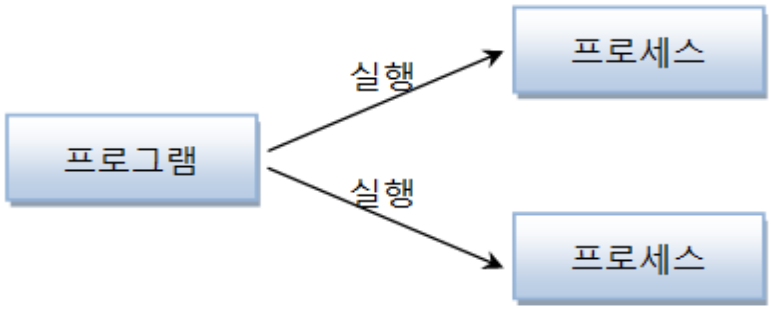
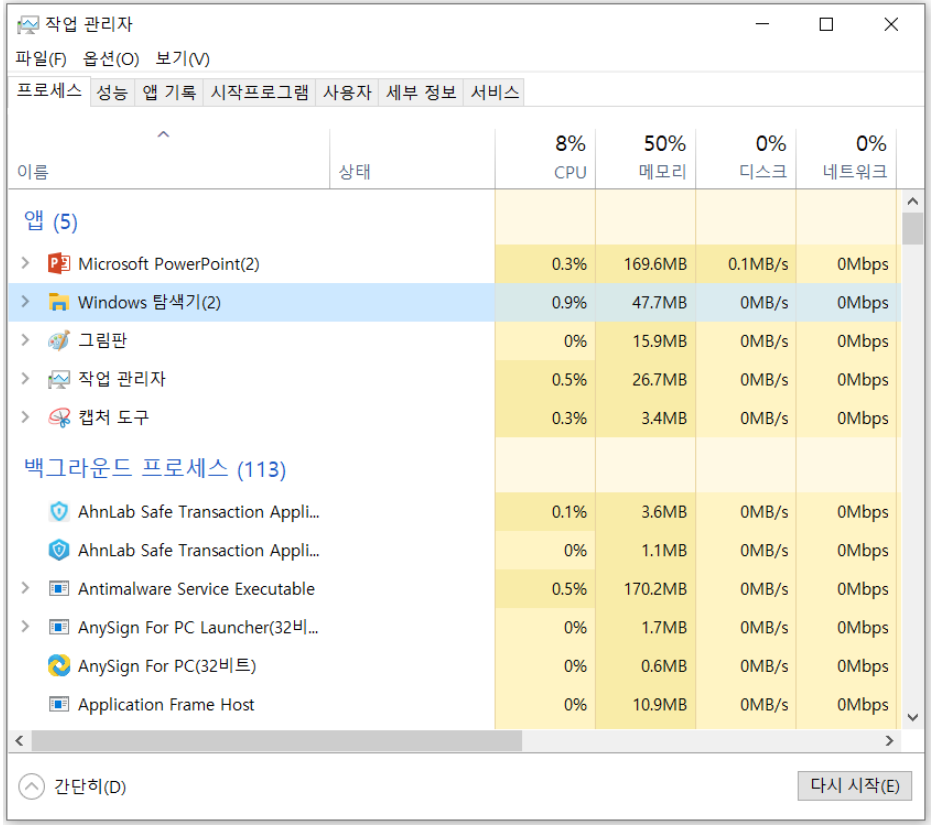
JAVA언어로 구현하는 스레드/소켓 통신 프로그래밍

프로세스와 스레드

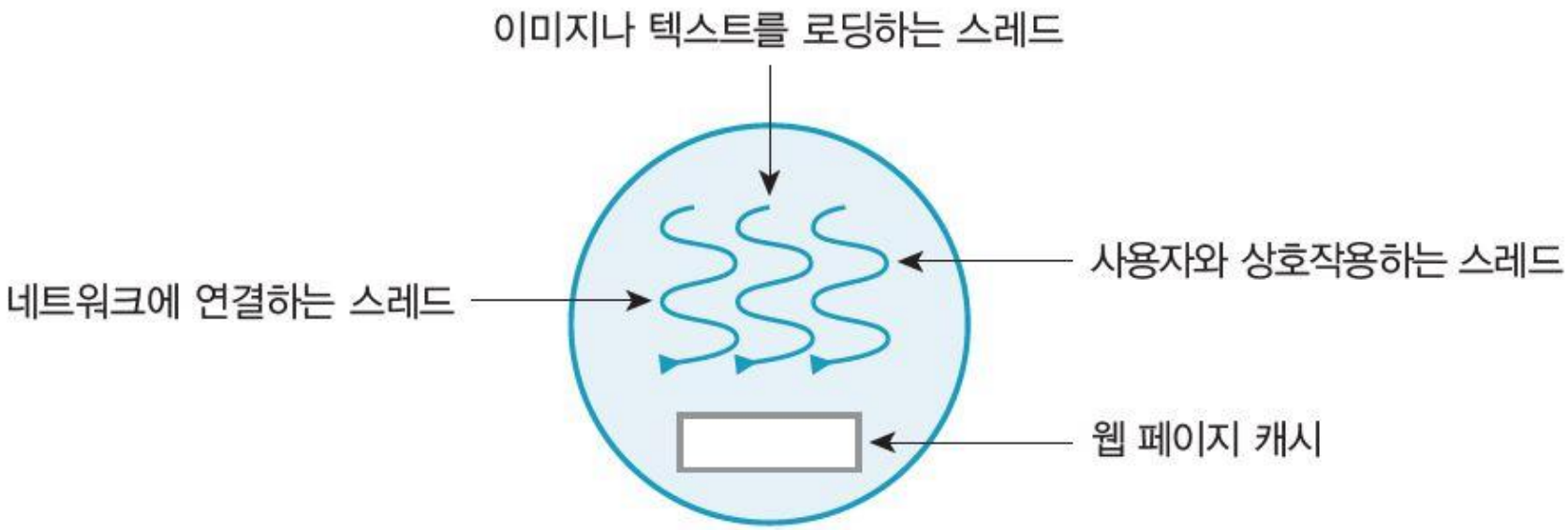
- 프로세스(process)
  - 실행 중인 하나의 프로그램
  - 하나의 프로그램이 다중 프로세스 만들기도 함





프로세스와 스레드

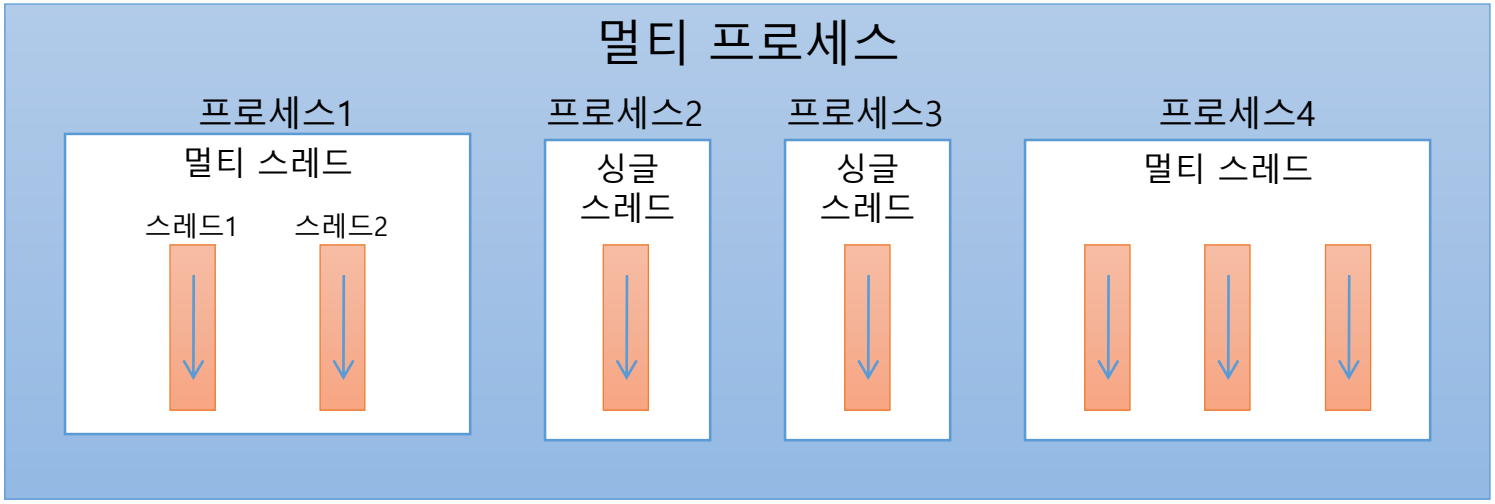
- 스레드(Thread)
  - 프로세스 내부에 있는 제어의 단일 순차 흐름
  - 하나의 스레드는 프로세스와 같이 시작, 실행, 종료의 순서를 가짐
  - 스레드 자체는 프로세스가 아니므로 홀로 실행될 수 없음
    - 프로세스 내에서만 실행 가능



프로세스와 스레드

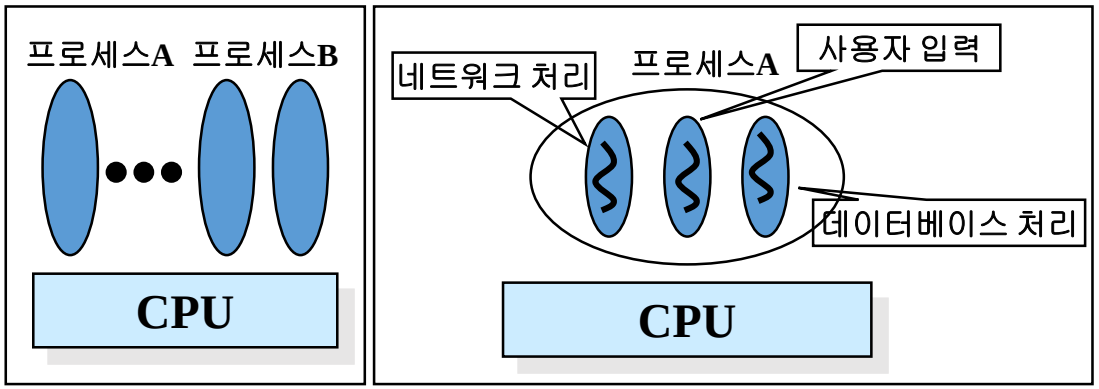
- 멀티 태스킹(multi tasking)
  - 두 가지 이상의 작업을 동시에 처리하는 것

1. 멀티 프로세스
   - 독립적으로 프로세스들을 실행하고 여러 가지 작업 처리
2. 멀티 스레드
   - 한 개의 프로세스를 실행하고 내부적으로 여러 가지 작업 처리

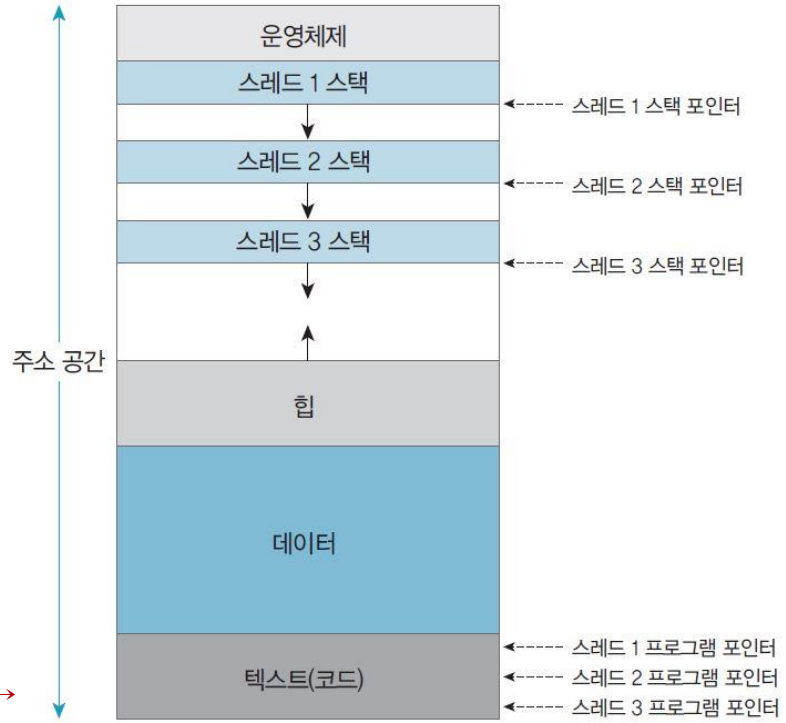


프로세스와 스레드

- 스레드의 장점
  - 완전한 프로세스의 상태를 저장하는 값비싼 부담을 덜 수 있음
    - 문맥 전환이 동일 주소 공간 내에서 행해지기 때문
    - 스레드 간의 문맥 전환에서는 단지 소수의 레지스터, 스택 포인터, 프로그램 카운터 등에 대해서만 상태 저장과 복구가 수행
  - 프로그램의 실행 시간을 단축시킴



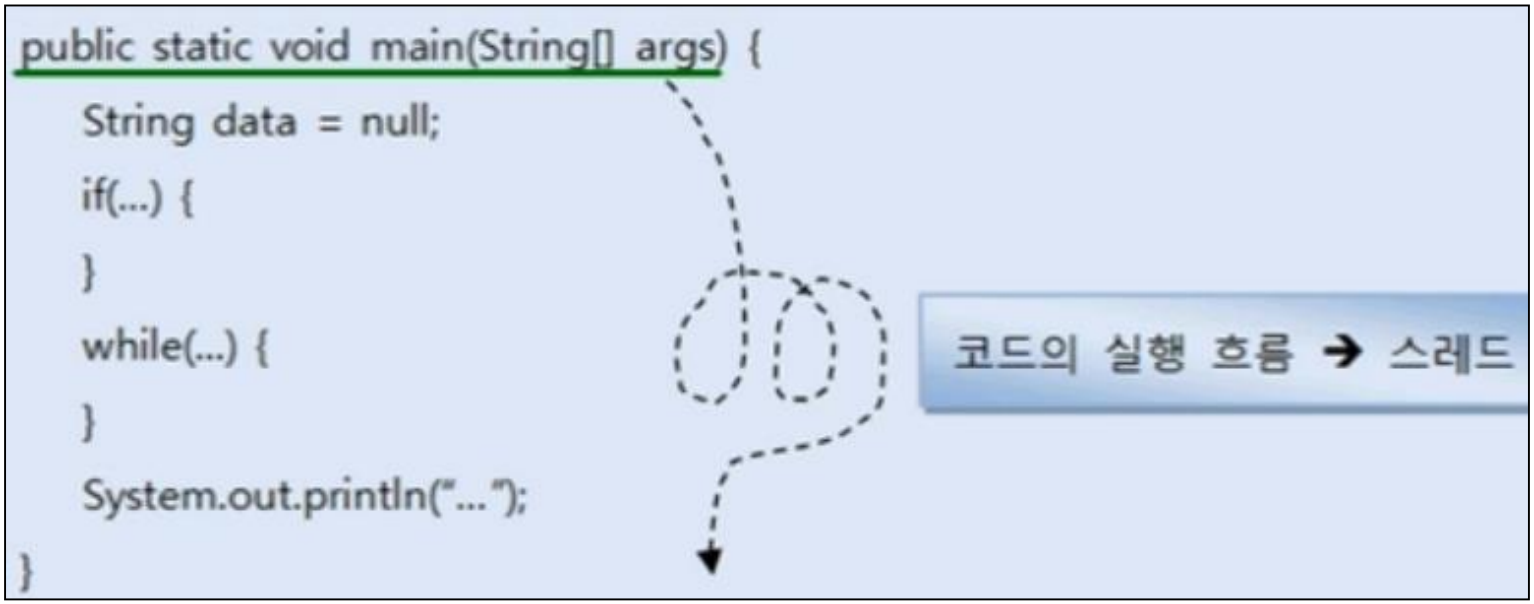
<멀티프로세스> <멀티스레딩>



같은 프로세스의 스레드들은 동일한 주소 공간 공유 →

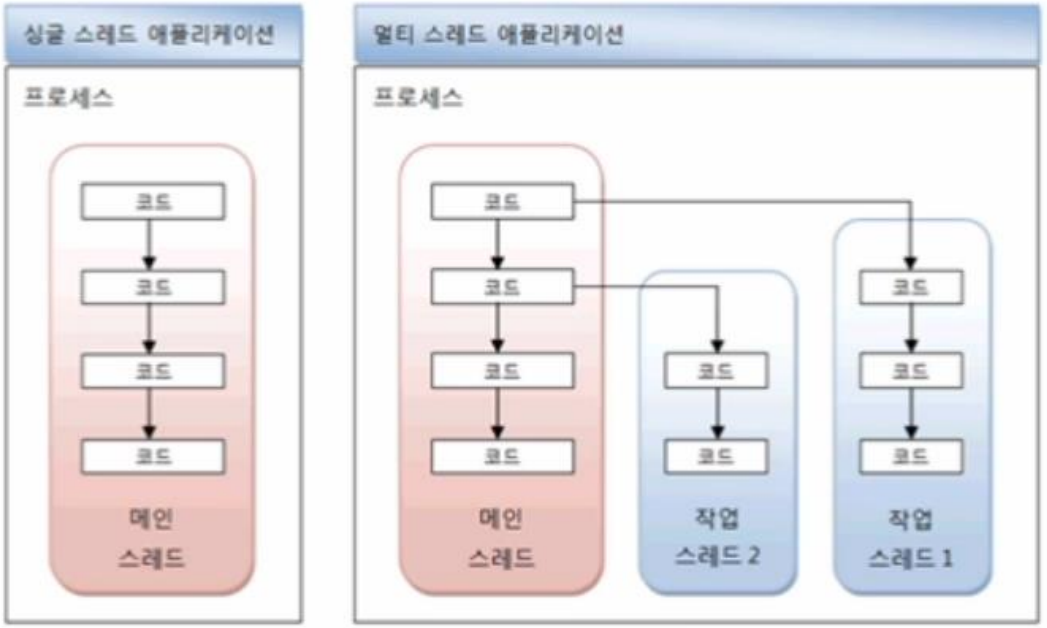
프로세스와 스레드

- 메인(main) 스레드
  - 모든 자바 프로그램은 메인 스레드가 main() 메소드 실행하며 시작
  - main() 메소드의 첫 코드부터 아래로 순차적으로 실행
  - 실행 종료 조건
    - 마지막 코드 실행
    - return문 실행



프로세스와 스레드

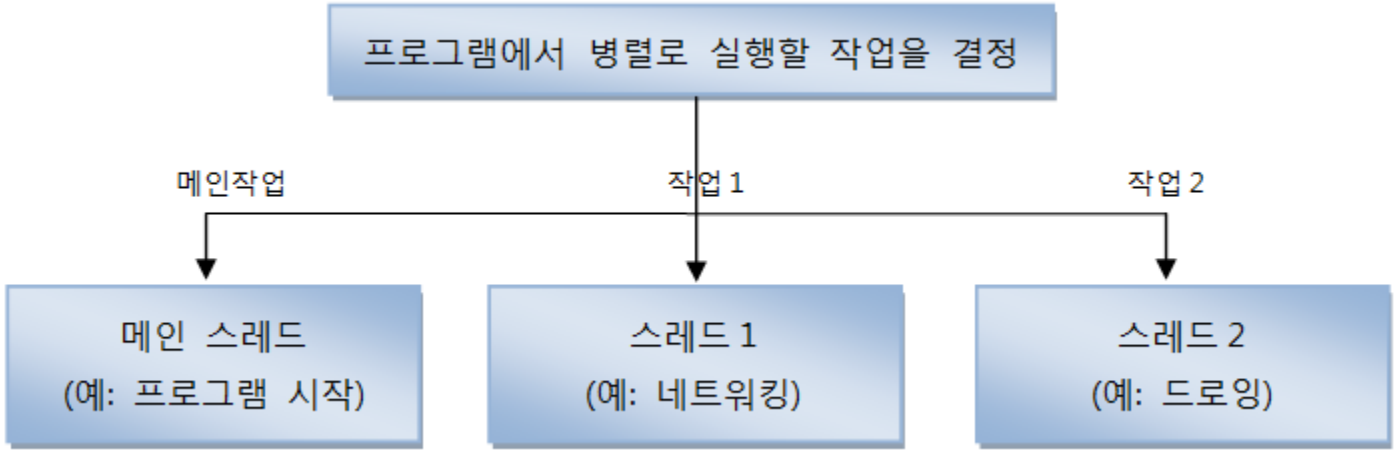
- main 스레드는 작업 스레드들을 만들어 병렬로 코드들을 실행



- 프로세스의 종료
  - 싱글 스레드: 메인 스레드가 종료하면 프로세스도 종료
  - 멀티 스레드: 실행 중인 스레드가 하나라도 있다면, 프로세스 미종료
    - 메인 스레드가 작업 스레드보다 먼저 종료되더라도 작업 스레드가 계속 실행 중이라면 프로세스는 종료되지 않음

작업 스레드 생성과 실행

- 멀티 스레드로 실행하는 어플리케이션 개발
  - 몇 개의 작업을 병렬로 실행할지 결정하는 것이 필요



- 작업 스레드 생성 방법
  1. Thread 클래스로부터 직접 생성
     - Runnable을 매개값으로 갖는 생성자 호출
  2. Thread 하위 클래스로부터 생성
     - Thread 클래스 상속 후 run 메소드 재정의 해 스레드가 실행할 코드 작성

프로그램 이해를 위해 필요한 개념들..

- 클래스
  - 정의: 객체를 정의해 놓은 것
  - 용도: 객체를 생성하는데 사용
- 객체
  - 정의: 실제로 존재하는 것. 사물 또는 개념
  - 용도: 객체가 가지고 있는 기능과 속성에 따라 다름

클래스	객체
제품 설계도	제품
TV 설계도	TV
붕어빵 틀	붕어빵

프로그램 이해를 위해 필요한 개념들..

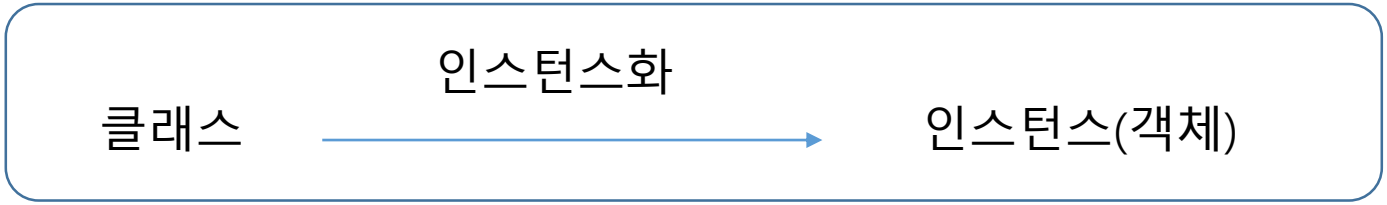
속성	크기, 색상, 볼륨, 채널 등
기능	켜기, 끄기, 볼륨 높이기, 볼륨 낮추기, 채널 바꾸기 등

```
class Tv {
        int inch;
        string color;
        int volume;
        int channel;
        boolean power;

        void power( )          { power !=power; }
        void channelUp( )      { channel++; }
        void channelDown()     { channel--; }
                :
}
```

프로그램 이해를 위해 필요한 개념들..

- 인스턴스화

  - 클래스로부터 객체를 만드는 과정



- 객체의 생성과 사용

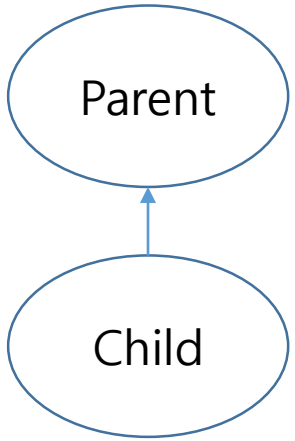
```
클래스명 변수명;
변수명 = new 클래스명();

Tv t;
t=new Tv();
```

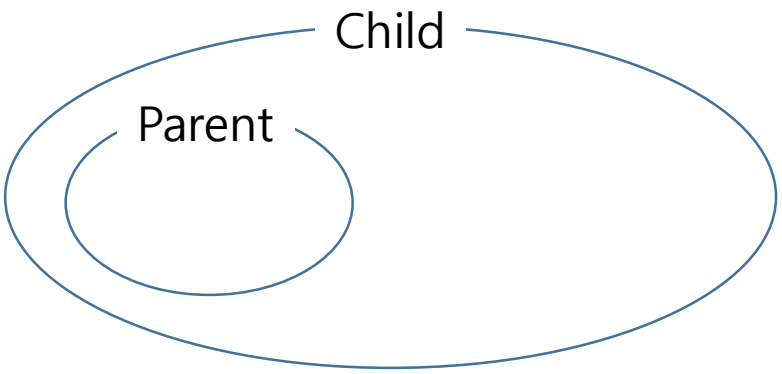
```
Tv t=new Tv();
```

프로그램 이해를 위해 필요한 개념들..

- 상속
  - 기존 클래스를 재사용하여 새로운 클래스를 작성하는 것
  - 재사용성이 높고 코드의 중복제거에 용이

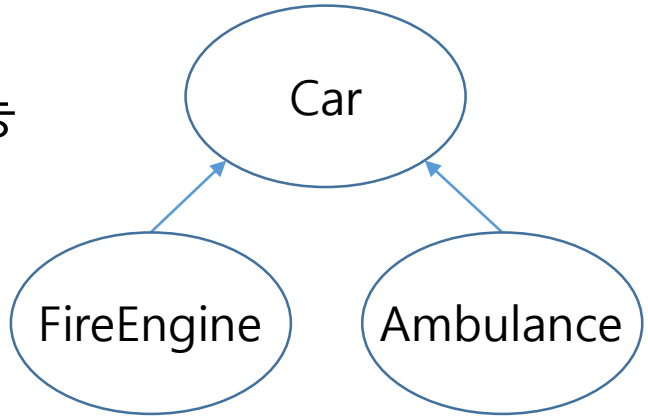
```
class Parent {    }
class Child extends Parent {
      ……
}
```





프로그램 이해를 위해 필요한 개념들..

- 참조변수의 형변환
  - 상속관계에 있는 클래스 사이에서만 가능




```
class Car{ }
class FireEngine extends Car{ }
class Ambulance extends Car{ }
```

```
FireEngine f = new FireEngine( );
Car c = (Car)f;
FireEngine f2 = (FireEngine)c;
Ambulance a = (Ambulance)f;  //에러, 형제관계이므로 형변환 불가
```

※ 참조변수 형변환을 사용하는 이유: 사용할 수 있는 멤버의 개수 조절

프로그램 이해를 위해 필요한 개념들..

- 추상 클래스
  - 추상 메서드를 포함하고 있는 클래스

```
abstract class 클래스 이름 {
        ...
        abstract 리턴타입 메서드이름( );
}
```

```
abstract class Unit {
        int x, y;
        abstract void move(int x, int y);
        void stop() { /* 현재 위치에 정지 */ }

class Marine extends Unit {   //보병
        void move(int x, int y) {  /* 지정된 위치로 이동 */ }
        void stimPack( ) { /* 보병 고유기능 */   }

Class Dropship extends Unit {   //수송선
        void move(int x, int y) {  /* 지정된 위치로 이동 */ }
        void load( ) { /* 선택된 대상을 싣는다 */   }
```

프로그램 이해를 위해 필요한 개념들..

- 인터페이스
  - 일종의 추상클래스

```
interface 인터페이스이름{
        public static final 타입 상수이름 = 값;
        public abstract 메서드이름(매개변수 목록);
}
```

```
class 클래스 이름 implements 인터페이스 이름 {

        //인터페이스에 정의된 추상메서드 구현
}
```

※ 인터페이스도 일종의 추상 클래스이므로
인터페이스와 그 인터페이스를 구현한 자식 클래스는 참조변수의 형변환 가능

※ 두가지 작업을 처리하는 예제

```
1  package threadExam.createthread;
2
3  import java.awt.Toolkit;
4
5  public class BeepPrintExample1 {
6      public static void main(String[] args) {
7          Toolkit toolkit = Toolkit.getDefaultToolkit();
8          for(int i=0; i<5; i++) {
9              toolkit.beep();
10             try { Thread.sleep(500); } catch(Exception e) {}
11         }
12
13         for(int i=0; i<5; i++) {
14             System.out.println("띵");
15             try { Thread.sleep(500); } catch(Exception e) {}
16         }
17     }
18 }
```

<termin
띵 띵 띵 띵 띵

1. Thread 클래스로부터 직접 생성

```java
package threadExam.createthread;

import java.awt.Toolkit;

public class BeepTask implements Runnable {
    public void run() {
        Toolkit toolkit = Toolkit.getDefaultToolkit();
        for(int i=0; i<5; i++) {
            toolkit.beep();
            try { Thread.sleep(500); } catch(Exception e) {}
        }
    }
}
```

```java
package threadExam.createthread;
import java.awt.Toolkit;
public class BeepPrintExample2 {
    public static void main(String[] args) {
        //how1
        Runnable beepTask = new BeepTask();
        Thread thread = new Thread(beepTask);

        //how2
        /*Thread thread = new Thread(new Runnable() {
            @Override
            public void run() {
                Toolkit toolkit = Toolkit.getDefaultToolkit();
                for(int i=0; i<5; i++) {
                    toolkit.beep();
                    try { Thread.sleep(500); } catch(Exception e) {}
                }
            }
        });*/
        thread.start();

        for(int i=0; i<5; i++) {
            System.out.println("띵");
            try { Thread.sleep(500); } catch(Exception e) {}
        }
    }
}
```

<termi
띵 띵 띵 띵 띵

참고문헌

- 이것이 자바다, 신용권 저, 한빛미디어
- Java의 정석[기초편], 남궁성 저, 도우출판
- 운영체제, 구현회 저, 한빛미디어