**구성**

**1. 자바의 개념**

**2. 자바의 클래스**

**3. 자바의 상속**

**4. 자바의 입출력**

**5. 자바 소켓 프로그래밍 (소켓이란? 소켓의 통신 방법 TCP)**

**6. 자바 소켓 프로그래밍 (thread란?)**

**+ 소켓프로그래밍 구현해놓은 블로그 주소**

**자바 Java**

* 개념 : c언어 + 객체 지향성 = c++가 달리, 처음부터 **객체 지향 언어로 개발된 프로그래밍 언어**
* 특징 : **자바가상머신(java virtual machine, JVM)**을 사용해, 운영체제와 독립적으로 동작 가능

        그래서 어느 운영체제에서나 같은 형태로 실행됨

* 역사

가전제품 내에서 동작하는 임베디드 프로그램을 위한 언어로 썬 마이크로시스템즈사의 제임스 고슬링 팀에 의해 개발

1991년 오크(oak)라는 이름으로 시작해 1996년 개발된 1.0.2 버전부터 자바라는 이름을 사용

2009년에 썬마이크로시스템즈사가 오라클과 인수 합병됨에 따라 자바 또한 오라클의 소유가 됨

* 장점

① 운영체제와 독립적으로 실행

② 불필요한 기능을 과감히 제거해 다른 언어에 비해 배우기 쉬움

③ 자동 메모리 관리 등을 지원해 다른 언어에 비해 안정성이 높음

④ 연산자 오버로딩을 금지하고, 제네릭을 도입함으로써 코드의 가독성을 높임

⑤ 자바에 관한 수많은 참고 자료 찾을 수 있음

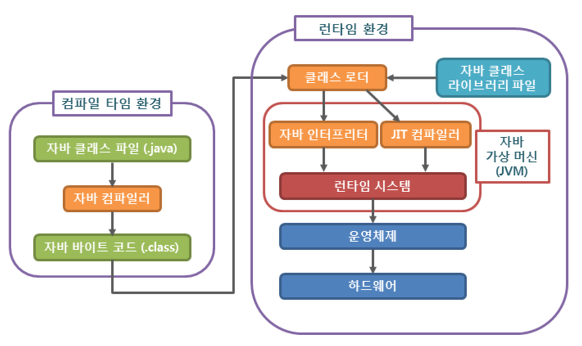
* 단점

① 실행을 위해 자바 가상 머신을 거쳐야 하므로, 다른 언어에 비해 실행 속도가 느림

② 예외 처리가 잘 되어 있지만, 개발자가 일일이 처리를 지정해 줘야 한다는 불편함

③ 다른 언어에 비해 작성해야 하는 코드의 길이가 긴 편

* 실행 과정



- 자바 컴파일러(Java compiler)

  작성한 자바 소스 코드를 자바 가상 머신이 이해할 수 있는 자바 바이트 코드로 변환

- 자바 바이트 코드(Java bytecode)

 자바 가상 머신이 이해할 수 있는 언어로 변환된 자바 소스 코드를 의미

       자바 컴파일러에 의해 변환되는 코드의 명령어 크기가 1바이트라서 자바 바이트 코드라고 부름

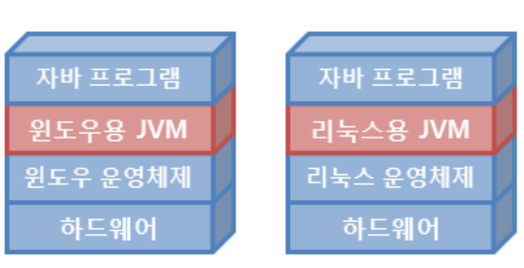
 자바 바이트 코드는 자바 가상 머신만 설치되어 있으면, 어떤 운영체제에서라도 실행 가능

- 자바 가상 머신(JVM)

  자바 바이트 코드를 실행시키기 위한 가상의 기계

        자바로 작성된 모든 프로그램은 자바 가상 머신에서만 실행될 수 있으므로,

  자바 프로그램을 실행하기 위해서는 반드시 자바 가상 머신이 설치 필수



  위의 그림처럼 서로 다른 운영체제라도 자바 가상 머신만 설치되어 있다면,

  (각 운영체제에 맞는 자바 가상 머신을 설치)

  같은 자바 프로그램이 아무런 추가 조치 없이 동작 가능

자바 프로그램은 한 개 이상의 클래스로 구성

**클래스(Class)** ? 객체 지향 프로그래밍의 추상화(abstraction)라는 개념을 직접 구현한 것

선언방법

접근제어자 **class** 클래스이름 {

    접근제어자 필드1의타입 필드1의이름;

    접근제어자 필드2의타입 필드2의이름;

    ...

    접근제어자 메소드1의 원형

    접근제어자 메소드2의 원형

    ...

}

실제 코드

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | class Car {      private int currentSpeed;      private int accelerationTime;        public void accelerate(int speed, int second) {          System.out.println(second + "초간 속도를 시속 " + speed + "(으)로 가속함!!");      }  }    public class Method01 {      public static void main(String[] args) {          Car myCar = new Car();   // 객체 생성, 클래스 이름 + 객체 참조 변수 이름 = new 클래스 이름();          myCar.accelerate(60, 3); // 메소드 호출      }  }    [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

자바프로그램이 실행되면 맨 먼저 main()메소드를 찾아 그안에 명령문을 차례대로 실행

따라서 하나의 자바 프로그램에는 main()메소드를 가지는 클래스가 반드시 하나는 존재해야 함

main()메소드는 반드시 **public static void**로 선언되어야한다.

**클래스**는 한 개 이상의 **필드(field)나 메소드로(method)**구성

**필드(field)**? 클래스에 포함된 변수(variable)

변수 : 데이터를 저장할수 있는 메모리 공간

자바에서는 다음과 같이 8가지 종류의 기본형 변수를 제공

- 정수형 : byte, short, int, long

- 실수형 : float, double

- 문자형 : char

- 논리형 : Boolean

클래스 내에서 필드는 선언된 위치에 따라 다음과 같이 구분한다.

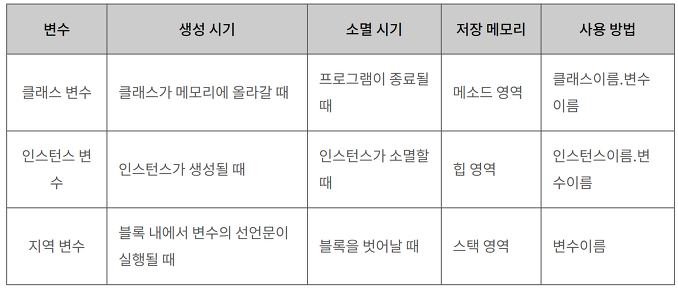
|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8 | class car {      static int modelOutput; // 클래스 변수      String modelName;       // 인스턴스 변수        void method() {          int something = 10; // 지역 변수      }  }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

**1. 클래스 변수(static variable) : 클래스 영역에 위치한 변수 중에서 static 키워드를 가지는 변수**

**2. 인스턴스 변수(instance variable) : 클래스 영역에 위치한 변수 중 static 키워드를 가지지 않는 변수**

**3. 지역 변수(local variable) : 메소드나 생성자, 초기화 블록 내에 위치한 변수**

선언된 위치에 따라 구분되는 변수는 생성 및 소멸 시기, 저장되는 메모리 공간과 사용 방법까지도 서로 다름



**메소드(method)**? 어떠한 작업을 수행하기 위한 명령문의 집합

클래스 내에서 메소드는 static 키워드의 여부에 따라 다음과 같이 구분됩니다.

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | class Car {      boolean door; // 인스턴스 변수      void openDoor() {  // 인스턴스 메소드          door = true;      }      static void toggleDoor(boolean d) { // 클래스 메소드          return !d;      }  }    [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

**1. 클래스 메소드(static method) : static 키워드를 가지는 메소드**

**2. 인스턴스 메소드(instance method) : static 키워드를 가지지 않는 메소드**

**\*\*static 키워드\*\***

프로그램 시작시 최초에 단 한번만 생성되고 초기화 됨

 static이 붙으면 클래스의 모든 객체가 공유 가능 심지어 클래스 객체 생성 없이도 호출하여 사용 가능

 다른 클래스에서 접근하려면 class명.static변수명 써주면 됨

클래스 앞에 붙은 **접근 제어자**가 무엇이냐.

객체 지향에서 정보 은닉(data hiding)이란 사용자가 굳이 알 필요가 없는 정보는 사용자로부터 숨겨야 한다는 개념이다.

그렇게 함으로써 사용자는 언제나 최소한의 정보만으로 프로그램을 손쉽게 사용할 수 있게 된다.

자바에서는 이러한 정보 은닉을 위해 접근 제어자(access modifier)라는 기능을 제공한다.

**접근 제어자의 종류**

**① private**

private 접근 제어자를 사용하여 선언된 클래스 멤버는 외부에 공개되지 않으며, 외부에서는 직접 접근할 수 없습니다.

**② public**

    public 접근 제어자를 사용하여 선언된 클래스 멤버는 외부로 공개되며, 해당 객체를 사용하는 프로그램 어디에서나 직접 접근할 수 있습니다.

**③ default**

접근 제어자가 지정되지 않으면 자동적으로 default 접근 제어를 가지게 됩니다.

     default 접근 제어를 가지는 멤버는 같은 클래스의 멤버와 같은 패키지에 속하는 멤버에서만 접근할 수 있습니다.

**④ protected**

protected 멤버는 부모 클래스에 대해서는 public 멤버처럼 취급되며, 외부에서는 private 멤버처럼 취급됩니다.

접근 제어자의 접근 범위가 보다 많은 제어자부터 적은 제어자 순으로 나열하면

public > protected > default > private

**접근 제어자말고 다른 제어자는 무엇이 있느냐.**

**제어자 종류**

**① static //위에서 다룸**

**② final**

변경할 수 없다는 의미로 사용. c++에서 const랑 똑같은 기능

필드나 지역 변수에 사용하면 값을 변경할 수 없는 상수(constant)가 되고

클래스에 사용하면 해당 클래스는 다른 클래스가 상속받을 수 없게 된다.

메소드에 사용하면 해당 메소드는 오버라이딩(overriding)을 통한 재정의가 불가능 하다.

**③ abstract // 객체지향프로그래밍 설명할때 다룸**

**abstract** **class** Car {       *// 추상 클래스*

**abstract** **void** **brake**(); *// 추상 메소드*

}

클래스나 메소드에 붙일 수 있음

반드시 사용해야하는 메소드를 추상 클래스에 추상 메소드로 선언해놓으면, 이 클래스를 상속받는 모든 클래스에서는 이 추상 메소드를 반드시 재정의 해야 한다.

**자바의 상속**

- 자바는 하나의 클래스만을 상속받는 **단일 상속만 가능**

   (다중 상속일 경우, 메소드 출처의 모호성 문제 때문에)

- 상속을 통해 기존해 작성된 클래스 재활용 가능하고,

  클래스 간 계층적 관계를 구성함으로써 다형성 문법적 토대 마련

코드 예시

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | class Parent {      private int a = 10; // private 필드      public int b = 20;  // public 필드  }    class Child extends Parent {      public int c = 30;  // public 필드      void display() {         // System.out.println(a); // 상속받은 private 필드 참조, 오류 발생         System.out.println(b);    // 상속받은 public 필드 참조         System.out.println(c);    // 자식 클래스에서 선언한 public 필드 참조      }  }    public class Inheritance01 {      public static void main(String[] args) {          Child ch = new Child();          ch.display();      }  }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

클래스를 통한 다중 상속은 지원하지 않지만,

**인터페이스** 라는 것을 이용해 다중 상속은 할 수 있다.

**인터페이스(interface)** ?

- 다른 클래스를 작성할 때 기본이 되는 틀을 제공하면서, 다른 클래스 사이의 중간 매니저 역할 까지 담당하는 일종의 **추상 클래스**

**주의점**

자바의 추상클래스는 생성자, 필드, 일반메소드, 추상메소드를 포함할수 있지만

인터페이스는 추상메소드와 상수만 포함 가능하다.

- 선언 문법

접근제어자 **interface** 인터페이스이름 {

**public** **static** **final** 타입 상수이름 **=** 값;

    ...

**public** **abstract** **메소드이름**(*매개변수목록*);

    ...

}

 인터페이스의 모든 필드는 상수만 가능하니깐 public static final이어야 하며,

 모든 메소드는 추상 메소드만 가능하니깐 public abstract이어야 한다.

 접근제어자는 생략이 가능하며, 생략 시 컴파일 할때 자바 컴파일러가 자동으로 default로 정해서 추가해줌.

- 클래스가 인터페이스 상속받는 문법

**class** 클래스이름 **implements** 인터페이스이름 { .**.**. }

**class** 클래스이름 extend 상위클래스이름 **implements** 인터페이스이름 { .**.**. } //상속과 동시도 가능

- 코드 예시

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | interface Animal { public abstract void cry(); }  interface Pet { public abstract void play(); }    class Cat implements Animal, Pet {      public void cry() {          System.out.println("냐옹냐옹!");      }      public void play() {          System.out.println("쥐 잡기 놀이하자~!");      }  }    class Dog implements Animal, Pet {      public void cry() {          System.out.println("멍멍!");      }      public void play() {          System.out.println("산책가자~!");      }  }    public class Polymorphism04 {      public static void main(String[] args) {          Cat c = new Cat();          Dog d = new Dog();            c.cry();          c.play();          d.cry();          d.play();      }  }    [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

**자바 입출력**

- 자바에서는 파일이나 콘솔의 입출력을 직접 다루지 않고, **스트림(stream)**이라는 흐름을 통해 다룸.

**스트림** ? 실제의 입력이나 출력이 표현된 데이터의 이상화된 흐름.

순서있는 데이터의 연속적인 흐름

자료의 입출력을 도와주는 중간매개체

한 방향으로만 통신할 수 있어서, 입력과 출력을 동시에 처리할 수 없음.

두 작업을 동시에 하고 싶다면, 스트림 객체 2개를 생성해야한다.

입출력 스트림은 객체 형태로 사용하며 읽기,쓰기,바이트 개수 계산 등 미리 정의된 유용한 메서드를 제공.

스트림의 종류?

문자단위로 처리하느냐, 바이트 단위로 처리하느냐에 따라 나뉨

바이트 스트림 - 입력의 경우 InputStream 이 붙으며  출력의 경우 OutputStream 이 붙는다

이미지나 동영상 등을 송수신 할때 주로 사용

문자 스트림 -입력의 경우 Reader 이 붙으며 출력의 경우 Writer 이 붙는다

HTML 문서, 텍스트 파일을 송수실할 때 주로 사용

**자바의 표준 입출력**

- 자바에서는 표준 입출력(콘솔 입출력)을 위해 System 이라는 표준 입출력 클래스를 사용.

  (c언어의 scanf(),prinf()처럼)

  (java,lang 패키지에 포함되있음)

- System 클래스에는 표준 입출력을 위해 다음과 같은 클래스 변수(static variable)가 정의되어 있다.

   ① System.in : 표준 입력

   ② System.out : 표준 출력

   ③ System.err : 표준 출력

- 사용법?

**표준 입출력 스트림은 자바가 자동으로 생성하므로, 개발자가 별도로 스트림을 생성하지 않아도 사용가능**하다.

  예를 들어, System이라는 클래스의 static 필드 in에 이미 InputStream 객체가 하나 존재

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9 | //System.in.read()는 아스키 코드값이 입력  int num = System.in.read() - 48; //or - '0';    System.out.print(7);         // print() 메소드는 줄 바꿈을 하지 않음.  System.out.println(3);       // 정수 출력  System.out.println(3.14);    // 실수 출력  System.out.println("자바!"); // 문자열 출력  System.out.println("문자열끼리의 " + "연결도 가능합니다.");  System.out.println("숫자" + 3 + "과 문자열의 연결도 가능합니다."); | [cs](http://colorscripter.com/info#e) |

 \*\*System.in을 직접 사용하면 바이트를 문자나 숫자로 변환해야 하는 어려움 존재.\*\*

콘솔 입력 예시

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | InputStream in = System.in;  int a;  byte[] b = new byte[3];  a = in.read() - '0'; //System.in에 들어있던 그 객체의 read()메소드를 사용    /\*      a = System.in.read()랑 같은 말  \*/    in.read(b);    InputStream in = System.in;    InputStreamReader reader = new InputStreamReader(in);  char[] a = new char[3];  reader.read(a);    BufferedReader br = new BufferedReader(reader);  // new BufferedReader(new InputSteamReader(System.in));  String a = br.readLine();    //InputStream - byte  //InputStreamReader - character  //BufferedReader - String  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

입력 받기 데 편한 클래스로는 **Scanner 클래스**가 있다.

단어 뿐만 아니라 숫자, 문자열등 다양하게 읽어 들일 수 있는 여러 메소드들이 준비되어있음.

Scanner 클래스를 이용해 콘솔 입력 받는 예시

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9 | import java.util.Scanner; //기본적으로 import 문 필요    Scanner sc = new Scanner(System.in);    String name = scan.next(); // "HAN"  String city = scan.next(); // "INCHEON"  int age = scan.nextInt(); // 25  double weight = scan.nextDouble(); // 65.1  boolean single = scan.nextBoolean(); // true | [cs](http://colorscripter.com/info#e) |

next - 단어, nextLine - 라인, nextInt - 정수 등 다양

자바수업에서 마지막으로 다루는 것이 채팅프로그램인데

채팅프로그램은 **소켓 프로그래밍(=네트워크 프로그래밍)과 thread**를 이용해 구현한다

먼저 소켓프로그래밍에 관해 설명

**socket 소켓**

- 개념: 두 프로그램이 네트워크를 통해 서로 통신을 수행할 수 있도록 양쪽에 생성되는 단자

    즉 프로세스 통신 간에 사용되는 양쪽 끝단을 의미한다.

    소프트웨어 차원에서 원격에 존재하는 두 호스트를 연결해주는 매개체

- 자바 안에서 소켓의 종류에는 **서버 소켓과 클라이언트 소켓**이 있다.

- 소켓은 통신하기 위해 대표적으로 **TCP와 UDP**를 이용한다.

소켓이 통신하는 데 필요한 지식인 TCP와 UDP에 대해 알아야하는데

그 중 **주로** 사용하는 TCP에 대해서 자세히 설명해보겠다.

TCP는 보통 하위 계층에서 사용하는 IP와 엮어서 TCP/IP로 표현하는 경우가 많다.

(TCP와 IP는 별개지만, TCP를 기반으로 통신하는 것들이 IP위에서 동작하기 때문에)

따라서 TCP/IP 통째로 설명하겠다. **(사실 네트워크 요약할 때 해도 되는데 걍,, 미리 살짝 요약해서 해봤어)**

**Protocl 프로토콜? 컴퓨터와 네트워크 기기가 상호 간에 통신하기 위한 규칙**

**TCP/IP(Transmission control protocol, Internet Protocol)**

- TCP : 서버와 클라이언트간에 데이타를 신뢰성있게 전달하기 위해 만들어진 프로토콜

    패킷을 조립하고, 손실된 패킷을 확인하고, 재전송 하도록 요청하는 기능을 함

packet 패킷?

데이터를 작게 나눈 단위

대략 1kbyte

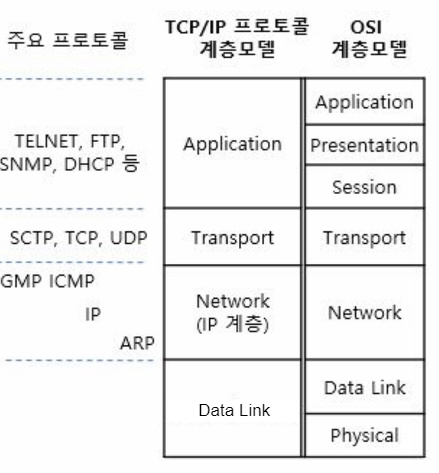
- IP : 인터넷상의 주소 규칙, 각 장치를 나타내는 주소를 가리킴

 이전에는 32bit의 값을 가지는 IPv4를 사용했지만, 주소 고갈로 인해 이제는 128bit의 값을 가지는 IPv6도 사용

->아래에서 더 자세히 설명하겠다.

- **TCP/IP는 4계층으로 나뉘어져 있다.**

  OSI 7계층(Open Systems Interconnections, 시스템들의 연결을 위한 모델) 을 웹서비스에 맞게 4계층으로 단순화한 모델



**4계층**

**Data link – Network – Transport – Application**

**링크 – 네트워크 – 전송 - 응용**

1. 링크계층 : Ethernet, ATM 등 네트워크에 접속하는 하드웨어적인 면을 다룸

                 물리적 계층으로 IP주소를 MAC 주소로 변환

2. 네트워크계층 : 데이터를 정의하고 데이터의 경로를 라우팅

routing 라우팅

어떤 네트워크 안에서 통신 데이터를 보낼 최적의 경로를 선택하는 과정

ICMP, IGMP, **IP** 등

3. 전송계층 : 네트워크에 접속되어 있는 컴퓨터 사이의 데이터 흐름 제공

     시스템을 연결하고 데이터를 전송하는 역할

**TCP**, UDP 등

4. 응용계층 : 유저에게 제공되는 애플리케이션에서 사용하는 통신의 움직임을 결정

HTTP(웹 접속), FTP(파일전송), Telnet, SMTP(메일보내기) 등

이렇게 계층화를 한 이유?

인터넷이 하나의 프로토콜로 되어있다면 사양 전체를 바꿔야 함.

하지만 계층화 되어있다면 해당 계층만 바꾸면 됨

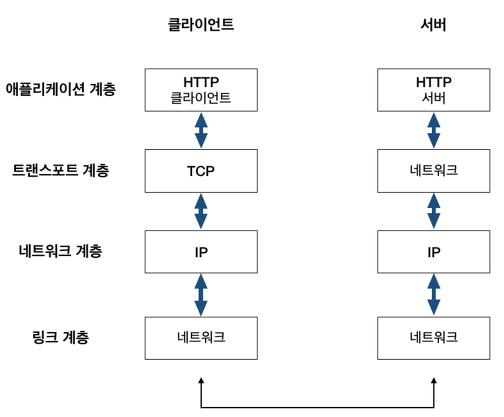
또한, 계층화로 인해 자신의 계층만 신경쓰면 되므로 설계를 편하게 할 수 있음

- **TCP/IP가 계층대로 통신하는 흐름**

  TCP/IP로 통신할 때 계층을 순서대로 거쳐 상대와 통신

  송신하는 측은 애플리케이션 계층에서부터 내려가고, 수신하는 측은 네트워크 계층부터 올라감

  HTTP를 예를 들어 설명하면 다음과 같은 순서로 통신한다.



  1. 송신측 클라이언트의 애플리케이션 계층(HTTP)에서 어느 웹 페이지를 보고 싶다라는 HTTP 리퀘스트를 지시

  2. 그 다음에 있는 트랜스포트 계층(TCP)에서는 애플리케이션 계층에서 받은 데이터(HTTP 메시지)를 통신하기 쉽게 조각 내어

(패킷으로 조각)

     안내 번호와 포트 번호를 붙여 네트워크 계층에 전달

  3. 네트워크 계층(IP)에서는 수신지 MAC 주소를 추가해서 링크 계층에 전달

     이로써 네트워크를 통해 송신할 준비가 되었음

  4. 수신측 서버는 링크 계층에서 데이터를 받아들여 순서대로 위의 계층에 전달하여 애플리케이션 계층까지 도달

  5. 수신측 애플리케이션 계층에 도달하게 되면 클라이언트가 발신했던 HTTP 리퀘스트를 수신 가능

대충 어떻게 통신하는 지 봤다면 이제 IP 에 대해서 자세히 설명해보자.

**IP(Internet Protocol)**

- 개념 : 위에서 설명했듯이 인터넷 상의 주소 규칙

- 네트워크 층에 해당 되며, 이름 그대로 인터넷을 활용하는 거의 대부분의 시스템이 IP를 이용

- 역할 : 패킷을 목적지까지 전달하는 역할 및 그에 수반되는 기타 역할

네트워크 층에서 패킷 전달할 때 IP주소도 사용하지만 MAC주소도 필요

IP주소는 각 노드에 부여된 주소를 가리키고 MAC 주소는 각 네트워크 카드에 할당된 고유의 주소

IP주소는 변경 가능하지만 기본적으로 MAC 주소는 변경할 수 없다

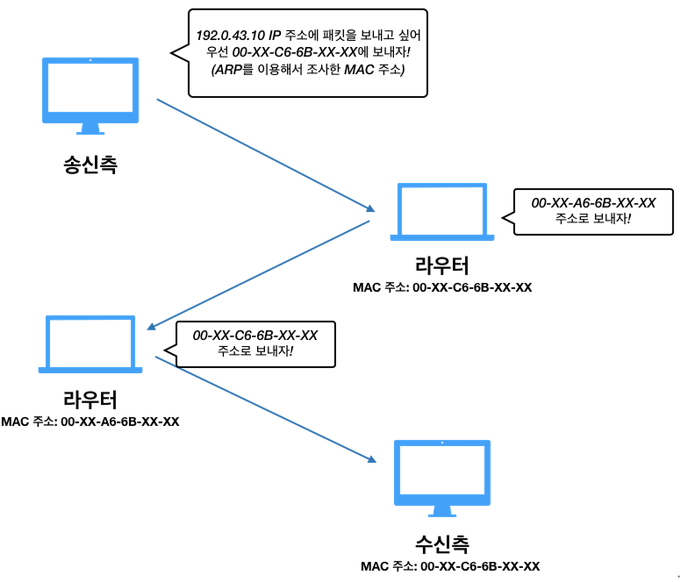
**IP 주소/ MAC 주소  
IP주소**: 컴퓨터 네트워크에서 장치들이 서로를 인식하고 통신을 하기 위해서 사용하는 특수한 번호

네트워크에 연결된 장치가 라우터이든 일반 서버이든, 모든 기계는 이 특수한 번호를 가지고 있어야 한다.

이 번호를 이용하여 발신자를 대신하여 메시지가 전송되고 수신자를 향하여 예정된 목적지로 전달된다.

**MAC 주소**: 네트워크 세그먼트의 데이터 링크 계층에서 통신을 위한 네트워크 인터페이스에 할당된 **고유 식별자**

그림을 통해 패킷 전달 과정을 보자면



다음으로 중계할 곳의 MAC 주소를 사용하여 목적지를 찾아가는데

이때, **ARP(Address Resolution Protocol: 주소 결정 프로토콜)**이라는 프로토콜이 사용됨.

ARP는 주소를 해결하기 위한 프로토콜 중 하나인데, 수신지의 IP 주소를 바탕으로 MAC 주소를 조사할 수 있다.

목적지까지 중계를 하는 도중에 컴퓨터와 라우터 등의 네트워크 기기는 목적지에 도착하기 전까지 대략적인 목적지들로 이동하는데 이 과정을 **라우팅**이라고 한다.

IP와 네트워크에서 IP의 역할에 대해 대충 알았다면

이젠 TCP와 TCP의 역할에 대해 보자

IP가 데이터의 배달을 처리한다면 TCP는 패킷을 추적 및 관리

**TCP(Transmission Control Protocol)**

- 개념 : 위에서 설명했듯이, 서버와 클라이언트 간의 데이터를 신뢰성 있게 전달하기 위한 프로토콜

     연결지향적인 프로토콜

연결지향적? 클라이언트와 서버의 소켓이 미리 **연결**되있어야 데이터 송수신이 가능

     데이터가 유실되면 데이터 재전송을 요청함으로써 신뢰성 보장

오류제어, 흐름제어, 혼잡제어, 타미어 재전송 등의 기능을 함(이는 나중에 네트워크 수업을 정리할때 다루도록 하겠다.)

- 전송계층에 해당

- 역할 : 송신 측에서는 용량이 큰 데이터를 패킷으로 나누는 역할

     수신 측에서는 전달받은 패킷을 조립하고, 패킷이 빠져서 왔을 경우 혹은 손실되서 왔을 경우 재전송 요청하는 역할

- 연결지향적인 프로토콜이기 때문에, 클라이언트와 통신을 하기 위해 **미리 연결을 맺어야함**

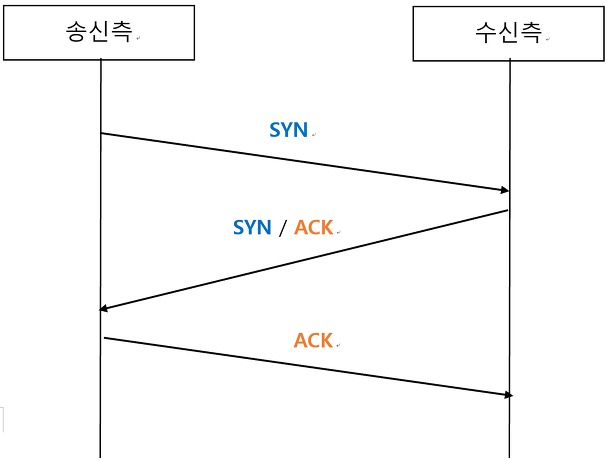
  → **three way handshake** 를 통해 연결을 맺는다.

**three way handshake** : TCP/IP 프로토콜을 이용해서 통신을 하는 응용프로그램이 데이터를 전송하기 전에

  정확한 전송을 보장하기 위해 상대방 컴퓨터와 세션을 수립하는 과정

  (양쪽 모두 데이터를 전송할 준비가 되있음을 보장)

그림으로 보자면



STEP 1 : 송신측에 접속을 요청하는 SYN 패킷을 보낸다.

STEP 2 : 수신측은 SYN 요청을 받고 송신측에 요청을 수락한다는 ACK와 SYN 패킷을 발송해

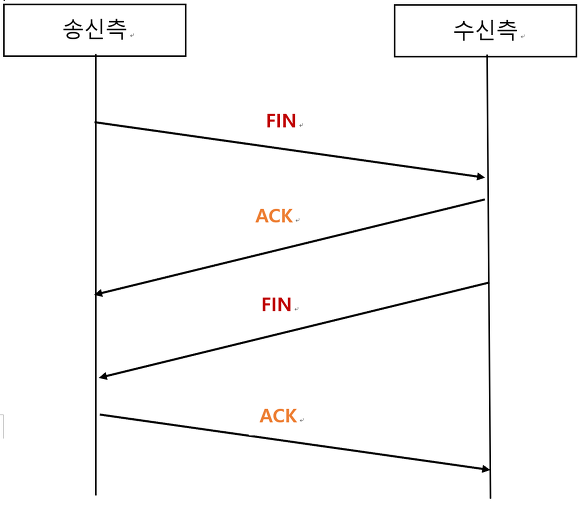
송신측이 다시 ACK으로 응답하기를 기다린다.

STEP 3 : 송신측이 수신측에 ACK을 보낸 이후로부터는 연결이 이루어지고 데이터가 오간다.

위와 같은 방식으로 통신하는 것이 신뢰성 있는 연결을 맺어 준다는 TCP의 3 way handshake 방식이다.

연결을 종료할 때는 **four way handshake**사용

그림으로 보자면



STEP 1 : 연결을 종료하고자 하는 송신측은 FIN 패킷을 전송한다.

STEP 2 : 수신측은 알겠다는 ACK 패킷을 전송하고, 자신의 통신이 끝날 때까지 기다림

(아직 전송할 데이터가 남아있다면 이어서 전송)

STEP 3 : 수신측의 통신이 완전히 끝났다면 연결 종료 요청에 합의한다는 의미로 송신측에

FIN 패킷 전송

STEP 4 : 송신측이 확인했다는 ACK 패킷 보냄

그 다음으로 thread

1:1 채팅프로그램이 아닌 다인원 채팅프로그램을 구현하려면

서버가 여러 클라이언트와 통신하면서 요청을 처리하기 위해 필요한 쓰레드에 대해 알아야한다.

(

 서버&클라이언트 1:1이면 thread사용 불필요

 but 서버 & 클라이언트1 & 클라이언트2 & 클라이언트3... 이면 thead사용해야함

)

그래서 therad란 무엇인가?

**thread 쓰레드**

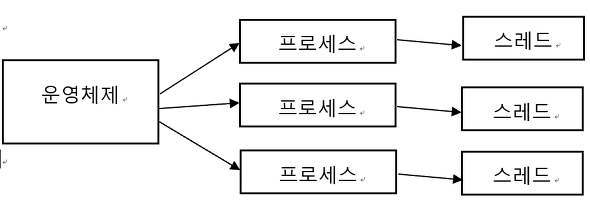
- 개념 : 하나의 프로세스 내부에서 독립적으로 실행되는 하나의 작업 단위

     세부적으로는 운영체제에 의해 관리되는 하나의 작업 혹은 태스크를 의미

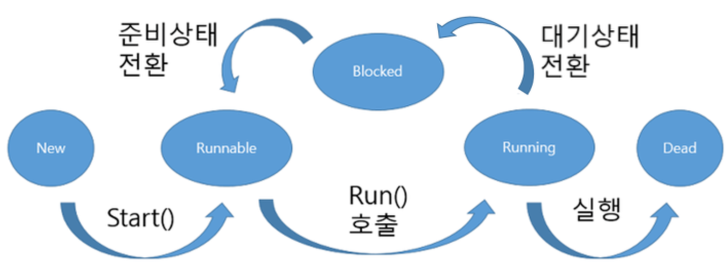
**프로세스(Process) :**실행중인 프로그램,

    프로그램을 실행하면 OS로부터 실행에 필요한 자원(메모리)을 할당받아 프로세스가 된다.

그림으로 자세히 보자면



- 쓰레드의 생명 주기



**1. Runnable (준비상태)**

스레드가 실행되기 위한 준비단계

CPU를 점유하고 있지않으며 실행(Running 상태)을 하기 위해 대기하고 있는 상태

코딩 상에서 start( ) 메소드를 호출하면 run( ) 메소드에 설정된 스레드가 Runnable 상태로 진입

“Ready“ 상태라고도 한다

**2. Running (실행상태)**

스케줄러에 의해 선택된 스레드가 CPU를 점유하여 실행하고 있는 상태이며 run() 메서드는 JVM만이 호출 가능

Runnable(준비상태)에 있는 여러 스레드 중 우선 순위를 가진 스레드가 결정되면 JVM이 자동으로 run( ) 메소드를 호출하여 스레드가 Running 상태로 진입

**3. Dead (종료상태)**

Running 상태에서 스레드가 모두 실행되고 난 후 완료 상태

“Done” 상태라고도 한다

**4. Blocked (지연 상태)**

CPU를 점유권을 상실한 상태 -> 스레드가 작업을 완수하지 못하고 잠시 작업을 멈추는 단계

후에 특정 메서드를 실행시켜 Runnable(준비상태)로 전환한다.

wait( ) 메소드에 의해 Blocked 상태가 된 스레드는 notify( ) 메소드가 호출되면 Runnable 상태로 간다.

sleep(시간) 메소드에 의해 Blocked 상태가 된 스레드는 지정된 시간이 지나면 Runnable 상태로 간다.

- 자바에서 쓰레드 구현 방법

1. Thread 클래스를 상속

2. Runnable 인터페이스를 구현

→ thread 클래스를 상속받으면 다른 클래스를 상속 받을 수 없기 때문에, Runnable 인터페이스를 구현하는 방법이 일반적.

1. Thread 클래스 상속받는 예시 코드

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | public class ThreadExam01 {     public static void main(String[] args){         MyThread01 m1 = new MyThread01("\*");         MyThread01 m2 = new MyThread01("+");         MyThread01 m3 = new MyThread01("#");         m1.start();         m2.start();         m3.start();         System.out.println("main메소드 종료.");    }  }    class MyThread01 extends Thread{     private String str;     public MyThread01(String str){         this.str = str;    }       @Override     public void run() {         for(int i = 0; i < 5; i++)             System.out.print(str);    }  }    [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

2. Runnable 인터페이스를 이용해서 구현하는 코드 예시

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | public class ThreadExam02 {     public static void main(String[] args){         Runnable r1 = new MyThread02("\*");         Runnable r2 = new MyThread02("%");         Runnable r3 = new MyThread02("#");           //Thread t1 = new Thread(new MyThread02("\*"));         Thread t1 = new Thread(r1);         Thread t2 = new Thread(r2);         Thread t3 = new Thread(r3);         t1.start();         t2.start();         t3.start();         System.out.println("main메소드 종료.");    }  }    class MyThread02 implements Runnable{     private String str;     public MyThread02(String str){         this.str = str;    }       @Override     public void run() {         for(int i = 0; i < 5; i++)             System.out.print(str);    }  }  /\*  [실행결과]  \*\*\*\*\*#####+++++main메소드 종료.  \*/    [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

두 방법 모두 공통적으로는 run()메소드를 상속받아서 오버라이딩 해줘야하며,

 쓰레드 작업을 해줄 곳에 start()호출해 줘야 한다.

start()호출하면 thread 실행 준비 완료 -> run() 실행시 thread 가 실행

이 thread 개념을 이용해서

**서버에서 하나의 클라이언트가 아닌 여러 명의 클라이언트가 접속 요청을 할때,**

**각 클라이언트와 통신하기 위해 클라이언트마다 새로운 스레드를 생성해서 멀티스레드로 운영**한다.

**멀티태스킹과 멀티쓰레딩**  
멀티태스킹(Multi-tasking, 다중작업) : 여러 개의 프로세스가 동시에 실행  
멀티스레딩(Multi-threading) : 하나의 프로세스 내에서 여러 쓰레드가 동시에 작업을 수행

- 멀티스레딩의 **장점**

① CPU 사용률을 향상

② 자원을 보다 효율적으로 사용 가능

③ 사용자에 대한 응답성이 향상

④ 작업이 분리되어 코드가 간결

- 멀티스레딩의 **단점**

: 여러 쓰레드가 같은 프로세스 내에서 자원을 공유하면서 작업을 하기 때문에 발생할 수 있는

  동기화(synchronization ) = 트랜잭션, 교착상태(deadlock)와 같은 문제들을 고려하며 프로그래밍 해야 한다.

//**소켓프로그래밍에 관한 코드를 잘 작성해놓은 블로그 주소**

<https://victorydntmd.tistory.com/135?category=704013>

-코드를 다 가져올 수 없어서 주소를 남겨놓으니 가서 보세유