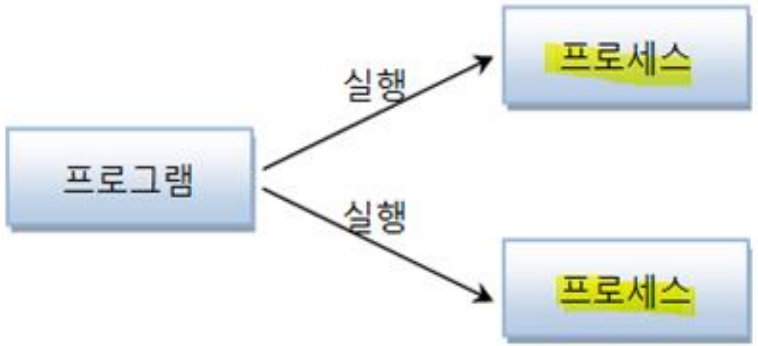
Thread

안 화 수

프로세스와 스레드

- 프로세스(process)

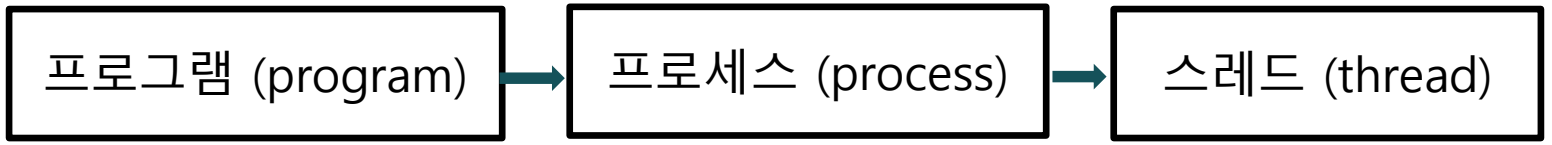
실행중인 하나의 애플리케이션을 프로세스(process)라고 한다.



프로세스와 스레드

- 스레드(Thread)

  하나의 프로세스를 구성하는 논리적인 작업 단위



- 스레드를 사용하는 경우

  1. 게임 프로그램
  2. 네트워크 프로그램
  3. 자바의 main() 메소드도 하나의 스레드 이다.

프로세스와 스레드

- 스레드(Thread)

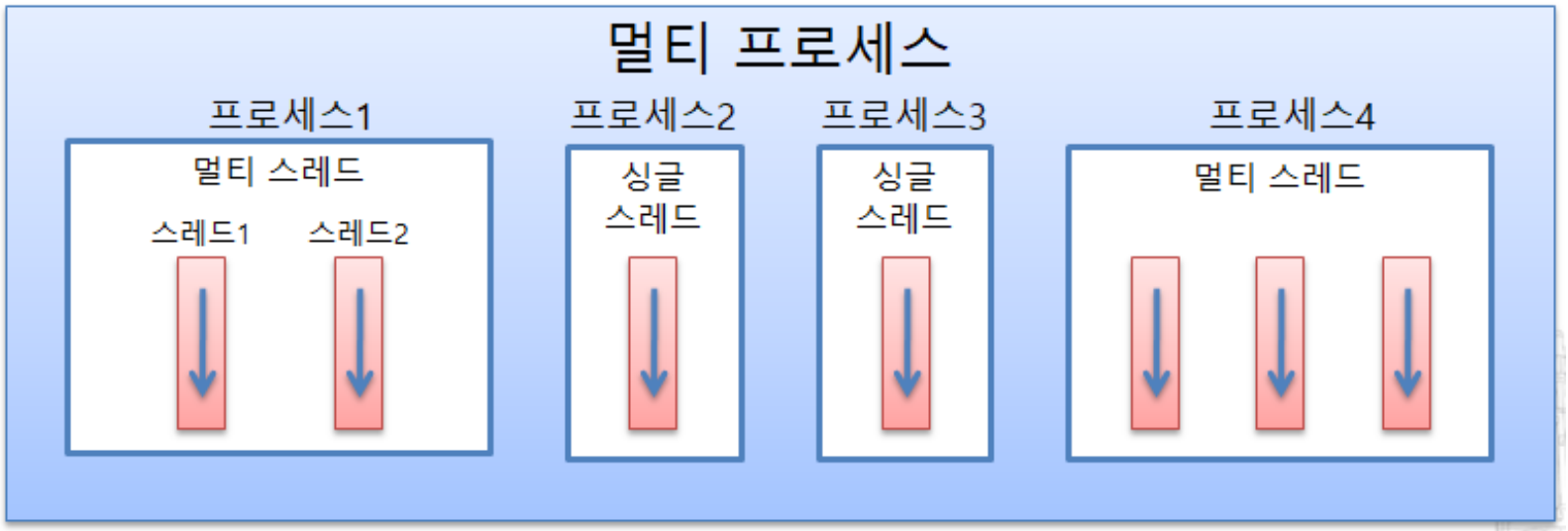
  하나의 프로세스를 구성하는 논리적인 작업 단위

- 멀티 프로세스

  독립적인 프로그램을 여러 개 실행하고 여러가지 작업을 처리

- 멀티 스레드

  한 개의 프로그램을 실행하고 내부적으로 여러가지 작업을 처리



스레드 생성방법

- 자바에서 스레드 생성 방법

1. Thread 클래스를 상속 받아서 만드는 방법

```
public class ThreadTest  extends  Thread
```

2. Runnable 인터페이스를 상속 받아서 만드는 방법

```
public class ThreadTest  implements  Runnable
```

스레드 생성방법

- 방법1. Thread 클래스를 상속 받아서 만드는 방법

```
public class ThreadEnd extends Thread {
    @Override
    public void run() {       // thread가 시작되면 실행되는 문장
        for (int i = 1; i <= 20; i++) {
            System.out.println("run number = " + i);
        }
    }
    public static void main(String[] args) {
        ThreadEnd tt = new ThreadEnd();

        tt.start();           // thread 실행

        for (int i = 101; i <= 120; i++) {
            System.out.println("-------> main number = " + i);
        }
    }
}
```

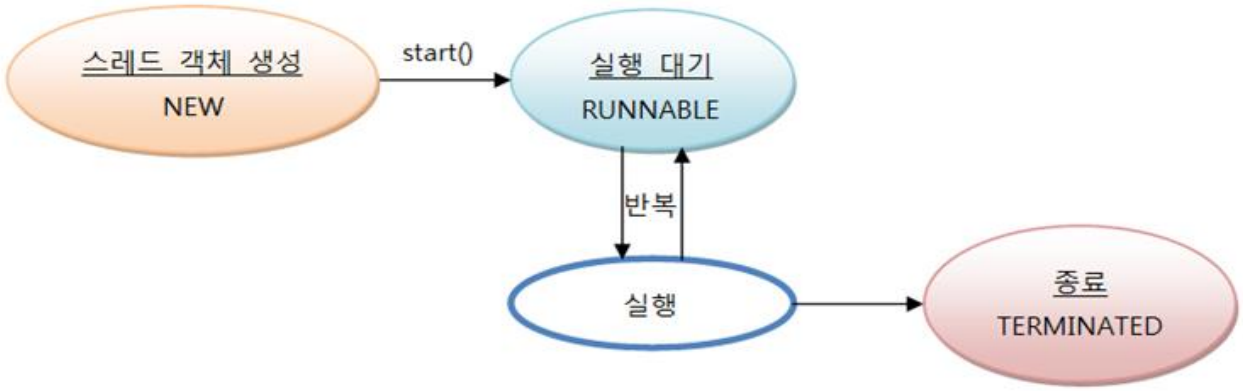
스레드 생성방법

- 방법2. Runnable 인터페이스를 상속 받아서 만드는 방법

```
public class RunnableTest implements Runnable {
    @Override
    public void run() {
        for (int i = 1; i <= 20; i++) {
            System.out.println("number = " + i);
        }
    }
    public static void main(String[] args) {
        RunnableTest tt = new RunnableTest();
        Thread t = new Thread(tt);
        t.start();        // thread 실행
        System.out.println("--------> main thread end");
    }
}
```

스레드의 생명주기

- Thread의 생명 주기(Life Cycle)



- Runnable(실행대기) : 새로 생성한 Thread를 start하면 Runnable 상태가 됨.
  실행 가능한 상태 (start() 메소드 호출한 상태)
- Running(실행) : CPU를 점유하고 run() 메소드 내의 명령문을 실행하는 상태.
  실행상태 (run() 메소드를 실행하는 상태)
- Blocked(일시정지) : 특정 메소드의 호출에 의해서 현재 실행중인 Thread가 CPU의 제어권을 잃어버린 상태.
- Teminated(종료) : run() 메소드의 명령 수행이 끝난 상태.

스레드의 생명주기

- Thread의 생명 주기(Life Cycle)

3개의 Thread가 running 상태와 blocked 상태를 번갈아 가면서 실행된다.

```
public class ThreadLife implements Runnable {
    @Override
    public void run() {
        for (int i = 1; i < 21; i++) {
            System.out.println(Thread.currentThread().getName() + " number = " + i);
        }
    }
    public static void main(String[] args) {
        ThreadLife tl = new ThreadLife();

        Thread first = new Thread(tl, "first1");
        Thread second = new Thread(tl, "second1");
        Thread third = new Thread(tl, "third1");

        second.start();
        first.start();
        third.start();
    }
}
```

스레드의 우선순위

- 자바의 스레드 스케줄링

  우선 순위(Priority) 방식과 순환 할당(Round-Robin) 방식 사용

- **우선 순위 방식 (코드로 제어 가능)**

  우선 순위가 높은 스레드가 실행 상태를 더 많이 가지도록 스케줄링

  1~10까지 값을 가질 수 있으며 기본은 5

- **순환 할당 방식 (코드로 제어할 수 없음)**

  시간 할당량(Time Slice) 정해서 하나의 스레드를 정해진 시간만큼 실행

스레드의 우선순위

- Thread의 우선 순위

  프로그램에서 우선 순위를 지정하지 않으면 우선 순위가 5로 설정되어 있다.

- 최고순위 - MAX_PRIORITY : 10
- 순위 미지정 - NORM_PRIORITY : 5
- 최저순위 - MIN_PRIORITY : 1

스레드의 우선순위

- Thread의 우선 순위 : 우선순위 – 기본값 5

```
public class ThreadPriority implements Runnable {
    @Override
    public void run() {
        for (int i = 1; i < 21; i++) {
            System.out.println(Thread.currentThread().getName() + " number = " + i);
        }
    }
    public static void main(String[] args) {
        ThreadPriority tl = new ThreadPriority();

        Thread first = new Thread(tl, "first1");
        Thread second = new Thread(tl, "second1");
        Thread third = new Thread(tl, "third1");
        System.out.println("first priority =" + first.getPriority());    // 5
        System.out.println("second priority =" + second.getPriority());  // 5
        System.out.println("third priority =" + third.getPriority());    // 5

        first.start();
        second.start();
        third.start();
    }
}
```

스레드의 우선순위

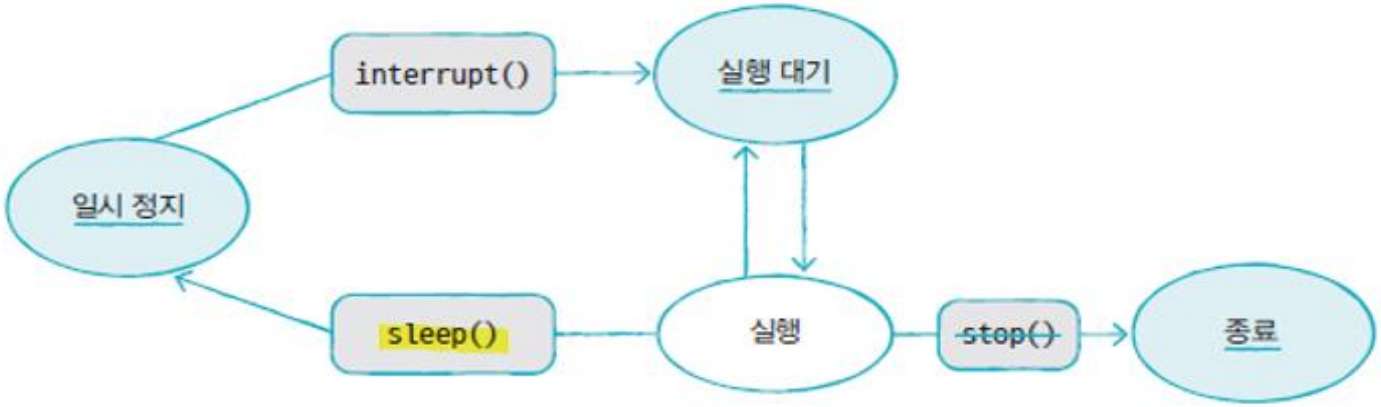
- Thread의 우선 순위 : 우선순위 – 기본값 5

```
public class ThreadPriorityControl implements Runnable {
    @Override
    public void run() {
        for (int i = 1; i < 11; i++) {
            System.out.println(Thread.currentThread().getName() + " number =" + i);
        }
    }
    public static void main(String[] args) {
        ThreadPriorityControl tl = new ThreadPriorityControl();

        Thread first = new Thread(tl, "first1");
        first.setPriority(Thread.MIN_PRIORITY);
        System.out.println("first priority = " + first.getPriority());     // 1

        Thread second = new Thread(tl, "second1");
        second.setPriority(Thread.MAX_PRIORITY);
        System.out.println("second priority = " + second.getPriority()); // 10

        Thread third = new Thread(tl, "third1");
        System.out.println("third priority = " + third.getPriority());    // 5

        first.start();
        second.start();
        third.start();
    }
}
```

스레드 상태제어

스레드 상태제어

실행중인 스레드의 상태를 변경



메소드	설명
interrupt()	일시 정지 상태의 스레드에서 InterruptedException을 발생시켜, 예외 처리 코드(catch)에서 실행 대기 상태로 가거나 종료 상태로 갈 수 있도록 합니다.
sleep(long millis)	주어진 시간 동안 스레드를 일시 정지 상태로 만듭니다. 주어진 시간이 지나면 자동적으로 실행 대기 상태가 됩니다.
stop()	스레드를 즉시 종료합니다. 불안전한 종료를 유발하므로 사용하지 않는 것이 좋습니다.

스레드 상태제어

- 스레드 상태제어

```
public class ThreadSleep implements Runnable {
    @Override
    public void run() {
        for (int i = 1; i < 10; i++) {
            System.out.println(Thread.currentThread().getName() + " : " + i);
            try {
                // sleep() 메소드를 사용해서 프로그래머가 강제로 blocked 상태로 변경
                // 1초 동안 스레드를 blocked 상태로 변경
                Thread.sleep(1000); // 단위 : 1/1000초
            } catch (InterruptedException ie) {
                System.out.println(ie.toString());
            }
        }
    }
    public static void main(String[] args) {
        ThreadSleep ts = new ThreadSleep();

        Thread first = new Thread(ts, "first1");
        Thread second = new Thread(ts, "second1");
        first.start();
        second.start();
    }
}
```

스레드의 동기화

❖ 스레드의 동기화

멀티 스레드 프로그램에서 한 번에 하나의 스레드만 객체에 접근할 수 있도록 객체에 락(lock)을 걸어서 데이터의 일관성을 유지하는 것.

❖ 동기화 메소드 및 동기화 블록

- 동기화 메소드

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

- 동기화 블록

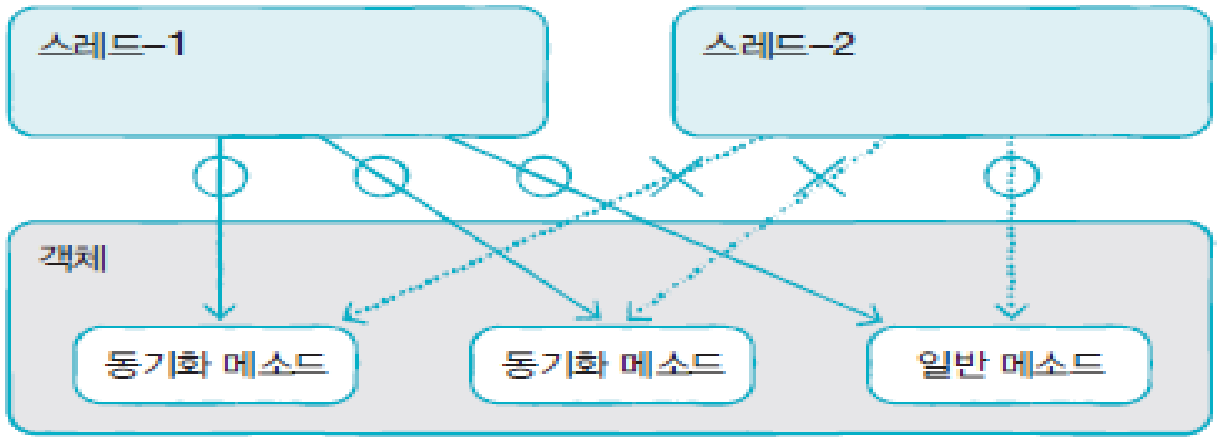
```
public void method () {
  //여러 스레드가 실행 가능 영역
  ...
  synchronized(공유객체) {
    임계 영역 //단 하나의 스레드만 실행
  }
  //여러 스레드가 실행 가능 영역
  ...
}
```

스레드의 동기화

❖ 동기화 메소드

- 스레드가 사용 중인 객체를 다른 스레드가 변경할 수 없게 하려면 스레드의 작업이 끝날 때까지 객체에 잠금 걸어야 함
- 임계 영역 (critical section) : 단 하나의 스레드만 실행할 수 있는 코드 영역
- 동기화 (synchronized) 메소드 : 스레드가 객체 내부의 동기화 메소드 실행하면 즉시 객체에 잠금이 설정 되어서 다른 스레드가 접근할 수 없다.

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



스레드의 동기화

- 스레드의 동기화 예. (1/3)

```
public class ManageToilet {

    public static void main(String[] args) {
        Toilet t = new Toilet();

        // thread 생성
        Family father = new Family("아버지", t);
        Family mother = new Family("어머니", t);
        Family sister = new Family("누나", t);
        Family brother = new Family("형", t);
        Family me = new Family("나", t);

        father.start();
        mother.start();
        sister.start();
        brother.start();
        me.start();
    }
}
```

스레드의 동기화

- 스레드의 동기화 예. (2/3)

```
public class Family extends Thread {

    Toilet toilet;
    String who;
    boolean key; // 초기값: false

    public Family(String name, Toilet t) {
        who = name;
        toilet = t;
    }

    public void run() {
        toilet.openDoor(who, key);
    }
}
```

스레드의 동기화

- 스레드의 동기화 예. (3/3)

```
public class Toilet {
    // 동기화 메소드
    public synchronized void openDoor(String name, boolean b) {
        if (b == false) {
            System.out.println(name);
            usingTime();
            System.out.println("아~~~~! 시원해");
            System.out.println();
        } else {
            System.out.println("사용중");
        }
    }
    public void usingTime() { // 화장실 사용하는 시간
        for (int i = 0; i < 1000000000; i++) {
            if (i == 10000) {
                System.out.println("끄으응");
            }
        }
    }
}
```