자바 프로그래밍 입문

1. Java 프로그래밍이란?

1-1. 프로그래밍이란?

- 개발자 업무 영역(소스) -> 컴파일러 -> 기계어(0101 1001) -> 기계

1-2. Java 언어의 탄생

- 1995년 제임스 고슬링(James Gosling)에 의해서 탄생
- 썬 마이크로시스템즈(Sun Microsystems)에서 발표
- 오크(Oak) 언어에서 시작해서 Java 언어로 발전
- 가전제품에 탑재할 수 있는 프로그램을 개발하기 위한 목적으로 탄생

1-3. Java 언어의 특징

- 초창기 시절에 Java 언어의 단점 : 기존 C/C++에 비해서 속도가 굉장히 느리다. 리소스(메모리, CPU)를 많이 사용한다.
- 현재 Java 언어의 장점 : 객체 지향 언어로 기능을 부품화할 수 있다. JRE를 이용해서 운영체제로부터 자유롭다. 웹 및 모바일 프로그래밍이 쉽다. GC를 통한 자동 메모리 관리를 지원한다. 실행 속도가 많이 개선되어 빨라졌다.

1-4. Java 프로그래밍을 위한 기본 준비물

- JDK(Java Development Kit) 설치 : JDK > JRE > API > JVM, 개발자는 JDK가 있어야 개발을 할 수 있고, 단지 프로그램만을 사용하는 사용자라면 JRE만 설치되어 있으면 된다.
- IDE(Integrated Development Environment : 통합개발환경) 설치 - eclipse

1-5. Hello Java World!!

- eclipse를 실행하고 실습할 파일을 담을 폴더를 하나 생성한 후, workspace를 설정해준다.
- 프로젝트 생성 : File - New - Project - Java Project
- 클래스 파일 생성 : src - New - Class
- 메인 메소드 생성 : main를 친 후 Ctrl + space

메인 메소드? 프로그램을 실행될 때 가장 먼저 실행되는 프로그램의 출발점 같은 존재.

- System.out.println(); : 콘솔 창에 무언가를 출력하기 위한 메소드, Sysout 치고 Ctrl + space
- 저장 후 실행 : Ctrl + S, 프로젝트 선택 후 Run As - Java Application

2. Java 프로그램의 실행 구조

2-1. 환경변수 설정

- javax.exe, java.exe를 다른 디렉터리에서도 실행할 수 있도록 하기 위해 환경변수(Path)에 bin 경로를 등록한다.
- JAVA_HOME 환경변수 추가
- Path 환경변수에 JDK의 bin 디렉터리 추가

2-2. Java 컴파일러와 JVM

- java 소스(xxx.java) -> java 컴파일러(javac.exe) -> 바이트 코드 파일(xxx.class) -> JVM 구동(java.exe) -> 기계어 -> 실행
- 기계어와 실행 : LINK(메모리 로딩 / 실행 준비 / 실행 결정 / 초기화)

2-3. Java 프로그램 실행

- 이클립스가 아닌 메모장, 컴파일러(javac.exe) 그리고 JVM을 구동시키는 java.exe를 java 프로그램을 실행한다.

2-4. 이클립스 사용의 장점

- 이클립스를 사용하면 컴파일, 디버깅 그리고 실행까지 쉽게 할 수 있다.

2-5. 가비지 컬렉터(Gabage Collector)

- 프로그램 실행에 필요한 메모리를 Gabage Collector가 자동으로 관리한다.
- C계열 프로그램 : 개발자가 직접 메모리를 관리해야 함. 만약 메모리 관리를 잘못 할 경우 메모리 누수가 발생하고 타 프로그램 동작이 멈출 수 있음.
- Java 프로그램 : 개발자가 메모리에 접근할 수 없음. 따라서 개발자는 메모리 관리를 할 수 없고, 가비지 컬렉터가 불필요한 메모리를 회수해서 메모리를 최적화함.

3. 변수

3-1. 변수란?

- 데이터(자료)를 임시로 담을 수 있는 상자(메모리 공간)

3-2. 변수 선언과 초기화

- int i = 10;일 때, 자료형은 int, 변수 이름은 i, 자료형과 변수 이름을 합쳐서 선언부라고 하고, =은 대입 연산자, 10은 변수 값이자, 초기화이다.
- 변수 선언 후 초기화 진행

```
int i;   // 변수 선언
i = 10;   // 변수 초기화
System.out.println("i = " + i);
```

- 변수 선언과 초기화를 동시에 진행

```
int j = 20;  // 변수 선언 & 초기화
System.out.println("j = " + j);
```

3-3. 메모리 할당과 진법

- int 자료형은 메모리에서 4byte 공간은 차지함.
- 8bit = 1byte

3-4. 변수 데이터 변경

- 변수에 저장된 데이터는 언제든지 변경할 수 있다.

4. 기본 자료형

4-1. 기본 자료형과 객체 자료형

- 기본 자료형은 데이터가 변수에 직접 저장되고, 객체 자료형은 객체 메모리 주소가 변수에 저장된다.
- 기본 자료형은 int i = 10;
- 객체 자료형은 0x33ec45 -> obj, 0x33ab45 -> obj, i = 0x33ec45

C계열에서 포인터라고 하고, Java에서는 레퍼런스라고 한다.

4-2. Java 기본 자료형

- 정수형 : byte - 1byte, char - 2byte, short - 2byte, int - 4byte, long - 8byte
- 실수형 : float - 4byte, double - 8byte
- 논리형 : boolean - 1byte

4-3. 형 변환

- 명시적 형 변환은 데이터가 누실될 수 있다.

5. 특수 문자와 서식 문자

5-1. 특수 문자

- 일반 문자가 아닌 특수한 목적으로 사용되는 문자

\t	탭

\n	줄 바꿈
\’	작은 따옴표
\“	큰 따옴표
\\	역슬래쉬

5-2. 서식 문자

- prinf() 메서드 이용 : f는 format(형식)을 뜻함.

%d	10진수
%o	8진수
%x	16진수
%c	문자
%s	문자열
%f	실수

5-3. 서식 문자의 정렬과 소수점 제한 기능

- 서식 문자를 이용해서 출력 문자의 정렬 및 소수점 제한 기능을 사용할 수 있다.

6. 연산자

6-1. 피연산자 개수에 의한 연산자 구분

- 피연산자 개수에 따라서 단항, 이항 그리고 삼항 연산자로 구분할 수 있다.
- 단항 연산자 : 피연산자가 하나 존재, +x, -x, !x
- 이항 연산자 : 피연산자가 두 개 존재, x = y, x(y, x !- y)
- 삼항 연산자 : 피연산자가 세 개 존재, 조건식 : true ? false

6-2. 대입 연산자

- 대입 연산자는 오른쪽의 결과를 왼쪽에 대입(할당)한다.
- ‘=’는 수학에서 ‘오른쪽 값과 왼쪽 값이 같다.’라는 의미이지만, 프로그램에서는 ‘오른쪽 값을 왼쪽에 대입’하는 의미로 쓰인다.
- 프로그램에서 ‘오른쪽과 왼쪽이 같다’ 의미는 ‘==’이다.

6-3. 산술 연산자

- 피연산자를 이용해서 +, -, *, /, % 등을 수행한다.

+	덧셈
-	뺄셈
*	곱셈
/	나눗셈
%	나머지

6-4. 복합 대입 연산자

- 산술 연산자와 대입 연산자를 결합한 연산자이다.

+=	더하고 대입
-=	빼고 대입
*=	곱하고 대입
/=	나누고 대입
%=	나머지를 대입

6-5. 관계 연산자

- 두 개의 피연산자를 비교해서 참/거짓의 결론을 돌출한다.

>	a > b : a가 b보다 크면 참
<	a < b : a가 b보다 작으면 참
>=	a >= b : a가 b보다 크거나 같으면 참
<=	a <= b : a가 b보다 작거나 같으면 참
==	a == b : a와 b가 같으면 참
!=	a != b : a와 b가 같지 않으면 참

6-6. 증감 연산자

- 1만큼 증가하거나 감소를 수행한다.

++	1만큼 증가
--	1만큼 감소

6-7. 논리 연산자

- 피연산자의 논리곱(AND), 논리합(OR), 논리부정(NOT)을 수행한다.

&&논리곱(AND)	a && b : a와 b가 모두 참이면 참
||논리합(OR)	a || b : a와 b중 하나라도 참이면 참
!논리부정(NOT)	!a : a의 상태를 부정

6-8. 조건(삼항) 연산자

- 삼항 연산자로 두 개의 피연산자 연산 결과에 따라서 나머지 피연산자가 결정된다.
- 조건식? 식1 : 식2 : 조건식이 참이면 식1이 실행되고, 조건식이 거짓이면 식2가 실행된다.

6-9. 비트 연산자

- 데이터를 비트(bit) 단위로 환산하여 연산을 수행하며, 다른 연산자보다 연산 속도가 향상된다.

& : AND 연산	a & b : a와 b가 모두 1이면 1
| : OR 연산	a | b : a와 b 중 하나라도 1이면 1
^ : XOR 연산	a ^ b : a와 b가 같지 않으면 1

7. 배열

7-1. 배열이란?

- 인덱스를 이용해서 자료형이 같은 데이터를 관리하는 것이다.

7-2. 배열 선언 및 초기화

- 배열도 변수와 마찬가지로 선언과 초기화 과정을 거쳐 사용한다.

7-3. 배열을 이용한 학사관리

- 배열은 주로 많은 데이터를 쉽게(효율적) 관리하기 위해서 사용한다.

8. 배열과 메모리

8-1. 배열의 메모리 크기

- 배열을 구성하는 데이터의 자료형에 따라서 배열의 메모리 크기가 결정된다.
- int[] arr = new int[3];
- int형 : 4byte, arr : 12byte

8-2. 배열을 가리키는 배열 이름

- 기본 자료형 데이터를 담고 있는 변수와 달리 배열 변수는 배열 데이터의 주소를 담고 있다.
- int i = 10; // 변수 i는 10
- int[] i = new int[3]; // 배열i는 i[0], i[1], i[2]

8-3. 배열 기본속성

- 기본 자료형 데이터를 담고 있는 변수와 달리 ㅂ열 변수는 배열 데이터의 주소를 담고 있다.

8-4. 다차원 배열

- 배열 안에 또 다른 배열이 존재한다.

9. 조건문

9-1. 조건문이란?

- 조건의 결과에 따라서 양자 택일 또는 다자 택일을 진행한다.
- 양자 택일은 주로 if문이 쓰인다.
- 다자 택일은 주로 switch문이 쓰인다.

9-2. if문

- if(조건식), if(조건식) else, if(조건식) else if(조건식)

9-3. switch문

- 비교 대상이 되는 결과 값과 선택사항이 많을 경우 주로 사용한다.

10. 반복문

10-1. 반복문이란?

- 프로그램 진행을 특정 조건에 따라 반복적으로 진행하는 것이다.
- for, while문 : 조건이 참일 때까지 반복 수행

[예1] 구구단을 구하기 위해서 1에서부터 1씩 더하면서 9까지 곱셈 연산을 진행한다.

[예2] 조도 센서를 센싱한 데이터가 10미만이면 건물의 LED를 1초 간격으로 계속 점등한다.

10-2. for문

- for(int i = 0; i < 10; i++) {...}
- for(int i = 1; i < 10; i++) : i가 1부터 10보다 작을 때까지 i에 1씩 더해가며 프로그램 반복 진행
- for(int i = 1; i < 10; i = i + 2) : i가 1부터 10보다 작을 때까지 i에 2씩 더해가며 프로그램 반복 진행

10-3. while문

- while(true or false) {...}
- while(rNum < 10) : rNum이가 10보다 작을 때까지 프로그램 반복 진행

10-4. do~while문

- while문과 비슷하며, 차이점은 조건 결과에 상관없이 무조건 최초 한 번은 {...}의 프로그램을 수행한다.
- do{...} while(rNum < 10) : rNum이가 10보다 작을 때까지 프로그램 반복 진행. 단, rNum의 조건에 상관없이 {...}의 프로그램은 한 번 수행한다.

11. 객체 지향 프로그래밍이란?

11-1. 객체란?

- 세상에 존재하는 모든 것을 뜻하며, 프로그래밍에서 속성과 기능을 가지는 프로그램 단위이다.
- 객체(인간세계) : 사람(키, 몸무게 / 의사), 체중계(바늘, 눈금 / 몸무게 측정), 자전거(바퀴, 체인 / 이동 수단), TV(채널, 사운드 / 미디어 방출), 승용차(바퀴, 엔진 / 이동 수단)
- 객체(프로그램) : 날씨 프로그램(온도, 미세먼지 / 날씨 예보), 사칙 연산 프로그램(+, -, *, / / 연산기능)

11-2. 클래스란?

- 객체를 생성하기 위한 틀로 모든 객체는 클래스로부터 생성된다.
- 그랜저(색상, 바퀴 / 배기량) -> 그랜저(검정, 일반 / 2000cc)

-> 그랜저(흰색, 광폭 / 2500cc)
-> 그랜저(회색, 일반 / 3000cc)

11-3. 클래스 구성요소

- 클래스는 속성(멤버 변수)와 기능(메서드)로 구성된다.
- 자전거 : 속성(멤버 변수) - 안장, 핸들, 바구니, 기어, 페달, 바퀴
  기능(메서드) - 기어 변수, 가속, 브레이크

12. 클래스 제작과 객체 생성

12-1. 클래스 제작

- 클래스는 멤버 변수(속성), 메서드(기능), 생성자 등으로 구성된다.

12-2. 객체 생성

- 클래스로부터 'new'를 이용해서 객체를 생성한다.

12-3. 생성자

- 클래스에서 객체를 생성할 때 가장 먼저 호출된다.

13. 메서드

13-1. 메서드 선언과 호출

- 메서드도 변수와 같이 선언 및 정의 후 필요시에 호출해서 사용한다.

13-2. 매개변수(parameter)

- 메서드를 호출할 때 데이터를 전달할 수 있다.
- 매개변수는 필요시에만 정의된다.
- 매개변수는 자료형과 변수(지역변수)로 이루어진다.

13-3. 중복 메서드(overloading)

- 이름은 같고, 매개변수의 개수 또는 타입이 다른 메서드를 만들 수 있다.

13-4. 접근자

- 메서드를 호출할 때 접근자에 따라서 호출이 불가할 수 있다.
- 메서드 호출부는 -X> private 메서드
  -> public 메서드

14. 객체와 메모리

14-1. 메모리에서 객체 생성(동적 생성)

- 객체는 메모리에서 동적으로 생성되며, 객체가 더 이상 필요 없게 되면 GC(Gabage Cdollector)에 의해서 제거된다.

14-2. 레퍼런스(가리킨다)

- 생성한 객체의 주소를 변수에 저장하는 것을 레퍼런스라고 한다.

14-3. 자료형이 같아도 다른 객체

- 자료형이 같아도 다른 공간에 존재하는 객체는 다른 객체이다.

14-4. null과 NullPointException

- 레퍼런스에 null 저장되면 객체의 연결이 끊기며, 더 이상 객체를 이용할 수 없다.

15. 생성자와 소멸자 그리고 this 키워드

15-1. 디폴트 생성자

- 객체가 생성될 때 가장 먼저 호출되는 생성자로, 만약 개발자가 명시하지 않아도 컴파일 시점에 자동

생성된다.

15-2. 사용자 정의 생성자

- 디폴트 생성자 외에 특정 목적에 의해서 개발자가 만든 생성자로, 매개변수에 차이가 있다.

15-3. 소멸자

- 객체가 GC에 의해서 메모리에서 제거될 때 finalize() 메서드가 호출된다.
- System.gc();를 사용한다고 해서 GC가 바로 작동하는 것이 아니라, 가급적 빨리 작동하도록 요청하는 것이다.
- java는 기본적으로 메모리를 개발자가 직접 관리하지 않으므로 일반적으로 System.gc();를 사용하는 경우는 드물다.

15-4. this 키워드

- 현재 객체를 가리킬 때 this를 사용한다.

16. 패키지와 static

16-1. 패키지(pakacage)

- java 프로그램은 많은 클래스로 구성되고, 이러한 클래스를 폴더 형식으로 관리하는 것을 패키지라고 한다.
- 패키지 이름 결정 요령

패키지 이름은 패키지에 속해있는 클래스가 최대한 다른 클래스와 중복되는 것을 방지하도록 만든다.
패키지 이름은 일반적으로 도메인을 거꾸로 이용한다.
개발 중에 패키지의 이름과 구조는 변경될 수 있다.
패키지 이름만 보고도 해당 패키지 안에 있는 클래스가 어떤 속성과 기능을 가지고 있는 예상이 될 수 있도록 이름을 만든다.

16-2. import

- 다른 패키지에 있는 클래스를 사용하기 위해서는 import 키워드를 사용한다.

16-3. static

- 클래스의 속성과 메서드에 static 키워드를 사용하면 어디서나 속성과 메서드를 공유할 수 있다.

17. 데이터 은닉

17-1. 멤버 변수의 private 설정

- 멤버 변수(속성)는 주로 private으로 설정해서, 외부로부터 데이터가 변질되는 것을 막는다.
- 데이터(속성) 변질이 우려되는 시나리오

직원 급여 프로그램에서, 급여 담당자가 실수로 급여액을 잘못 입력하는 경우

- 변경될 수 없는 사번이 변경되는 경우

17-2. setter, getter

- 멤버 변수를 외부에서 변경할 수 있도록 하는 메서드이다.

18. 상속

18-1. 상속이란?

- 부모 클래스를 상속받은 자식 클래스는 부모 클래스의 속성과 기능도 이용할 수 있다.
- 아들(집 + 자동차 + 요트) -상속> 아버지(집 + 아버지) -상속> 할아버지(집)
- child class(속성 + 기능) -상속> parent class(속성, 기능) -생성> 객체(parent class + child class)

18-2. 상속의 필요성

- 기존의 검증된 class를 이용해서 빠르고 쉽게 새로운 class를 만들 수 있다.

- G음식점(할아버지 운영) : 모든 메뉴를 새롭게 만들어야 한다. -비법 계승> P음식점(아버지 운영) : P음식점의 메뉴는 G음식점의 메뉴에 새로운 몇 가지만 추가하면 된다. -비법 계승> C음식점(아들 운영) : C음식점의 메뉴는 P음식점의 메뉴에 새로운 몇 가지만 추가하면 된다.
- 즉, P와 C는 검증된 메뉴를 이용해서 쉽고 빠르게 개업할 수 있다.

18-3. 상속 구현

- extend 키워드를 이용해서 상속을 구현한다.

18-4. 부모 클래스의 private 접근자

- 자식 클래스는 부모 클래스의 모든 자원을 사용할 수 있지만, private 접근자의 속성과 메서드는 사용할 수 없다.

19. 상속 특징

19-1. 메서드 오버라이드(override)

- child class(더 맛있는 짜장면 기능) -상속> parent class(짜장 만드는 기능
  -생성> 객체(parent class + child class)

19-2. 자료형(타입)

- 기본 자료형처럼 클래스도 자료형이다.

19-3. Object 클래스

- 모든 클래스의 최상위 클래스는 Object 클래스이다.
- class -> class -> class -> ... -> class -> Object

19-4. super 클래스

- 상위 클래스를 호출할 때 super 키워드를 이용한다.

20. 내부 클래스와 익명 클래스

20-1. 내부(inner) 클래스

- 클래스 안에 또 다른 클래스를 선언하는 것으로 이렇게 하면 두 클래스의 멤버에 쉽게 접근할 수 있다.

20-2. 익명(anonymous) 클래스

- 이름이 없는 클래스로 주로 메서드를 재정의 하는 목적으로 사용된다.
- 익명 클래스는 인터페이스나 추상 클래스에서 주로 이용된다.

21. 인터페이스

21-1. 인터페이스란?

- 클래스와 달이 객체를 생성할 수는 없으며, 클래스에서 구현해야 하는 작업 명세서이다.
- class -생성> 객체
- interface -생성X> 객체, class -구현> interface, class -생성> 객체

21-2. 인터페이스를 사용하는 이유

- 인터페이스를 사용하는 이유는 많지만, 가장 큰 이유는 객체가 다양한 자료형(타입)을 가질 수 있기 때문이다.
- class -> interfaceA, interfaceB, interfaceC, interfaceD

21-3. 인터페이스 구현

- class 대신 interface 키워드를 사용하며, extend 대신 implements 키워드를 이용한다.

21-4. 장난감 인터페이스

- interface를 이용하면 객체가 다양한 자료형(타입)을 가질 수 있다.

- ToyRobot -> Toy, ToyAirplane -> Toy

22. 추상 클래스

22-1. 추상 클래스란?

- 클래스의 공통된 부분을 뽑아서 별도의 클래스(추상 클래스)로 만들어 놓고, 이것을 상속해서 사용한다.
- class -상속> abstract class, class -생성> 객체
- abstract 클래스의 특징 : 멤버 변수를 가진다. abstract 클래스를 상속하기 위해서는 extends를 이용한다. abstract 메서드를 가지며, 상속한 클래스에서 반듯이 구현해야 한다. 일반 메서드도 가질 수 있다. 일반 클래스와 마찬가지로 생성자도 있다.

22-2. 추상 클래스 구현

- 클래스 상속과 마찬가지로 extends 키워드를 이용해서 상속하고 abstract(추상) 메서드를 구현한다.

22-3. Bank 추상 클래스

- MyBank 클래스 -상속> abstract Bank 클래스

22-4. 인터페이스 vs 추상 클래스

- 공통점 : 추상 메서드를 가진다. 객체 생성이 불가하며 자료형(타입)으로 사용된다.
- 차이점 : 인터페이스는 상수, 추상 메서드만 가지고, 추상 메서드를 구현만 하도록 하고, 다형성을 지원한다. 추상 클래스는 클래스가 가지는 모든 속성과 기능을 가지고 추상 메서드 구현 및 상속의 기능을 가지고 단일 상속만 지원한다.

23. 람다식

23-1. 람다식이란?

- 익명 함수(anonymous function)를 이용해서 익명 객체를 생성하기 위한 식이다.
- 기존 방법 : InterfaceType 변수 <할당(대입)- Interface 구현
-> 람다식 방법 : Lambda Expressions -할당(대입)> InterfaceType 변수

23-2. 람다식 구현

- 람다식은 기본적으로 함수를 만들어 사용한다고 생각하면 된다.

24. 문자열 클래스

24-1. String 객체와 메모리

- 문자열을 다루는 String 클래스(객체)는 데이터가 변하면 메모리 상의 변화가 많아 속도가 느리다.
- 문자열이 변경되면 기존의 객체를 버리고, 새로운 객체를 메모리에 생성한다. 이때, 기존 객체는 GC에 의해서 메모리 회수가 이루어진다.

24-2. StringBuffer, StringBuilder

- String 클래스의 단점을 보완한 클래스로 데이터가 변경되면 메모리에서 기존 객체를 재활용한다.
- 문자열이 변경되면 기존의 객체를 재활용한다.
- 속도는 StringBuilder가 조금 빠르며, 데이터 안정성은 StringBuffer가 조금 더 좋다.

25. Collections

25-1. List

- List는 인터페이스로 이를 구현한 클래스는 인덱스를 이용해서 데이터를 관리한다.
- 인덱스를 이용하고, 데이터 중복이 가능하다.

25-2. Map

- Map은 인터페이스로 이를 구현한 클래스는 key를 이용해서 데이터를 관리한다.

- key를 이용하고, key는 중복될 수 없다. 하지만, 데이터 중복은 가능하다.

26. 예외 처리

26-1. 예외란?

- 프로그램에 문제가 있는 것을 말하며, 예외로 인해 시스템 동작이 멈추는 것을 막는 것을 '예외 처리'라고 한다.
- Exception(Error는 개발자가 대처할 수 있음) vs Error(Error는 개발자가 대처할 수 없음)
- Exception -> Checked Exception('예외 처리'를 반듯이 해야 하는 경우(네트워크, 파일 시스템 등)), Unchecked Exception('예외 처리'를 개발자의 판단에 맞기는 경우(데이터 오류 등))

26-2. Exception 클래스

- Exception 클래스 하위 클래스로 NullPointerException, NumberFormatException 등이 있다.
- NullPointerException : 객체를 가리키지 않고 있는 레퍼런스를 이용할 때
- ArrayIndexOutOfBoundException : 배열에서 존재하지 않는 인덱스를 가리킬 때
- NumberFormatException : 숫자 데이터에 문자 데이터 등을 넣었을 때

26-3. try ~ catch

- 개발자가 예외 처리 하기 가장 쉽고, 많이 사용되는 방법이다.

26-4. 다양한 예외 처리

- Exception 및 하위 클래스를 이용해서 예외 처리를 다양하게 할 수 있다.

26-5. finally

- 예외 발생 여부에 상관없이 반듯이 실행된다.

26-6. throws

- 예외 발생 시 예외 처리를 직접 하지 않고 호출한 곳으로 넘긴다.

27. 입력과 출력

27-1. 입/출력 이란?

- 다른 곳의 데이터를 가져오는 것을 입력이라 하고, 다른 곳으로 데이터를 내보내는 것을 출력이라고 한다.
- 입력(Input) : 파일 읽기, 이미지 & 동영상 불러오기
- 출력(Output) : 파일 쓰기, 이미지 & 동영상 내보내기

27-2. 입/출력 기본 클래스

- 입/출력에 사용되는 기본 클래스는 1byte단위로 데이터를 전송하는 InputStream, OutputStream이 있다.
- InputStream : FileInputStream, DataInputStream, BufferedInputStream
- OutputStream : FileOutputStream, DataOutputStream, BufferedOutputStream

27-3. FileInputStream / FileOutputStream

- 파일에 데이터를 읽고/쓰기 위한 클래스로 read(), write() 메서드를 이용한다.
- read() : 1byte씩 읽고, read(byte[]) : []크기만큼 읽고
- write(byte[] b) : 전체 쓰기, write(byte[], int off, int len) : off(시작점), len(길이)

27-4. 파일 복사

- 파일 입/출력 클래스를 이용해서 파일을 복사할 수 있다.
- File -입력> InputStream, OutputStream -출력> File

27-5. DataInputStream, DataOutputStream

- byte 단위의 입출력을 개선해서 문자열을 좀 더 편리하게 다룰 수 있다.

- File -입력> DataInputStream(->InputStream), DataOutputStream(->OutputStream) -출력> File

27-6. BufferedReader, BufferedWriter

- byte 단위의 입출력을 개선해서 문자열을 좀 더 편리하게 다룰 수 있다.
- File -입력> BufferedReader(->FileReader), BufferedWriter(->FileWriter) -출력> File

28. 네트워킹

28-1. 네트워크 데이터 입력 및 출력

- 네트워크 대상(객체) 사이에 입/출력(InputStream, OutputStream)를 이용해서 데이터를 입력하고 출력한다.

28-2. 소켓(Socket)

- 네트워크상에서 데이터를 주고받기 위한 장치이다.

28-3. Socket 클래스

- 서버는 클라이언트를 맞을 준비를 하고 있다가 클라이언트의 요청에 반응한다.

28-4. Client와 Server 소켓(Socket)

- 서버에 ServerSocket를 준비하고 클라이언트에서 Socket를 이용해서 접속한다.

28-5. 양방향 통신

- 클라이언트와 서버는 InputStream, OutputStream을 이용해서 양방향 통신을 할 수 있다.