멀티쓰레드

동기 / 비동기 프로그램

- 동기 : 싱글쓰레드 환경에서는 하나의 흐름을 가지고 프로그램이 실행된다.

이해를 돕는 예제) hi^^ 출력이 완료되어야 bye~가 출력된다.

```
for( int i=0; i<100;i++)
                System.out.println( i+ "hi ^^ ") ;

for( int i=0; i<100;i++)
                System.out.println( i+ " bye~ ") ;
```

- 비동기 : 멀티쓰레드에서 여러 개의 흐름을 가지고 프로그램이 실행된다 (
  (우리가 사용하는 대부분의 프로그램이 비동기로 만들어져 있다.)
  하나의 프로그램에 여러 개의 흐름을 가지고 있다.
  (웹브라우저 : 다운로드 받고, 음악도 들으면서, 검색도 하고 있다.)

이해를 돕는 예제) hi^^ 와 bye~가 출력이 비동기로 실행되도록 한다.
별도의 흐름을 만들기 위해서는 쓰레드를 작성하여 실행하여야 한다.

```
public class 비동기예제기본{

	public static void main(String[] args){

		Thread t = new Thread( new Runnable() {
				public void run(){
				for( int i=0; i<100;i++)
					System.out.println( i+"hi ^^") ;
				}
			} );
		t.start();

		for( int i=0; i<100;i++)
		System.out.println( i+" bye~") ;
	}

}
```

Thread 만드는 법

1.Thread 상속

```
class A  extends Thread{
        void run(){
                for( int i=0; i<100;i++)
                  System.out.println( i+ " 실행할 코드  ") ;
        }
}
A a = new A();
a.start();
```

2.Runnable인터페이스 구현

```
Thread t = new  Thread( new Runnable() {

        void run(){
                for( int i=0; i<100;i++)
                  System.out.println( i+ " 실행할 코드2  ") ;
        }

} ) ;

t.start();
```

Call stack

함수 호출과 관련된 정보를 저장하는 자료 구조인 "호출 스택" 또는 "콜 스택 "이라고 함

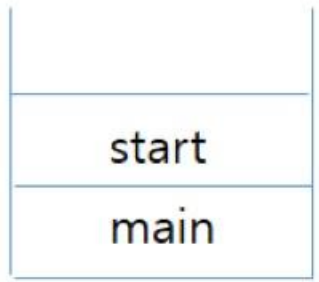
프로그램의 실행 흐름과 함수 호출 순서를 추적하기 위해 사용된다.
함수 호출 및 반환을 관리하고 실행하는 데 사용됨
호출 스택은 여러 함수가 중첩되어 호출될 때 현재 실행 중인 함수와 이전에 호출된 함수들의 관계와 순서를 유지한다.

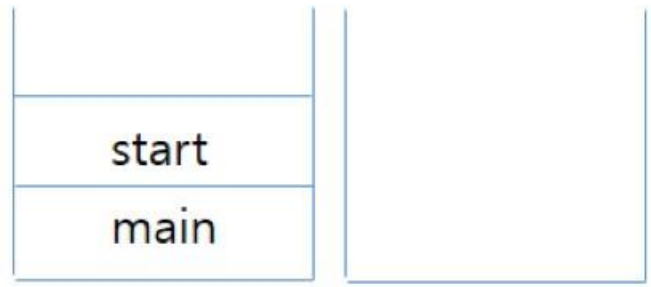
stack
선입후출
먼저들어간 것이 나중에 꺼내지는 자료구조

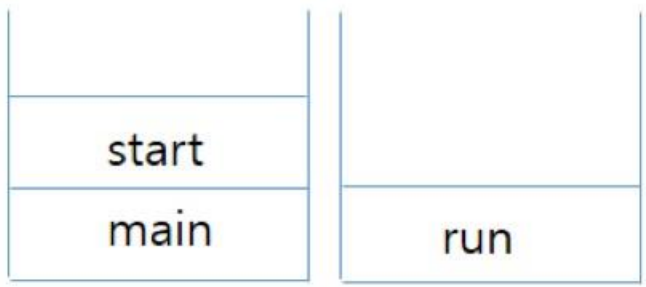
```
class Main{

    void 매서드1(){
        매서드2();
        System.out.println( "매서드1" );
    }
    void 매서드2(){
        System.out.println( "매서드2" );
    }


    public static void main(String[] args){
        Main m = new Main();
        m.매서드1();
        System.out.println( " main bye  ");
    }

}
```

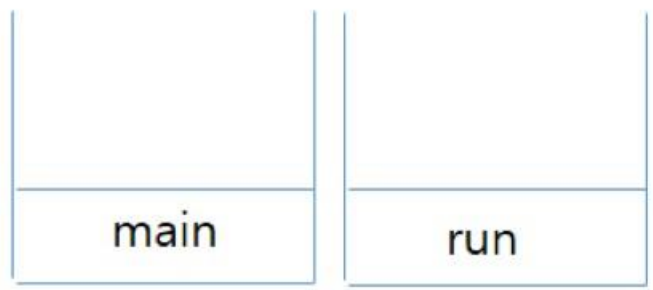
Thread 실행 =>
스레드를 생성하였다고 실행되는것은 아니다
Start()를 호출해야만 스레드가 실행된다

스레드가 start()되면
새로운 호출스택이 마련된다.

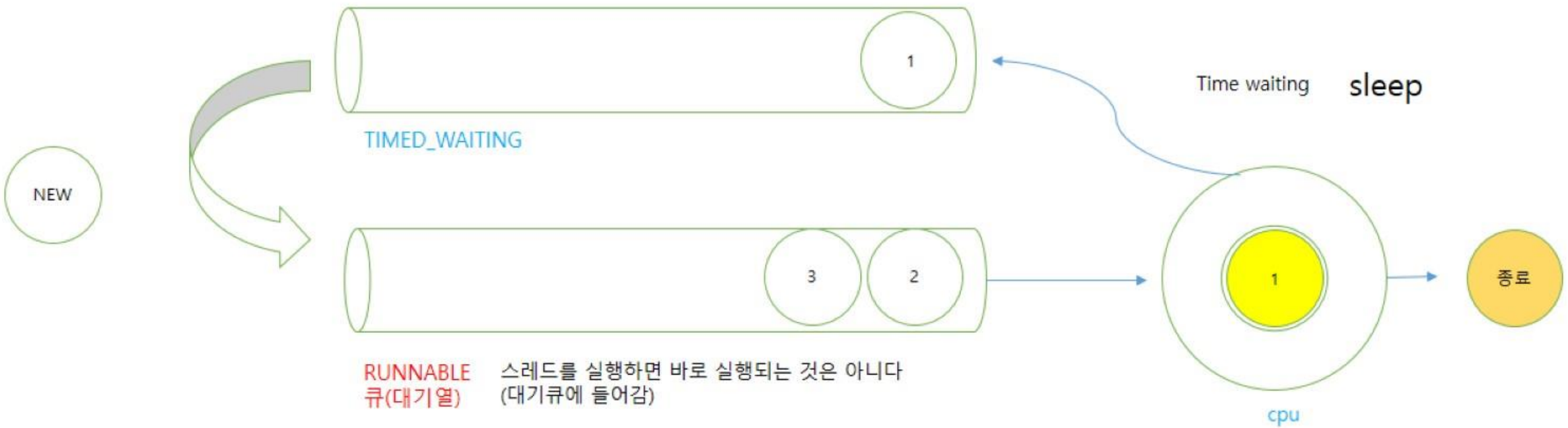








ID	스레드의 식별 값
NAME	스레드의 이름
Priority	스레드의 우선순위 (MIN (0) ~ MAX(10)) 숫자가 클 수록 우선 순위가 높음
Status	스레드의 상태 new , runnable , blocked, waiting, time waiting, terminated



sleep

```
public class Sleep예제 {
	public static void main(String[] args) {

		//스레드 생성 ,스레드 시작
		MyThread4  th1 = new MyThread4();
		th1.start();

	}
}
```

```
class MyThread4 extends Thread {
	@Override
	public void run() {
		for(int i=10; i > 0; i--) {
			System.out.println(i);
			try {
				sleep(2000);
				System.out.println( "2초간 잠자는 중 !!!!!");
			} catch (InterruptedException e) {
				e.printStackTrace();

			}
		}
	}
}
```

join

```
public class Intrrupt예제 {
	public static void main(String[] args) {


		MyThread4 th1 = new MyThread4();
		th1.start();

		try {
			th1.join();
		} catch (InterruptedException e) {
			e.printStackTrace();
		}
		System.out.println(" main  종료 !!!");
	}
}
```

```
class MyThread4 extends Thread {
	@Override
	public void run() {
		for(int i=10; i > 0; i--) {
			System.out.println(i);
			try {
				sleep(2000);
				System.out.println( "2초간 잠자는 중 !!!!!");
			} catch (InterruptedException e) {
				e.printStackTrace();

			}
		}
	}
}
```

Daemon(데몬)

```
public class Intrrupt예제 {
	public static void main(String[] args) {


		MyThread4 th1 = new MyThread4();
		th1.setDaemon(true);
		th1.start();

		//시간지연코드 추가 !!
		System.out.println(" main  종료 !!!");
	}
}
```

```
class MyThread4 extends Thread {
	@Override
	public void run() {
		for(int i=10; i > 0; i--) {
			System.out.println(i);
			try {
				sleep(2000);
				System.out.println( "2초간 잠자는 중 !!!!!");
			} catch (InterruptedException e) {
				e.printStackTrace();

			}
		}
	}
}
```

interrupt

```
public class Intrrupt예제 {
	public static void main(String[] args) {

		//스레드 생성 ,스레드 시작
		MyThread4 th1 = new MyThread4();
		th1.start();
		//시간지연을 시키기 위해 추가된 코드
		try {
			Thread.sleep(5000);
		} catch (InterruptedException e) {
			e.printStackTrace();
		}
		th1.interrupt(); // 5초 후 th1 스레드에 interrupt() 발생시킨다.
	}
}
```

```
class MyThread4 extends Thread {
	@Override
	public void run() {
		for(int i=10; i > 0; i--) {
			System.out.println(i);
			try {
				sleep(2000);
				System.out.println( "2초간 잠자는 중 !!!!!");
			} catch (InterruptedException e) {
				e.printStackTrace();
				//인터럽트 발생시  작업을 종료하고 돌아감 ..
				return;
			}
		}
	}
}
```

Daemon응용
3초마다 자동저장 스레드 만들기

```
public class 자동저장3초마다 {

	public static void main(String[] args) {
		AutoSave t = new AutoSave ();
		t.setDaemon(true); //  데몬스레드로 생성됨
		t.start();
		for(int i=1; i <= 10; i++) {
			System.out.println(i + "작업을 진행합니다.");
			try{
				Thread.sleep(1000);
			} catch(InterruptedException e) {}
		}

	System.out.println("프로그램을 종료합니다.");
	//데몬스레드도 함께 종료됨
    }
}
```

```
class AutoSave extends Thread{
	public void autoSave() {
		System.out.println("작업파일이 자동저장되었습니다.");
	}

	//3초 간격으로 autoSave() 호출함
	public void run() {
		while(true) {
		try {
			Thread.sleep(3 * 1000); // 3초 잠자기
		} catch(InterruptedException e) {}
		autoSave();
		}
	}
}
```

여러스레드에서 동일한 코드를 수행하는 경우

```
public class Test{

        public static void main(String[] args) {
                Thread th1 = new Thread ( );
                th1.start();

                Thread th2  = new Thread( );
                th2.start();
        }
}
```

```
Runnable

        void run{
            print();
        }


        void print() {

           Thread th = Thread.currentThread();

            for( int i=0; i< 100; i++)
                System.out.println("print"+ i);

            System.out.println( th.getName()  + th.getId() );

        }
```

실행주체
확인할수 있음