제네릭 타입

결정되지 않은 타입을 파라미터로 가지는 클래스와 인터페이스이다.

선언부에 '< >' 부호가 붙고 그 사이에 타입 파라미터들이 위치한다.

```
public class 클래스명<A, B, ···> { ... }
public interface 인터페이스명<A, B, ···> { ... }
```

타입 파라미터는 일반적으로 대문자 알파벳 한 글자로 표현한다.

외부에서 제네릭 타입을 사용하려면 타입 파라미터에 구체적인 타입을 지정한다.

지정하지 않으면 Object 타입이 암묵적으로 사용된다.

와일드카드 타입 파라미터

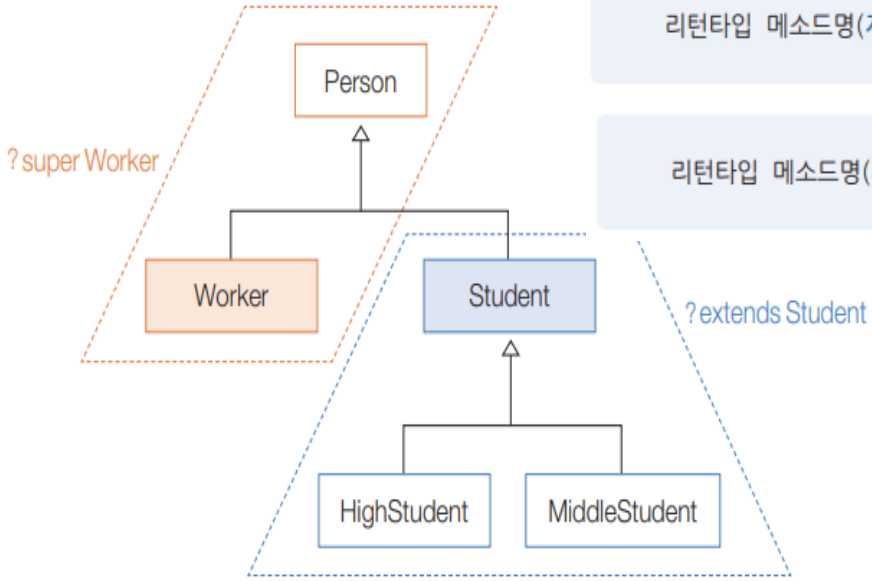
제네릭 타입을 매개값이나 리턴 타입으로 사용할 때 범위에 있는 모든 타입으로 대체할 수 있는 타입 파라미터이다.

? 기호로 표시한다.

```
리턴타입 메소드명(제네릭타입<? extends Student> 변수) { ··· }
```

```
리턴타입 메소드명(제네릭타입<? super Worker> 변수) { ··· }
```

```
리턴타입 메소드명(제네릭타입<?> 변수) { ··· }
```



제네릭 메서드

타입 피라미터를 가지고 있는 메서드다.

타입 파라미터가 메서드 선언부에 정의된다.

리턴 타입 앞에 < > 기호와 타입 파라미터를 정의한 후에 리턴 타입과 매개변수 타입에서 사용한다.

```
public  <A, B, ···>  리턴타입  메소드명(매개변수, ...) { ... }
```

타입 파라미터 정의

타입 파라미터 T는 메서드 호출시 전달되는 아규먼트의 타입에 따라 컴파일 과정에서 구체적인 타입으로 대체된다.

```
public <T> Box<T> boxing(T t) { ··· }
```

```
① Box<Integer> box1 = boxing(100);
② Box<String> box2 = boxing("안녕하세요");
```

제한된 타입 파라미터

모든 타입으로 대체할 수는 없고, 특정 타입과 자식 또는 구현 관계에 있는 타입만 대체할 수 있는 타입 파라미터로서 extends 절을 사용한다.(상위 타입은 클래스뿐만 아니라 인터페이스도 가능)

```
public <T extends 상위타입> 리턴타입 메소드(매개변수, ...) { ... }
```

```
public <T extends Number> boolean compare(T t1, T t2) {
  double v1 = t1.doubleValue();   //Number의 doubleValue() 메소드 사용
  double v2 = t2.doubleValue();   //Number의 doubleValue() 메소드 사용
  return (v1 == v2);
}
```

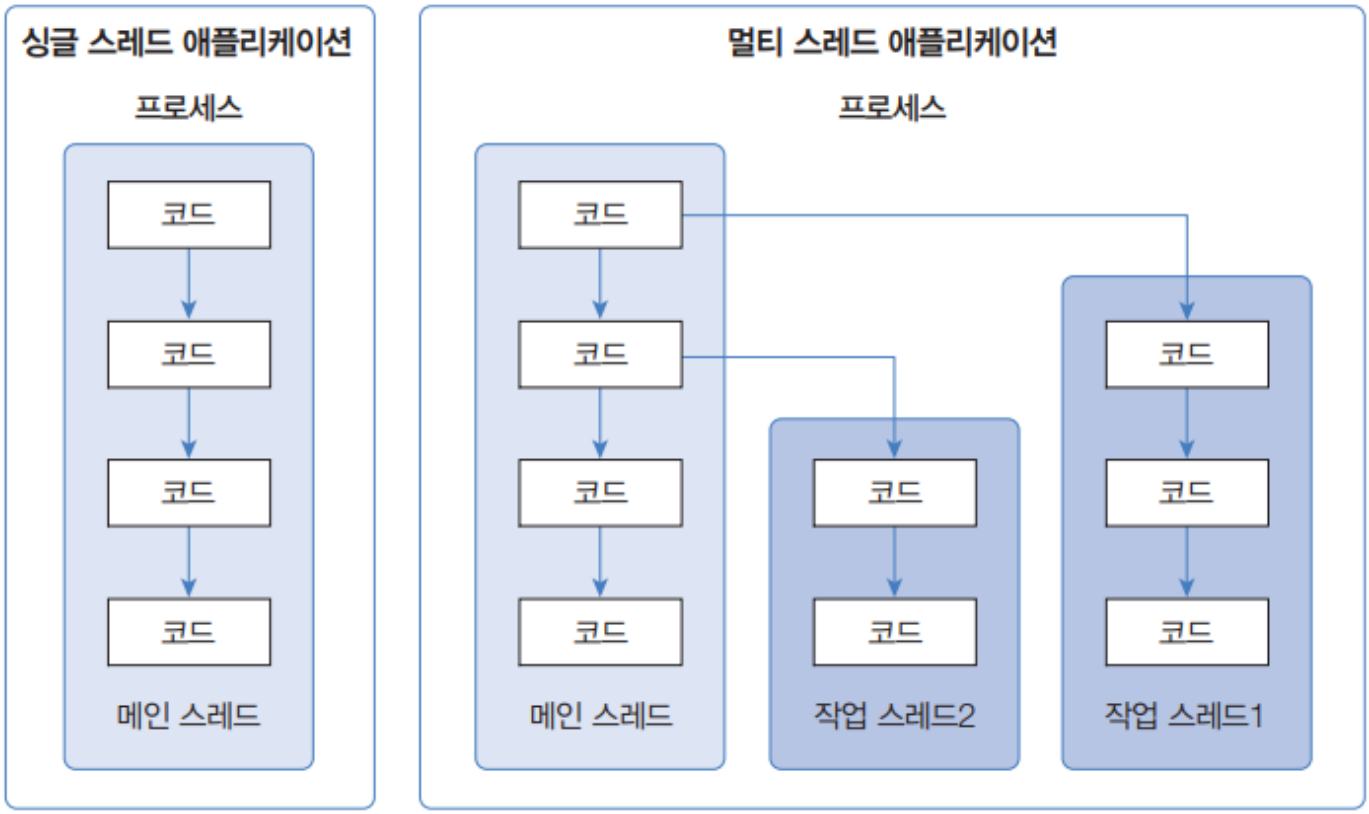
멀티스레드 프로그래밍

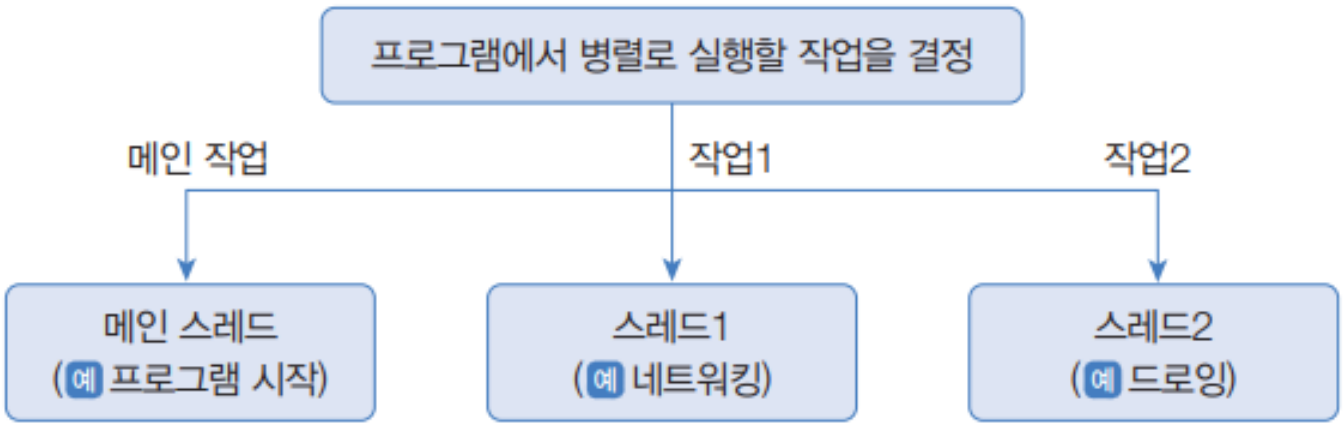
멀티 프로세스와 멀티 스레드

- 프로세스: 실행 중인 프로그램
- 멀티 태스킹: 두 가지 이상의 작업을 동시에 처리하는 것
- 스레드: 프로세스 내에서 코드의 실행 흐름
- 멀티 스레드: 두 개의 코드 실행 흐름. 두 가지 이상의 작업을 처리
- 멀티 프로세스 = 실행 중인 프로그램이 2개 이상

프로그램 단위의 멀티 태스킹 - 멀티 프로세스

프로그램 내부에서의 멀티 태스킹 - 멀티 스레드


싱글 스레드 애플리케이션
프로세스
코드
코드
코드
코드
메인 스레드
멀티 스레드 애플리케이션
프로세스
코드
코드
코드
코드
메인 스레드
코드
코드
작업 스레드2
코드
코드
코드
작업 스레드1


프로그램에서 병렬로 실행할 작업을 결정
메인 작업
작업1
작업2
메인 스레드
(예 프로그램 시작)
스레드1
(예 네트워킹)
스레드2
(예 드로잉)

프로세스 : 실행 중인 프로그램, 자원(resources)과 쓰레드로 구성

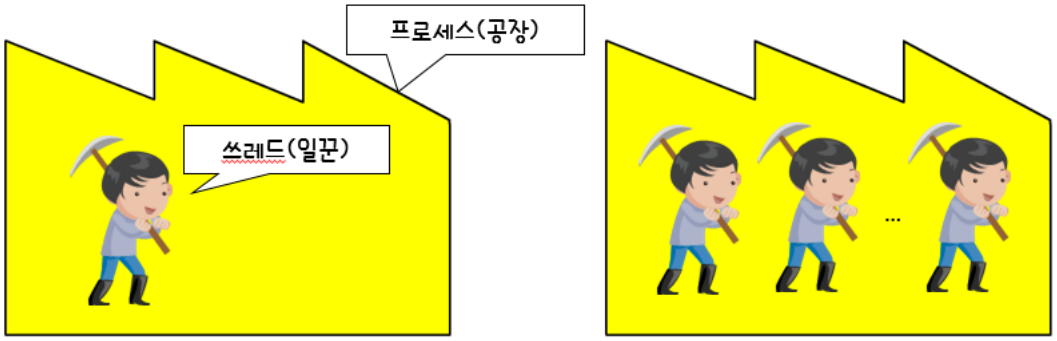
쓰레드 : 프로세스 내에서 실제 작업을 수행.

모든 프로세스는 하나 이상의 쓰레드를 가지고 있다.

프로세스 : 쓰레드 = 공장 : 일꾼

싱글 쓰레드 프로세스
= 자원+쓰레드

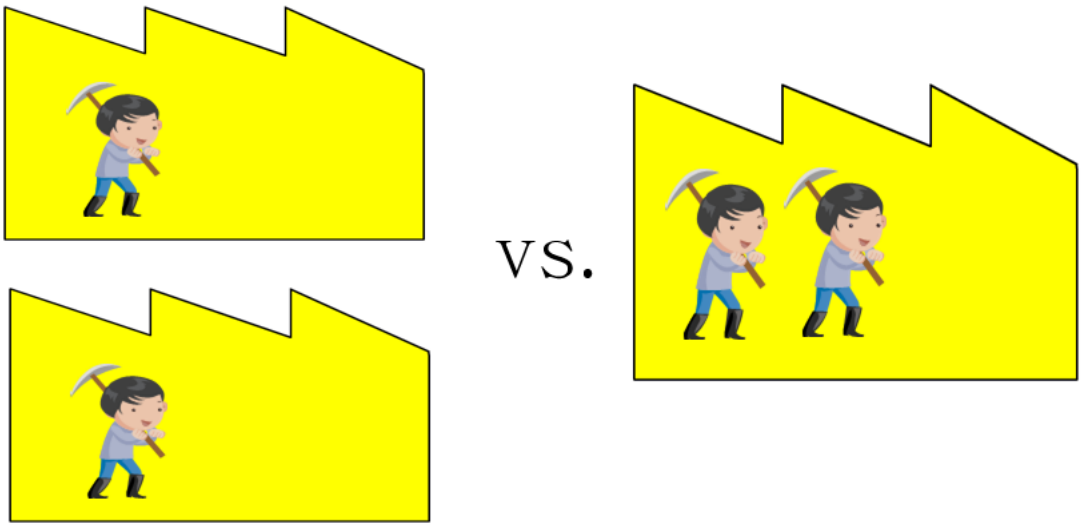
멀티 쓰레드 프로세스
= 자원+쓰레드+쓰레드+…+쓰레드



"하나의 새로운 프로세스를 생성하는 것보다

하나의 새로운 쓰레드를 생성하는 것이 더 적은 비용이 든다."

2 프로세스 1 쓰레드 vs. 1 프로세스 2 쓰레드



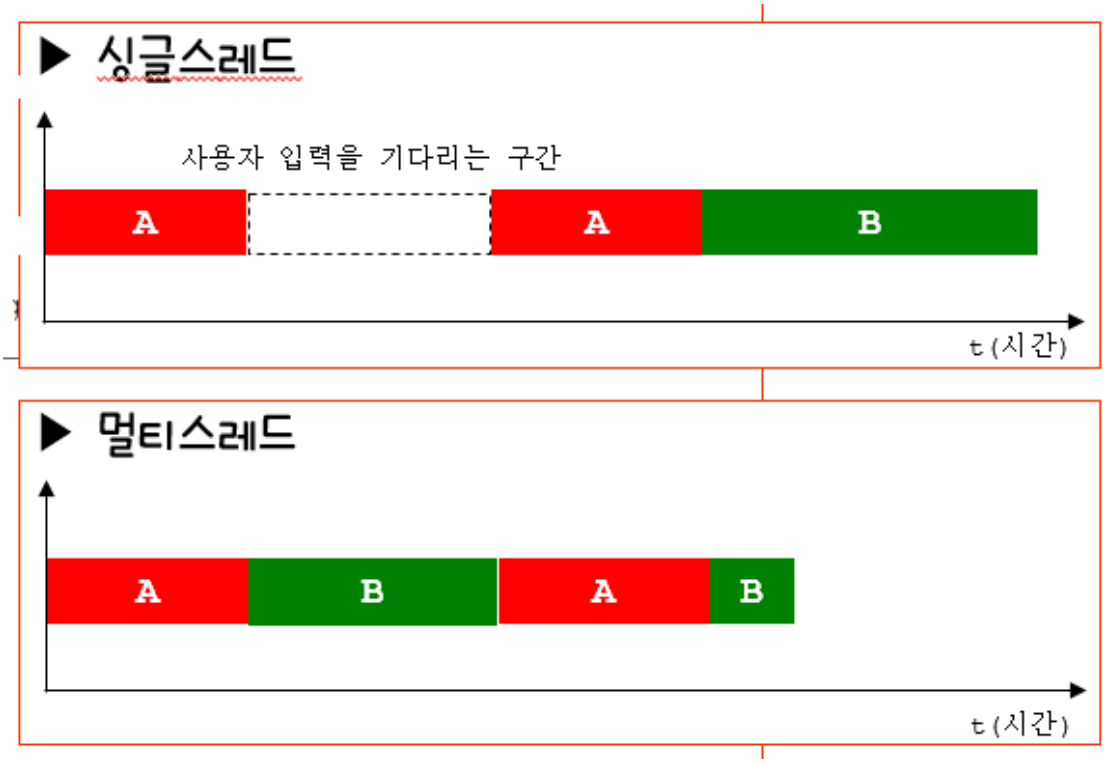
1. Thread클래스를 상속

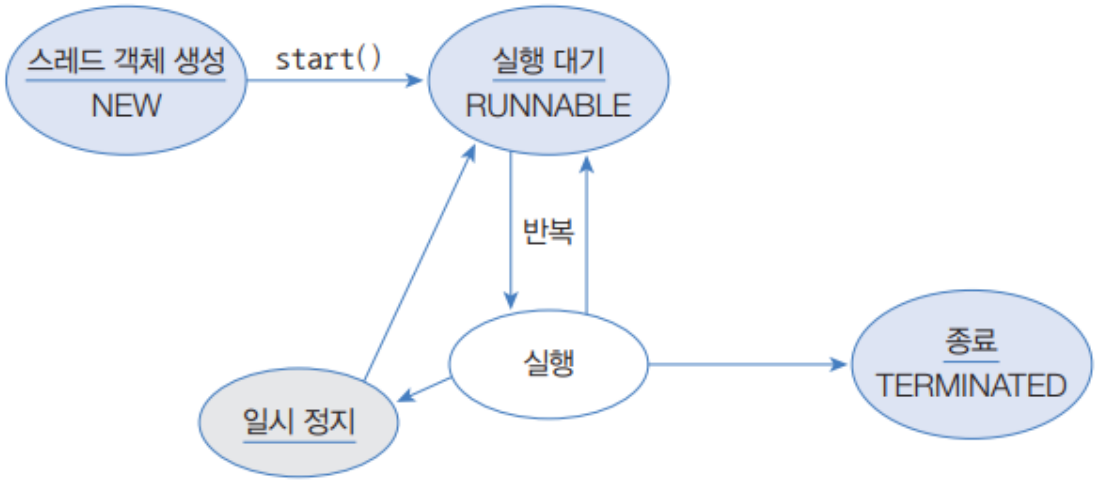
```
class MyThread extends Thread {
    public void run() { /* 작업내용 */ } // Thread클래스의 run()을 오버라이딩
}
```

2. Runnable인터페이스를 구현

```
class MyThread implements Runnable {
    public void run() { /* 작업내용 */ } // Runnable인터페이스의 추상메서드 run()을 구현
}
```

```
public interface Runnable {
    public abstract void run();
}
```




스레드 객체 생성
NEW
start()
실행 대기
RUNNABLE
반복
실행
종료
TERMINATED
일시 정지