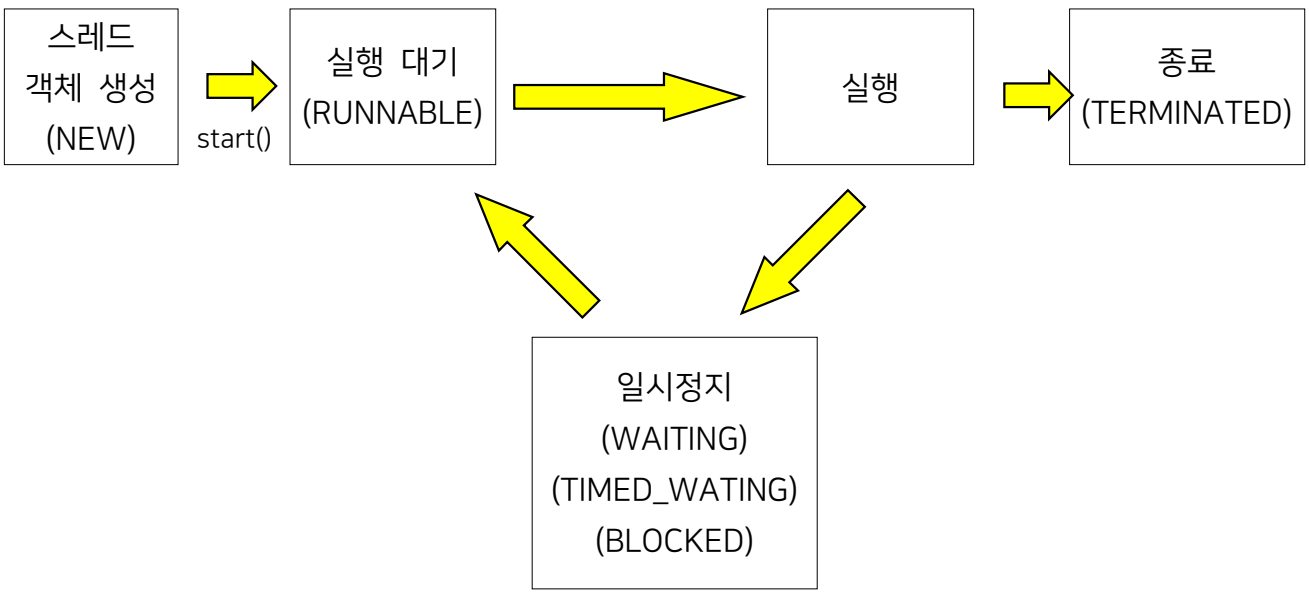
```
// 동기화 메소드
public synchronized void method(){
   임계 영역; // es 하나의 스레드만 실행
}
```

- 일부 내용만 임계 영역으로 만들고 싶다면 동기화 블록을 만들면 된다.

```
// 동기화 블록
public void method(){
   // 여러 스레드가 실행 가능 영역
   ...
   synchronized(공유객체){ // 공유 객체가 자신이면 this를 사용할 수 있다.
      임계 영역 // 단 하나의 스레드만 실행
   }
   // 여러 스레드가 실행 가능 영역
   ...
}
```

7. 스레드 상태

- 스레드는 생성, 대기, 실행, 일시정지, 종료 상태를 갖는다.



- 해당 상태는 Thread 클래스의 getState() 메소드를 통해 Thread.State 열거 상수를 리턴한다.

상태	열거 상수	설명
객체 생성	NEW	스레드 객체가 생성, 아직 start() 메소드가 호출되지 않은 상태
실행 대기	RUNNABLE	실행 상태로 언제든지 갈 수 있는 상태
일시 정지	WAITING	다른 스레드가 통지할 때까지 기다리는 상태
	TIMED_WAITING	주어진 시간 동안 기다리는 상태
	BLOCKED	사용하고자 하는 객체의 락이 풀릴 때까지 기다리는 상태
종료	TERMINATED	실행을 마친 상태

```
Thread.State state = targetThread.getState();
if(state == Thread.State.NEW){} // 객체 생성 상태일 경우
else if(state == Thread.State.TERMINATED){} // 종료 상태일 경우
```