쉽게 배우는 자바(1)

1. Java 프로그래밍 소개

1) Java

- 컴퓨터 프로그래밍 언어 중 하나, 자바

컴퓨터 프로그래밍 언어의 하나인 자바 (JAVA)는 1991년 제임스 고슬링(James Gosling)에 의해 만들어진 언어입니다. 자바는 '한 번 작성하면 어디서든 실행된다'는 목표를 가지고 만들어진 언어이며, 자바로 작성된 소프트웨어는 자바가 설치된 컴퓨터라면 어디서든 실행이 가능하다는 특징을 가지고 있습니다.

- 자바의 활용

2019년 기준 티오베(www.tiobe.com)의 프로그래밍 언어 인기 순위에서 자바는 1위를 차지하고 있습니다.

Aug 2019	Aug 2018	Change	Programming Language	Ratings	Change
1	1		Java	16.028%	-0.85%
2	2		C	15.154%	+0.19%
3	4	^	Python	10.020%	+3.03%
4	3	v	C++	6.057%	-1.41%
5	6	^	C#	3.842%	+0.30%
6	5	v	Visual Basic .NET	3.695%	-1.07%
7	8	^	JavaScript	2.258%	-0.15%
8	7	v	PHP	2.075%	-0.85%
9	14	^^	Objective-C	1.690%	+0.33%
10	9	v	SQL	1.625%	-0.69%

https://www.tiobe.com/tiobe-index/

출처: https://www.tiobe.com/tiobe-index/

이렇게 인기있는 언어인 자바는 웹 및 안드로이드의 애플리케이션 소프트웨어 개발에 활용될 수 있습니다.

2. 개발환경 Java 설치

1) 온라인 실습 환경 장만하기

- 자바를 온라인에서 실습하기

자바를 배우고 실습하기 위해서는 각자의 컴퓨터 환경(로컬 컴퓨터)에 자바를 설치하는 과정이 필요합니다. 하지만 설치 과정에 너무 많은 시간이 들게 된다면 더 이상 지체하지 않고 우선 온라인 편집기를 이용해 실습을 진행합니다. 자바를 위한 온라인 편집기는 여러 종류가 있지만 JDoodle(https://www.jdoodle.com/)을 이용해 보도록 합니다.

- JDoodle 열기 및 Java 언어 선택 : JDoodle을 브라우저에서 열고 온라인 편집기 구역에서 Java 언어를 선택합니다.

- 편집기 구성

편집 구역 : 실제 자바 코드를 작성하는 구연으로 main 구역 내부에서 코드를 작성해 원하는 기능을 구현할 수 있습니다.

입력 및 실행, 결과 구역 : 작성한 코드를 실행할 때 사용할 아규먼트를 CommandLine

Arguments 부분에 입력할 수 있습니다. 여러 개의 인수는 띄어쓰기로 구분하여 입력하고, 인덱스(ex: args[0], args[1])를 이용하여 각각 사용할 수 있습니다. Execute 버튼을 클릭하여 코드를 실행하면 Result 구역에서 실행 결과가 나타나게 됩니다.

2) Java 설치_MacOS

- JDK란? JDK는 Java Development Kit로 자바 소프트웨어 개발을 위한 여러 도구의 구성을 의미합니다. 자바를 설치할 때는 JDK의 형태로 설치하게 됩니다. 자바는 2020년 기준 14 버전까지 나와 있으며 이 강의를 위해서는 어떤 버전을 사용해도 무관합니다.

(이 강의에서는 JDK 8 버전을 사용)

※ 주의사항

2019년 4월 16일 기준으로 Oracle의 Java 라이센스 규정이 변경되어, Oracle에서 Java SE 8 버전의 JDK를 다운로드 하기 위해서는 오라클 계정이 필요합니다. 따라서 오라클 계정을 미리 생성할 필요가 있습니다. 다만 최신 버전의 경우 계정을 생성할 필요가 없으니 유의하시길 바랍니다.

- JDK 다운로드 페이지 접속하기 : 포털 사이트의 검색을 통해 Oracle Java SE Development Kit 다운로드 페이지에 접속합니다.

https://www.oracle.com/kr/java/technologies/javase-downloads.html

설치하고자 하는 자바 버전의 항목에서 JDK download 버튼을 클릭합니다.

- Mac OS용 JDK 파일 다운로드 : MacOS에 해당하는 설치 파일을 클릭하여 다운로드 합니다.

- JDK 설치 : Continue, Install, Close 버튼을 차례로 클릭하여 JDK를 설치합니다.

- Java 설치 확인 : Spotlight(단축키: Command + Space)에서 Terminal을 입력하여 터미널을 실행합니다. 터미널에서 java -version을 입력하여 자바의 버전을 확인합니다. javac -version을 입력하여 자바 컴파일러의 버전을 확인합니다.

3) Java 설치_Linux

- 리눅스에서 자바 설치하기 : 리눅스는 여러 종류가 있습니다. 그중 리눅스의 일종인 우분투(Ubuntu)에서 JDK를 설치하는 방법을 알아봅니다.

- 자바 버전 확인하기 : 터미널에서 java -version을 입력하여 자바가 설치되어 있는지, 설치되어 있다면 무슨 버전인지 확인합니다.

- 우분투 패키지 목록 업데이트 및 설치 : 우분투는 커맨드라인의 apt 명령어를 통해 패키지를 관리합니다. 우선 패키지 목록을 최신 상태로 갱신하기 위해 sudo apt update 명령어를 입력하여 실행합니다. 이후 sudo apt install default-jdk의 명령어로 자바를 설치합니다.

- 설치 완료 후 설치 확인 : 설치 완료된 후 다시 자바의 버전을 확인하는 java -version 명령어를 입력하여 설치된 자바의 버전을 확인합니다.

https://itsfoss.com/install-java-ubuntu/

https://story.pxd.co.kr/732

4) Java 설치_윈도우

- 윈도우에서 자바 설치하기

※ 주의사항

Java SE 9 버전 이후부터는 32bit(x86) 윈도우용 JDK 설치 파일을 제공하지 않습니다. 만일 32bit 운영체제에서 설치하는 경우 Java SE 8 버전 JDK의 설치를 시도합니다.

운영체제의 종류를 확인하기 위해서는, 파일 탐색기 → 내 PC 항목 오른쪽 클릭 → 속성 →

시스템 종류에서 확인할 수 있습니다.
한편, 2019년 4월 16일 기준으로 Oracle의 Java 라이센스 규정이 변경되어, Oracle에서 Java SE 8 버전의 JDK를 다운로드 하기 위해서는 오라클 계정이 필요합니다. 따라서 오라클 계정을 미리 생성할 필요가 있습니다.

- 자바 버전 확인하기 : 실행(단축키 : 윈도우 키 + R)에서 cmd를 입력해 명령 프롬프트를 실행합니다. java -version 명령어를 입력하고 실행해 자바의 설치 유무 및 설치 버전을 확인합니다. 자바 컴파일러의 설피 유무 설치 버전도 javac -version 명령어를 이용해 확인합니다.
- JDK 다운로드 페이지 접속하기 : 포털 사이트의 검색을 통해 Oracle Java SE Development Kit 다운로드 페이지에 접속합니다.

https://www.oracle.com/kr/java/technologies/javase-downloads.html

설치하고자 하는 자바 버전의 항목에서 JDK download 버튼을 클릭합니다.

- Windows용 JDK 설치하기 : Windows에 해당하는 설치 파일을 클릭하여 다운로드 합니다. 32비트 운영체제의 경우 Windows x86에 해당하는 설치 파일을 다운로드 합니다.
- JDK 설치 : Next, Next, Close 버튼을 클릭하여 설치를 완료합니다.
- 설치 확인 및 환경 변수 설정 : 명령 프롬프트를 재시작하여 java -version 명령어를 입력합니다. 만약 버전이 확인되지 않는다면, 자바의 설치경로를 환경변수로 지정하여야 합니다. 우선 자바가 설치된 경로로 들어갑니다. 설치 경로 내의 bin 폴더 안에 java.exe와 javac.exe라는 실행파일이 들어 있는 것을 볼 수 있습니다. 이 두 파일이 포함되어 있는 폴더(자바 설치 경로\bin)를 환경 변수에 지정합니다.

파일 탐색기 → 내 PC 항목 오른쪽 클릭 → 속성 과정으로 내 PC의 속성에 들어갑니다. 고급 시스템 설정에서 환경 변수 버튼을 클릭합니다. 사용자 변수의 Path를 더블클릭하거나, Edit 버튼을 눌러 편집기를 연 후 New 버튼을 눌러 새로운 경로를 추가합니다. 추가된 새로운 경로에 java.exe와 javac.exe가 포함되어 있는 폴더의 경로(자바 설치 경로\bin)을 입력합니다. 명령 프롬프트를 재실행한 후 아래의 명령어를 차례로 입력하여 제대로 자바의 버전이 출력되는지 확인합니다.

```
java -version
javac -version
```

3. 개발환경 eclipse 설치

1) Java 개발환경 eclipse 설치하기

- 이클립스 설치하기 : 이클립스는 자바로 프로그램을 만들 때 그 제작을 쉽게 해 주는 도구로, 많은 자바 개발자들의 사랑을 받고 있습니다. 이클립스와 같은 도구를 IDE(Intergrated Development Environment, 통합 개발 환경)이라고 합니다.
- 설치 파일 다운로드 : 이클립스 홈페이지(https://www.eclipse.org/)에 방문하여 Download 버튼을 찾아서 클릭합니다. Download 버튼을 눌러 설치 파일을 다운받습니다. 만약 다운로드에 문제가 생기게 되면 아래의 Select Another Mirror를 클릭하고 다른 지역을 선택하여 다운로드 받습니다.
- 이클립스 설치 및 실행하기 : Eclipse IDE for Java Developers를 클릭하여 다음 단계로 진행합니다. 자바가 설치된 경로와 이클립스가 설치될 경로를 확인한 후 Install 버튼을 눌러

설치를 진행합니다. 설치가 완료되면 Launch 버튼을 눌러 이클립스를 실행합니다. Launch 버튼을 클릭하여 실행합니다.

4. Java 실행

1) 실행 HelloWorld

- 이클립스 화면 조절하기 : 우선 이클립스에서 Task List와 Outline과 같은 필요 없는 도구들을 보이지 않게 합니다. 그리고 Window - Show View 메뉴에서 Navigator를 클릭하여 활성화합니다.

- 자바 프로젝트 만들기 : Create a Java Project를 눌러 자바 프로젝트를 생성합니다. 프로젝트 이름은 HelloWorld로, Location에는 앞으로 실습을 진행할 작업 공간에 같은 이름의 폴더를 입력합니다. Project layout은 첫 번째 옵션, 소스와 클래스 파일을 같은 프로젝트 폴더에 두는 옵션을 선택합니다. Finish를 눌러 프로젝트를 생성합니다. Navigator 뷰에서 HelloWorld 프로젝트를 마우스 오른쪽 버튼으로 클릭한 후, New - File을 클릭하여 새로운 파일을 생성합니다. 파일명에 HelloWorldApp.java를 입력하고 Finish를 눌러 자바 파일을 생성합니다. HelloWorldApp에 입력하고 저장합니다.

```
public class HelloWorldApp {
    public static void main(String args[]) {
        System.out.println("Hello World!!");
    }
}
```

저장을 하고 Project Explorer에서는 변화가 없지만, Navigator 뷰를 보면 저장한 자바 파일과 동일한 이름의 클래스 파일이 생성된 것을 확인할 수 있습니다. 편집기 내에서 마우스 오른쪽 버튼을 클릭하여 팝업 메뉴를 띄운 후 Run As - 1. Java Application을 클릭하여 실행을 합니다. 또는 이클립스 상단 툴바의 같은 모양의 버튼을 클릭해서 실행할 수 있습니다. 실행하면 하단의 콘솔 뷰에서 Hello World!!가 출력되는 것을 확인할 수 있습니다.

https://github.com/egoing/java1

2) AS 강의 Editor deos not contain a main type

- 오류의 원인 : Editor dose not contain a main type이라는 오류가 나타났다면 이클립스가 소스 코드가 컴파일된 class 파일을 찾지 못하였기 때문입니다. 왜 찾지 못하였을까요? 다시 프로젝트를 생성해 봅시다. 이번에는 MainType이라는 프로젝트를 생성하고 Project layout에서 두 번째 옵션을 선택해 봅니다. 옆에 Configure default를 클릭해 봅니다. 그러면 위와 같이 Source folder name과 Output folder name에 각각 src, bin폴더로 입력되어 있는 것을 볼 수 있습니다. src는 source, bin은 binary를 뜻합니다. 이러한 상태에서 이클립스는 소스 코드는 src 폴더에 저장하고 실행 파일은 bin 폴더에 저장하게 됩니다. 다음 단계로 넘어가면 다음과 같이 MainType 프로젝트의 src폴더가 기본 소스코드 폴더가 되고, bin폴더가 기본 실행파일 폴더가 됩니다. Package Explorer 뷰를 통해 보면 src 폴더가 기본 소스 폴더가 된 것을 볼 수 있습니다. 이 프로젝트에서 MainType 프로젝트의 루트 경로(가장 상위 폴더)에 자바 파일을 생성해 봅니다. MyApp.java 파일을 생성하고 코드를 입력합니다.

```
public class MyApp {
```

```
    public static void main(String args[]) {
        System.out.println("Hello World!!");
    }
}
```

저장한 후 실행을 시켜 보면 같은 오류가 발생되는 것을 볼 수 있습니다. 왜냐하면, 작성한 MyApp.java 파일이 기본 소스 코드 폴더인 src 안에 있지 않기 때문에 이클립스는 MyApp 파일을 소스 코드로 인식하지 않기 때문입니다. 이클립스에서 자바 애플리케이션을 실행할 때 src 폴더의 소스 파일은 없기 때문에 오류가 발생하게 됩니다.

- 오류를 해결하는 방법 : 오류를 해경하는 방법은 세 가지가 있습니다. 첫 번째는 작성한 MyApp.java 파일을 src 폴더로 옮기는 것입니다. 두 번째는 Project layout 부분의 첫 번째 옵션을 선택해 다시 새로운 프로젝트를 생성하는 것입니다. 마지막으로는 프로젝트의 속성을 변경하는 것입니다.

프로젝트 항목에서 팝업 메뉴를 띄운 후 Properties 버튼을 클릭합니다. Java Build Path 메뉴에서 기존의 소스 폴더인 src 폴더를 Remove 버튼을 클릭하여 삭제합니다. 그리고 Add Folder 버튼을 클릭하여 소스 폴더를 프로젝트의 루트 폴더로 새로 지정합니다. 마찬가지로 하단의 Default output folder에도 똑같이 루트 폴더를 입력합니다. 다시 MyApp.java를 실행시켜 보면 잘 실행되는 것을 확인할 수 있습니다.

- 응용 과정 : 프로젝트의 속성을 변경하면 소스 폴더를 복수 개를 만들고, 컴파일된 실행 파일을 같은 폴더에 저장할 수 있지 않을까요? 위와 같이 src1, src2 폴더를 만들고 그 안에 각각 아래와 같이 MyApp1.java, MyApp2.java파일을 생성합니다.

MyApp1.java

```
public class MyApp1 {
    public static void main(String args[]) {
        System.out.println("Hello World!!");
    }
}
```

MyApp2.java

```
public class MyApp2 {
    public static void main(String args[]) {
        System.out.println("Hello World!!");
    }
}
```

그리고 프로젝트의 설정에서 Java Build Path를 설정합니다. 그리고 저장하게 되면 아래와 같이 bin 폴더 안에 MyApp1과 MyApp2의 컴파일된 파일들이 저장되는 것을 볼 수 있습니다.

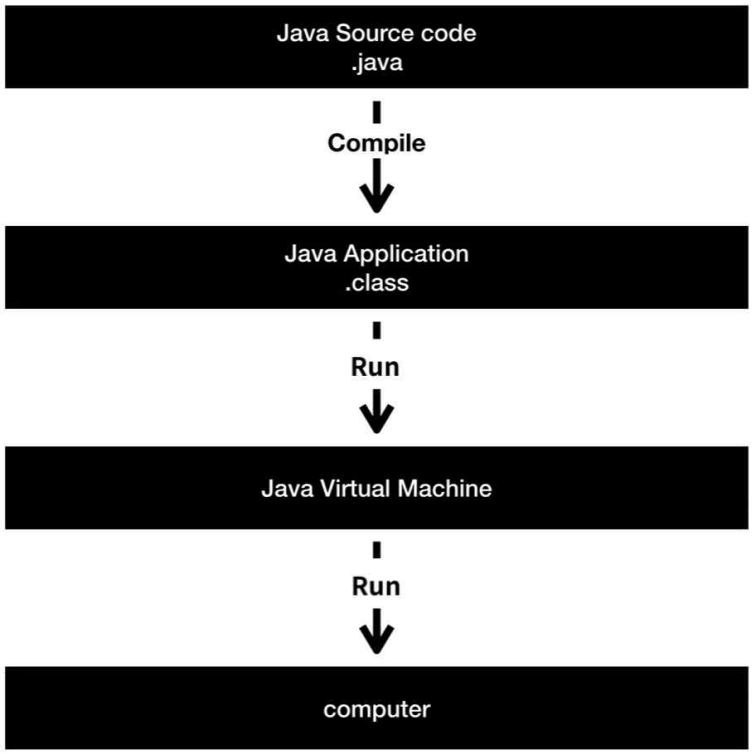
3) 실행 Java의 동작 원리

- 자바의 동작 원리 : “Hello World!!”를 출력하는 기능을 구현하는 프로그램(=애플리케이션)을 만들었습니다. 이 기능을 구현하기 위해 아래와 같은 소스를 만들었습니다.

```
public class HelloWorldApp {
    public static void main(String args[]) {
        System.out.println("Hello World!");
```

```
    }
}
```

소스는 소프트웨어가 구현되는 원인이라는 의미에서 소스로 불리기도 하며, 부호라는 의미에서 코드로도 불립니다. 마찬가지로 의미를 전달하는 약속이라는 측면에서 언어로 불리기도 합니다. 자바 소스가 프로그램으로서 동작하는 과정은 아래와 같습니다.



우선 java 파일의 형태로 소스 코드를 작성하면, 컴파일 과정을 거쳐 JVM(Java Virtual Machine, 자바 가상 머신)이 알아들을 수 있는 class 파일로 변환됩니다. 이클립스에서는 자바 파일을 저장할 때 컴파일을 하여 class 파일로 저장합니다. 이클립스에서 실행 버튼을 누르게 되면 JVM에서 class 파일을 읽어들입니다. JVM은 class 파일을 읽고서 그대로 컴퓨터를 동작시키게 됩니다.

https://asfirstalways.tistory.com/158

5. Java로 할 수 있는 일

1) Hello Java World

- Hello World라는 문자열을 화면에 출력하는 프로그램을 만들어 보았습니다. 아직은 구체적으로 프로그램을 만드는 방법에 대해서 알지는 못하지만, 앞으로 배워서 알게 된다면, 원하는 무언가를 화면에 출력할 수 있다는 것을 파악할 수 있습니다. 하지만 이것만으로는 아직 감이 잘 오지 않을 수도 있습니다. 자바를 이용해서 데스크탑, IoT, 안드로이드의 애플리케이션을 만드는 모습을 보고, 자바로 이런 일도 가능하다는 것을 느낄 수 있습니다.

2) 데스크톱 애플리케이션 만들기

- 자바로 데스크탑 앱 만들기

```
import javax.swing.*;
import java.awt.Dimension;
import java.awt.Toolkit;
public class HelloWorldGUIApp{
    public static void main(String[] args){
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                JFrame frame = new JFrame("HelloWorld GUI");
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.setPreferredSize(new Dimension(400, 300));
                JLabel label = new JLabel("Hello World!!", SwingConstants.CENTER);
                frame.getContentPane().add(label);
                Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
                frame.setLocation(dim.width/2-400/2, dim.height/2-300/2);

                frame.pack();
                frame.setVisible(true);
            }
        });
    }
}
```

Eclipse에서 HelloWorldGUI 프로젝트를 생성하고 위와 같이 HelloWorldGUIApp.java를 만들어서 실행해 봅니다. 이 프로그램은 작은 데스크탑 창이 뜨고 그 안에 Hello World!!가 나오는 기능을 수행합니다.

```
JLabel label = new JLabel("Hello World!!", SwingConstants.CENTER);
```

11번째 줄의 코드가 작은 윈도우에 Hello World!!를 나타내 주는 코드이고, 나머지 부분은 데스크탑 창을 구성하기 위한 부분들입니다.

프로그램은 원인과 결과로 구성되어 있습니다. 코드가 원인이 되어 실행된 결과가 프로그램입니다. 이번에는 코드의 다른 부분이 어떤 결과를 가져오는지 한번 살펴 봅니다.

```
frame.setPreferredSize(new Dimension(800, 300));
```

10번째 줄에서 400을 800으로 고쳐 봅니다.

오른쪽에 Hello World!!가 나타난 것을 확인할 수 있습니다.

소스 코드는 github(https://github.com/egoing/java1)의 HelloWorldGUI 프로젝트에 있습니다.

3) 사물을 자바로 제어하기

- 사물인터넷 : 컴퓨터는 시대가 바뀌면서 속도가 빨라지고, 비용이 저렴해지고, 크기가 작아져 왔습니다. 이 세 가지 특성이 일정 단계에 도달했을 때 사회적으로 큰 변화를 이끌어 왔습니다. 컴퓨터는 빠르고 저렴해지고 크기가 작아지면서 이제는 작은 전구에도 들어갈 수 있을 정도가 되었습니다. 전구에 작은 컴퓨터가 들어갈 수 있다면, 그 컴퓨터에 자바를 설치해서

프로그램을 넣어서 프로그램으로 제어하는 스마트 전구를 만들 수 있게 됩니다.
이 컴퓨터에 통신기술도 포함시키게 된다면, 이것을 사물을 인터넷으로 연결되어 프로그램으로 제어하는 사물인터넷, 즉 IoT(Internet of Things)라고 부르게 됩니다.
 - 자바로 사물을 제어하기 : 라즈베리 파이라는 작은 컴퓨터를 이용해서 자바로 전구으이 동작을 제어하는 앱을 살펴봅니다. 라즈베리파이는 작고 가격이 비싸지 않은 교육용 컴퓨터입니다. 여러 구성 요소 중 전류를 흐르게 할 수 있는 GPIO(General Purpose Input Output)가 있는데, 각각의 핀들은 프로그램으로 전류의 흐름을 제어할 수 있습니다. 이를 응용하여 원하는 시간에 화분에 물을 주거나, 원격으로 조명을 켜는 등 세상의 수많은 사물을 기계적이고 자동화된 방법으로 제어할 수 있게 됩니다.
 Hello World 중 "Hell"부분의 모스부호 대로 LED전구를 점멸하게 하는 앱을 살펴 봅니다.
 HelloWorldRaspberryPi.java

```
import com.pi4j.io.gpio.GpioController;
import com.pi4j.io.gpio.GpioFactory;
import com.pi4j.io.gpio.GpioPinDigitalOutput;
import com.pi4j.io.gpio.PinState;
import com.pi4j.io.gpio.RaspiPin;

public class HelloWorldRaspberryPi {

	public static void main(String[] args) throws InterruptedException {

		final GpioController gpio = GpioFactory.getInstance();
		final GpioPinDigitalOutput pin = gpio.provisionDigitalOutputPin(RaspiPin.GPIO_01, "PinLED", PinState.LOW);
		final int SHORT_INTERVAL = 200;
		final int LONG_INTERVAL = SHORT_INTERVAL * 3;
		final int LETTER_INTERVAL = SHORT_INTERVAL * 7;

		while (true) {
			// H
			pin.high();
			Thread.sleep(SHORT_INTERVAL);
			pin.low();
			Thread.sleep(SHORT_INTERVAL);
			pin.high();
			Thread.sleep(SHORT_INTERVAL);
			pin.low();
			Thread.sleep(SHORT_INTERVAL);
			pin.high();
			Thread.sleep(SHORT_INTERVAL);
			pin.low();
```

```
Thread.sleep(SHORT_INTERVAL);
pin.high();
Thread.sleep(SHORT_INTERVAL);
pin.low();
Thread.sleep(LETTER_INTERVAL);

// e
pin.high();
Thread.sleep(SHORT_INTERVAL);
pin.low();
Thread.sleep(LETTER_INTERVAL);

// l
pin.high();
Thread.sleep(SHORT_INTERVAL);
pin.low();
Thread.sleep(SHORT_INTERVAL);

pin.high();
Thread.sleep(LONG_INTERVAL);
pin.low();
Thread.sleep(SHORT_INTERVAL);

pin.high();
Thread.sleep(SHORT_INTERVAL);
pin.low();
Thread.sleep(SHORT_INTERVAL);
pin.high();
Thread.sleep(SHORT_INTERVAL);
pin.low();
Thread.sleep(LONG_INTERVAL);

// l
pin.high();
Thread.sleep(SHORT_INTERVAL);
pin.low();
Thread.sleep(SHORT_INTERVAL);

pin.high();
Thread.sleep(LONG_INTERVAL);
pin.low();
```

```
                    Thread.sleep(SHORT_INTERVAL);

                    pin.high();
                    Thread.sleep(SHORT_INTERVAL);
                    pin.low();
                    Thread.sleep(SHORT_INTERVAL);
                    pin.high();
                    Thread.sleep(SHORT_INTERVAL);
                    pin.low();
                    Thread.sleep(LONG_INTERVAL);
                }
        }
}
```

라즈베리파이의 커널에서 다음과 같은 명령어를 입력하여 HelloWorldRaspberryPi 앱을 실행합니다.

javac -cp ".:/opt/pi4j/lib/*" HelloWorldRaspberryPi.java ; java -cp ".:/opt/pi4j/lib/*" HelloWorldRaspberryPi

https://ko.wikipedia.org/wiki/%EC%82%AC%EB%AC%BC%EC%9D%B8%ED%84%B0%EB%84%B7

https://ko.wikipedia.org/wiki/%EB%9D%BC%EC%A6%88%EB%B2%A0%EB%A6%AC_%ED%8C%8C%EC%9D%B4

4) 안드로이드 애플리케이션 만들기

- 안드로이드 앱을 만드는 법 : 안드로이드 앱을 만드는 방법을 알아보기 위해 안드로이드 개발 문서(https://developer.android.com/docs?hl=ko)를 찾아봅니다.

시작하기의 첫 앱 빌드 문서에서 안드로이드 프로젝트 만들기에 들어갑니다. 안드로이드 스튜디오 설치 링크를 들어가서 안드로이드 스튜디오를 설치합니다. 새로운 프로젝트를 생성하여 프로젝트 이름과 사용 언어를 설정합니다. 프로젝트를 생성하면 MainActivity.java 파일이 자동적으로 생성됩니다. onCreate 메소드에서 setContentView를 통해 안드로이드 앱의 메인 화면을 구성합니다. 메인 화면인 /res/layout/activity_main.xml를 다음과 같이 설정합니다.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:text="Hello World"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

6. 기본 자료형

1) 데이터와 연산

- 컴퓨터는 데이터를 가지고 연산을 하는 기계라고 할 수 있습니다. 자바와 같은 컴퓨터 프로그래밍 언어를 배워서 사용하는 것은 컴퓨터에게 데이터를 가지고 우리가 원하는 대로 연산을 시킬 수 있게 하는 것이라고 할 수 있습니다. 그러므로 컴퓨터가 다룰 수 있는 데이터는 무엇이고 어떤 연산을 할 수 있는지 알아보는 것은 중요합니다.

컴퓨터가 다룰 수 있는 데이터는 아주 다양하다. 숫자, 문자열, 영상/소리/기타 등등

컴퓨터가 데이터를 구분하는 것은 데이터마다 처리하는 방식이 다르기 때문이다. 예를 들면, 숫자의 경우 +, -, *, / 또는 미적분에 이르기까지 여러 수학적 작업들을 수행합니다. 문자열의 경우, 길이를 호출하고, 특정 구간을 제거하고, 특정 문자열이 있는지 검사하는 것과 같은 작업들을 수행합니다. 이렇듯 데이터의 종류에 따라 처리할 수 있는 연산의 종류가 달라지기 때문에 데이터의 종류를 잘 분류할 필요가 있습니다.

https://ko.wikipedia.org/wiki/%EC%9E%90%EB%A3%8C%ED%98%95

2) 데이터 타입

- 자바의 데이터 타입 : 앞서 컴퓨터는 다양한 데이터를 다룰 수 있고 데이터의 종류에 따라 수행할 수 있는 연산이 달라진다는 것을 알 수 있었습니다.

Data_and_operation의 이름으로 프로젝트를 생성합니다. 생성된 프로젝트의 루트 경로에 Datatype.java파일을 생성하여 다음과 같이 입력합니다. class의 이름은 파일명과 동일하게 작성하여야 합니다.

Datatype.java

```
public class Datatype{
    public static void main(String[] args) {
        System.out.println(6); // Number
        System.out.println("six"); // String

        System.out.println("6"); // String 6

        System.out.println(6+6); // 12
        System.out.println("6"+"6"); // 66

        System.out.println(6*6); // 36
//      System.out.println("6"*"6");
```

```
        System.out.println("1111".length()); // 4
//      System.out.println(1111.length());
    }
}
```

System.out.println()을 입력 할 때 이클립스에서는 sout이라고 입력한 후 Ctrl + Space를 치면 자동으로 입력해 주는 기능이 있습니다. 자바에서는 // 뒤에 입력한 것은 주석으로 인식하여 실행할 때 마치 없는 것처럼 취급합니다.

- 숫자와 문자 데이터 타입과 연산

숫자 : 자바에서 숫자는 다른 기호와 함께 입력하지 않고 그대로 입력합니다. + 연산자는 덧셈의 연산을 수행합니다.

문자 : 자바에서 문자열은 쌍따옴표(“”) 안에 적습니다. + 연산자는 결합의 연산을 수행합니다. 문자열 간에는 *연산자를 사용할 수 없습니다. length 연산은 문자열의 길이를 반환합니다.

https://docs.oracle.com/javase/7/docs/api/java/lang/String.html

3) 숫자와 연산

- 자바의 숫자 데이터의 다른 연산 : 자바의 숫자 연산에는 +와 * 이외에도 다른 연산들이 있습니다. 이전에 생성했던 Data_and_operation 프로젝트에서 진행하도록 합니다. 지금까지는 프로젝트에서 New - File을 클릭하여 새로운 java 파일을 만들어 왔습니다. 하지만 New 메뉴에서 Class를 통해 더 쉽게 자바 클래스를 생성할 수 있습니다. 위와 같이 이름이 Number인 클래스를 생성합니다. public static void main(String[] args) 옵션을 체크하면 main 메소드도 자동적으로 생성해 줍니다. 자바에는 기본적인 +, -, *, /와 같은 사칙연산자 이외에도 숫자를 연산할 수 있는 도구들이 있습니다. 그 중에서 수학과 관련된 것들을 모아놓은 일종의 캐비닛과 같은 Math라는 클래스가 있습니다. Math 안에 있는 여러 데이터와 명령들 중에서 이번에는 PI 데이터를 불러와서 floor 명령어를 이용해 내림을 해 보고, ceil 명령어를 이용해 올림을 해 봅니다. 생성된 클래스에 다음과 같이 입력합니다.

Number.java

```
public class Number {

    public static void main(String[] args) {
        // Operator
        System.out.println(6 + 2); // 8
        System.out.println(6 - 2); // 4
        System.out.println(6 * 2); // 12
        System.out.println(6 / 2); // 3

        System.out.println(Math.PI); // 3.141592653589793
        System.out.println(Math.floor(Math.PI));
        System.out.println(Math.ceil(Math.PI));
    }
}
```

4) 문자열의 표현

- 문자열을 표현하는 방법 : 문자열은 String으로 쌍따옴표(“”) 안에 작성해 표현합니다.

Data_and_operation 프로젝트에 StringApp 클래스를 생성합니다.

StringApp.java

```
public class StringApp {

    public static void main(String[] args) {

        // Character VS String
        System.out.println("Hello World"); // String
        System.out.println('H'); // Character
        System.out.println("H");

        System.out.println("Hello "
                + "World");

        // new line
        System.out.println("Hello \nWorld");

        // escape
        System.out.println("Hello \"World\"");// Hello "World"
    }

}
```

- 문자열(String)과 문자(Character) : 문자열은 문자의 나열입니다. 문자는 따옴표 안에 입력하여 표현하는데, 한 개의 문자만 표함할 수 있습니다. 문자는 오직 한 개의 문자만 포함될 수 있지만, 문자열은 1개 이상의 문자들도 포함할 수 있습니다.

- 이스케이프 기호 : 이스케이프 기호는 줄바꿈 기호나 쌍따옴표와 같은 특수한 기호를 문자열에 넣기 위해 사용합니다. 이스케이프 기호는 역슬래시(\)로 입력할 때는 키보드 상의 원화 기호로 입력하면 됩니다. 특정 문자와 결합하여 제어문자로 가능하기도 하고, 쌍따옴표와 같은 특수한 문자와 결합해 특수한 문자가 일반 문자라고 나타내 주는 역할을 합니다.

문자열에서 줄바꿈을 하고 싶을 때 문자열 사이에서 바로 줄바꿈을 하면 문법상의 오류가 생기게 됩니다. 그렇다면 줄바꿈을 하기 위해서는 \n을 줄을 바꾸고 싶은 위치에 삽입하면 됩니다. 쌍따옴표를 문자열에 입력하고자 할 때에도 쌍따옴표 앞에 역슬래시를 삽입합니다.

https://nanite.tistory.com/42

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

5) 문자열 다루기

- 문자열 연산 : 문자열을 처리하는 여러 명령들이 있지만, 문자열의 길이를 산출하는 length, 문자열의 특정 문자열을 다른 문자열로 교체하는 replace 명령이 있다.

Data_and_operation 프로젝트에서 StringOperation 클래스를 생성합니다.

StringOperation.java

```
public class StringOperation {

    public static void main(String[] args) {

        System.out.println("Hello World".length()); // 11
        System.out.println("Hello, [[[name]]] ... bye. ".replace("[[[name]]]", "duru"));

    }
```

- length와 replace : length는 문자열의 길이를 산출합니다. replace는 두 개의 인자 oldChar, newChar를 받습니다. oldChar에는 기존의 문자열에서 바꾸고 싶은 부분을 입력하고 newChar에는 바꾸고자 하는 문자열을 입력합니다.

"Hello, [[[name]]] ... bye. ".replace("[[[name]]]", "duru")

즉, 이 경우 기존의 "Hello, [[[name]]] ... bye" 문자열에서 "[[[name]]]" 부분을 "duru" 문자열로 치환합니다. 이렇게 문자열에서는 문자열의 길이를 알아내거나, 문자열의 일부를 조작하는 등의 연산을 수행할 수 있습니다.

https://docs.oracle.com/javase/7/docs/api/java/lang/String.html

7. 작심삼일공학

1) 작심삼일공학

- 작심삼일이 되었을 때 어떻게 해야 할까? 작심삼일이 되는 것은 무엇일까요. 뇌에서 이제 공부를 하지 말라는 명령이 아닐까요? 그렇다면 과감히 잠시 공부를 중단해 보세요. 그리고 배운 것들을 곰곰히 되새겨 보면서 내 삶의 문제들과 연관지어 보고, 실제로 한번 해결하기 위해 노력해 봅니다. 이 작업을 통해 우리가 배우고 있는 것들이 우리의 삶에 쓸모있다는 것을 뇌에게 증명할 수 있게 됩니다. 이 과정을 반복하면 점점 뇌의 흥분도가 올라가고, 뇌에서는 이렇게 명령을 내리게 됩니다.

"더 공부해도 좋다"

그리고 나서 공부를 하게 되면 지루하지 않고 고통스럽지 않게 됩니다.

- 강의자 입장에서 작심삼일에 대한 견해 : 우선 강의를 다 마치지 못한 경우 "아무것도 할 수 없는 무능한 상태"가 됩니다. 한편 강의를 다 마친 경우 "앞으로의 목표를 찾지 못하고 무기력한 상태"가 됩니다. 그래서 강의자의 입장에서, 작심삼일 수강생을 위해서 지금까지 배운 것만으로 삶의 문제를 해결할 수 있도록 하고, 또 더 많은 공부를 하라는 뇌의 명령이 떨어지면 다시 공부를 시작할 수 있게 하는 강의를 하도록 노력하고 있습니다.

"언제든지 공부를 그만하고 일을 하도록 합시다. 그리고 언제든지 일을 그만하고 공부를 합시다."

마지막으로 지금까지 우리는 자바라는 것에 대해 0.1%도 안되는 분량의 공부를 했습니다. 그렇지만 지구상에서 자바로 숫자도 계산하고 문자열도 조작하는 일을 할 수 있기까지 45억년이라는 시간이 걸렸습니다. 우리는 지구 역사상 가장 혁명적인 기술을 접한 것이라고 할 수 있습니다. 앞으로 지금까지 배운 자바의 지식도 포함하여 우리는 갖고 있는 모든 지식을 총동원하여 문제를 해결 할 수 있을 것입니다. 그것만으로 충분히 일을 잘 하는 것입니다. 하지만, 일과 공부를 할 때를 주체적으로 판단하여 자신이 할 일을 실행할 수 있는 사람이라면 완

벽할 것입니다. 여러분도 그런 사람이 되기를 바랍니다.

8. 변수

1) 변수의 정의

- 자바에서 변수 만들기 : 수학에서 변수는 수식에 따라 변하는 값입니다. 컴퓨터 프로그래밍 언어에서도 변수는 이것과 사뭇 다른 의미입니다.

Data_and_operation 프로젝트에서 Variable 클래스를 생성합니다.

Variable.java

```
public class Variable {

    public static void main(String[] args) {

        int a = 1; // Number -> integer  ... -2, -1 , 0, 1, 2 ...
        System.out.println(a);

        double b = 1.1; // real number -> double ... -2.0, -1.0, 0, 1.0, 2.0 ...
        System.out.println(b);

        String c = "Hello World";
        System.out.println(c);
    }

}
```

- 숫자와 문자열 변수 지정하기 : 자바에서는 변수를 지정할 때 변수의 데이터 타입을 먼저 선언합니다. 지금까지 숫자 데이터를 세부적으로 다루지 않았지만, 숫자 데이터는 세부적으로 종류가 나뉘어 있습니다.

정수 데이터는 Integer, 즉 int 데이터 타입입니다.

```
int a = 1;
```

그래서 자바에서 정수 데이터를 변수로 만들경우 앞에 int 데이터 타입이라고 선언합니다.

1.5와 같이 실수 데이터의 경우에는 double이라고 합니다.

```
double b = 1.1;
```

실수 데이터를 변수에 담기 위해서는 double이라고 먼저 선언합니다.문자열 데이터는 String 데이터 타입이기 때문에 String이라고 선언하면 됩니다.

```
String c = "Hello World";
```

- 데이터 타입을 지정하는 이유 : 변수의 데이터 타입을 바로바로 판단할 수 있기 때문에 자바에서는 변수를 만들 때 데이터 타입을 지정합니다.

비유적으로 빈 물컵에 담겨있는 보라색 액체가 있다고 하면, 이 액체가 무엇인지 판단하기 위해서는 냄새도 맡아보고 맛을 보는 등 여러 과정을 거쳐야 합니다. 하지만 개봉하지 않은 포도주스 병에 담겨있는 보라색 액체는 어떨까요? 이것이 포도주스라고 바로 판단할 수 있을 것입니다.

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html
https://docs.oracle.com/javase/tutorial/java/nutsandbolts/variables.html

2) 변수의 효용

- 변수 지정의 효용

Data_and_operation 프로젝트에 Letter 클래스를 만듭니다.

Letter.java

```
public class Letter {

    public static void main(String[] args) {
        String name = "leezche";
        System.out.println("Hello, "+name+" ... "+name+" ... egoing ... bye");

        double VAT = 10.0;
        System.out.println(VAT);
    }

}
```

- 변수의 재사용 : 어떤 데이터는 한두 번 사용되지만, 수백 번 이상 훨씬 많이 사용될 수 있습니다. 이럴 경우 변수로 만들어서 만들면 편리하게 사용할 수 있습니다.

```
String name = "leezche";
System.out.println("Hello, "+name+" ... "+name+" ... egoing ... bye");
```

이렇게 name이라는 변수를 지정하면, 자주 사용하는 데이터를 여러번 재사용할 수 있고, 만약 이름을 수정할 경우, 변수의 값만 수정하면 되기 때문에 효율적으로 처리할 수 있습니다.

- 코드의 가독성 : 변수의 이름을 잘 지으면 코드의 의미를 파악할 때 도움이 됩니다.

```
double VAT = 10.0;
System.out.println(VAT);
```

이와 같이 VAT(부가가치세)라는 이름을 붙이게 되면, 단순히 실수 10이 아닌 부가가치세라는 의미를 지니게 됩니다. 위의 name 변수도 마찬가지로 단순한 문자열 "leezche"가 아닌 이름이라는 의미를 지니게 됩니다. 변수를 사용하면 코드의 의미를 쉽게 파악할 수 있게 됩니다. 따라서 미래의 자신 혹은 프로젝트 동료에게도 코드가 쉽게 읽힐 수 있게 됩니다.

코드의 가독성과 재사용을 위해서 여러 방법이 쓰일 수 있지만, 변수를 이용하는 것이 그 중에서도 중요한 수단 중 하나입니다.

3) 데이터 타입의 변환 - casting

- 데이터 타입 변환 : 데이터 타입을 정수에서 실수로 변환하는 과정이 필요합니다.

Data_and_operation 프로젝트에서 Casting 클래스를 생성합니다.

Casting.java

```
public class Casting {

    public static void main(String[] args) {

        double a = 1.1;
```

```
        double b = 1;
        double b2 = (double) 1;

        System.out.println(b);

        // int c = 1.1;
        double d = 1.1;
        int e = (int) 1.1;
        System.out.println(e);

        // 1 to String
        String f = Integer.toString(1);
        System.out.println(f.getClass());


    }

}
```

- 정수와 실수 간에 변환하기 : 정수에서 실수로 변환하기 위해서 명시적으로 데이터 타입 변환을 나타낼 필요가 없습니다.

`double b = 1;`

이렇게 변수의 데이터 타입을 실수로 지정하는 것만으로도 변환이 가능합니다. 하지만, 반대로 실수에서 정수로 변환할 경우에는 그렇지 않습니다.

`int e = (int) 1.1;`

실수 1.1을 정수로 변환하기 위해서는 데이터 타입을 명시적으로 변경해 주어야 합니다.

- 정수를 문자열로 변환하기

`String f = Integer.toString(1);`

이렇게 Integer 객체의 toString 메소드를 이용하면 숫자를 문자열로 변환할 수 있습니다.

https://www.w3schools.com/java/java_type_casting.asp

9. 프로그래밍

1) 프로그래밍이란 무엇인가

- 프로그램이란 : 음악회와 같은 곳에서 연주할 곡들의 순서를 나타내는 것이었습니다. 요즘도 연주회에서 음악의 진행 순서를 알려주는 안내문에 프로그램이라고 쓰여 있는 것을 볼 수 있습니다. 컴퓨터에서 프로그램도 마찬가지로 시간에 따라 실행할 작업들의 순서를 의미합니다. 이렇게 시간에 따라 컴퓨터가 정해진 순서에 작업을 실행하게 된다면 우리는 업무를 자동적으로 처리할 수 있게 됩니다. 컴퓨터 언어를 이용해서 프로그램을 만드는 것은 업무의 자동화된 처리를 위해서라고 할 수 있습니다.
- 컴퓨터 프로그래밍의 의미 : 기존의 작업 공간에서 Programming 프로젝트를 새로 생성합니다. 그리고 그 안에 Program 클래스를 생성합니다.

Program.java

```
public class Program {

    public static void main(String[] args) {

        System.out.println(1);
        System.out.println(2);
        System.out.println(3);

    }

}
```

이 작업은 그저 화면에 1, 2, 3을 순서대로 출력하는 작업을 수행합니다. 만약 굉장히 오래 걸리는 작업이라 하염없이 붙잡고 있어야 하는 작업이라면 컴퓨터 프로그래밍을 이용해 사람이 잘 못하는 일을 기계에게 위임해서 자동화할 수 있습니다. 그리고 이러한 프로그래밍을 가능하게 해주는 언어 중 가장 유명한 것이 자바입니다.

2) 프로그램 만들기 - IoT 라이브러리 설치하기

- IoT 모의 프로그램 만들기 : 프로그램으로 할 수 있는 작업 자동화를 체감하기 위해서 모의 프로그램을 만들어 보도록 하겠습니다. 실제로 우리 주변의 사물을 움직이게 하는 프로그램은 아니지만, 프로그램으로 이런 것도 할 수 있다는 점을 느낄 수 있도록 구성하였습니다.
- 패키지 설치하기 : 이번 프로그램을 제작하기 위해서 외부 패키지를 이용해 보도록 하겠습니다.

https://github.com/egoing/java-iot
https://gitlab.com/egoing/java-iot
https://bitbucket.org/egoing/java-iot/src/master/

이 중 하나의 링크를 통해 org.opentutorials.iot 패키지를 다운받습니다. 이제 Programming 프로젝트에 org.opentutorials.iot 패키지를 설치합니다. 다운받은 패키지를 압축을 푼 후 이클립스 Navigator 뷰에서 org 폴더를 Programming 프로젝트에 드래그 앤 드랍합니다. (Package Explorer 뷰에서도 가능합니다.) Copy files and folders 옵션을 선택하여 OK버튼을 클릭합니다.

2) IoT 프로그램 만들기

- 프로그램 만들기 : Programming 프로젝트에서 OkJavaGoInHome 클래스를 생성합니다. 집에 갈때 반복적으로 일어났던 작업을 자동화하는 프로그램을 만들어 봅시다. 우선 집에 갈 때 엘리베이터를 1층으로 부르고, 현관문 보안을 해제하고, 집에 들어온 다음에는 불이 좀 켜져 있으면 좋겠습니다. 이런 기능들을 이전 시간에 설치했던 가짜 IoT 패키지를 이용해서 만들어 보도록 하겠습니다.

OkJavaGoInHome.java

```
import org.opentutorials.iot.Elevator;
import org.opentutorials.iot.Lighting;
import org.opentutorials.iot.Security;
```

```
public class OkJavaGoInHome {

    public static void main(String[] args) {

        String id = "JAVA APT 507";

        // Elevator call
        Elevator myElevator = new Elevator(id);
        myElevator.callForUp(1);

        // Security off
        Security mySecurity = new Security(id);
        mySecurity.off();

        // Light on
        Lighting hallLamp = new Lighting(id+" / Hall Lamp");
        hallLamp.on();

        Lighting floorLamp = new Lighting(id+" / floorLamp");
        floorLamp.on();

    }

}
```

import 구문을 이용하면 기존에 org.opentutorials.Elevator과 같이 입력해야 하는 구문을 Elevator로 단순하게 입력할 수 있습니다. "JAVA APT 507"과 같이 반복적으로 사용되는 값의 경우는 변수로 지정하여 재사용할 수 있고 의미있는 이름을 지정하여 코드의 가독성을 높일 수 있습니다.

10. 디버거

1) 디버거

- 이클립스에서 디버거 이용하기 : 이클립스 상단의 메뉴 도구 중 벌레 모양으로 생긴 버튼을 클릭하면 디버거를 실행할 수 있습니다. 창이 뜨면, Switch 버튼을 누르면 Debug perspective로 화면 구성을 바꿀 수 있습니다. 또는 이클립스 우상단의 벌레 모양을 클릭해도 디버그 화면으로 전환할 수 있습니다.
- 브레이크 포인트 지정하기 : 코드 편집 창에서 줄 번호 왼편에서 더블클릭하면 9번 줄에서와 같이 브레이크 포인트가 지정됩니다. 브레이크 포인트를 지정한 상태에서 디버거를 실행하게 되면 브레이크 포인트까지 코드가 실행되고 그 이후로 실행이 일시정지 됩니다. 브레이크 포인트를 다시 클릭하면 브레이크 포인트를 지울 수 있습니다.

Step Over 버튼을 클릭하면, 다음 줄에 브레이크 포인트가 생성되어 그 지점까지만 코드가

실행됩니다. Resume 버튼을 클릭하면 다음 브레이크 포인트까지 실행되고, 만약 더 이상 브레이크 포인트가 없다면 끝까지 실행됩니다. 우측의 Variable 탭에서 변수들을 확인할 수 있습니다.

- 자세한 실행 과정 들여다보기 : Step Into 버튼을 클릭하면 코드의 자세한 실행 과정을 들여다볼 수 있습니다. hallLamp 변수의 on 메소드가 실행되는 줄에서 Step Into 버튼을 누르면, Lighting 객체의 on 메소드의 코드가 실행되는 과정을 확인할 수 있습니다. 다시 원래의 코드로 돌아가고자 할 경우에는 Step Return 버튼을 클릭하여 돌아갑니다.

11. 입력과 출력

1) 입력과 출력

- 프로그램의 입력과 출력 : 프로그램은 입력정보를 받아서 출력을 하는 것이라고 할 수 있습니다. 입력 정보는 문자열, 숫자 등의 아규먼트가 될 수도 있고, 파일, 네트워크를 통해 받은 정보, 소리, 다른 프로그램에서 출력된 정보 등이 될 수도 있습니다. 출력 정보도 화면에 출력하는 형태가 될 수 있고, 파일에 쓸수도 있고, 소리로 내보낼 수도 있습니다.

OkJavaGoInHome을 살짝 변형하여 입력을 받을 수 있도록 바꾸어 봅니다.

- 프로그램이 입력을 받을 수 있게 하기 : 만약 JAVA 아파트에서 Pusan 아파트로 이사를 갔다고 한다면, OkJavaGoInHome은 Pusan 아파트에서 같은 동작을 하게 할 수 없을 것입니다. 변수 id가 사용자로부터 입력을 받아서 프로그램을 실행할 때마다 다른 값을 가질 수 있게 한다면, Pusan 아파트에서도 같은 동작을 할 수 있지 않을까요? OkJavaGoInHomeInput 객체를 새로 만들어서 아래와 같이 입력합니다.

OkJavaGoInHomeInput.java

```
import javax.swing.JOptionPane;

import org.opentutorials.iot.DimmingLights;
import org.opentutorials.iot.Elevator;
import org.opentutorials.iot.Lighting;
import org.opentutorials.iot.Security;

public class OkJavaGoInHomeInput {

    public static void main(String[] args) {

        String id = JOptionPane.showInputDialog("Enter a ID");
        String bright = JOptionPane.showInputDialog("Enter a Bright level");

        // Elevator call
        Elevator myElevator = new Elevator(id);
        myElevator.callForUp(1);

        // Security off
```

```
        Security mySecurity = new Security(id);
        mySecurity.off();

        // Light on
        Lighting hallLamp = new Lighting(id+" / Hall Lamp");
        hallLamp.on();

        Lighting floorLamp = new Lighting(id+" / floorLamp");
        floorLamp.on();

        DimmingLights moodLamp = new DimmingLights(id+" moodLamp");
        moodLamp.setBright(Double.parseDouble(bright));
        moodLamp.on();

    }

}
```

JOptionPane 객체의 showInputDialog 메소드를 이용하면 입력 다이어로그 창을 이용해서 id 값을 입력할 수 있습니다. JOptionPane 객체를 이용하기 위해서는 import 구문을 이용하여 불러옵니다.

```
import javax.swing.JOptionPane;
```

JOptionPane의 showInputDialog를 이용해서 입력한 정보는 String 데이터로 받아들여집니다.

```
String id = JOptionPane.showInputDialog("Enter a ID");
```

- 무드 램프 밝기 입력받기 : 무드 램프(DimmingLight 객체)는 setBright 메소드에 double 데이터를 입력받아서 밝기를 조절합니다. DimmingLight 객체를 생성하여 밝기를 10으로 조절해 봅니다.

```
DimmingLights moodLamp = new DimmingLights(id+" moodLamp");
moodLamp.setBright(10);
moodLamp.on();
```

만약 집에 들어올 때마다 밝기를 다르게 조정하고 싶다면, 밝기 값도 입력을 받아서 조절할 수 있을 것입니다.

```
String id = JOptionPane.showInputDialog("Enter a ID");
String bright = JOptionPane.showInputDialog("Enter a Bright level");
```

한편, setBright 메소드의 밝기는 double 데이터로 입력해야 하기 때문에 데이터 타입 변환을 해야 합니다. 문자열을 double로 데이터 타입을 변환하기 위해서 다음과 같이 수정합니다.

```
moodLamp.setBright(Double.parseDouble(bright));
```

https://docs.oracle.com/javase/7/docs/api/javax/swing/JOptionPane.html

https://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html

2) arguments & parameter

- 아규먼트를 입력받아 프로그램 실행시키기 : main 메소드의 args 파라미터를 이용해서 입

력 값을 받는 방법이 있습니다. Run 버튼의 팝업 버튼을 클릭하여 Run Configurations 메뉴를 클릭합니다. Argument 탭에서 Program arguments에 위와 같이 입력합니다. 만약 작은따옴표로 동작하지 않는다면 큰따옴표로 시도합니다. 그리고 id와 bright 부분을 다음과 같이 수정합니다.

String id = args[0];

Strign bright = args[1];

아규먼트를 입력하게 되면 main 메소드의 args 파라미터는 아규먼트 값을 받아서 동작하게 됩니다. args는 문자열 배열(array)로 여러 개의 String 데이터가 들어있을 수 있습니다. 인덱스를 통해 배열의 데이터를 꺼내 쓸 수 있고, 인덱스는 0번부터 시작합니다.

https://stackoverflow.com/questions/156767/whats-the-difference-between-an-argument-and-a-parameter

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html

12. 직접 컴파일 실행

1) 소개

- 이클립스 없이 혼자 컴파일하기 : 이클립스는 자바로 프로그램을 만들기 위해 유용한 도구입니다. 하지만 이클립스 없이도 자바로 프로그램을 만들기 위해서는 어떻게 해야 할까요? 이것이 가능하다면 이클립스가 없는 어떤 환경이라도 자바로 프로그래밍을 할 수 있게 될 것입니다. 이클립스 없이 자바로 프로그래밍을 하려면 우선 자바 파일을 스스로 컴파일할 수 있어야 합니다. 그리고 컴파일한 클래스 파일을 실행하는 과정을 거쳐야 합니다. 이 과정은 운영체제에 대한 많은 지식이 필요할 수 있습니다.

2) 실행환경 살펴보기

- 터미널 열기

Windows : 윈도우 키 + R로 실행 창을 키고, cmd를 입력하여 명령 프롬프트를 실행합니다.

MacOS : 스포트라이트에서 terminal을 입력해서 터미널을 실행합니다.

javac 확인 : 터미널에 javac를 입력하고 실행하면 javac 명령어의 사용 방법이 출력됩니다.

javac는 어디에 있는 명령어일까요?

Windows : 자바의 설치 경로(ex: C:\Program files\java\jdk-14.0.1)\bin\javac.exe

MacOS : 우선 pwd(Print Working Directory) 명령어로 현재 경로를 확인합니다. /usr/libexec/java_home를 입력하여 자바의 설치 경로를 확인합니다. 그러면 터미널에 자바의 설치 경로가 출력됩니다. 실행되지 않는다면 그냥 넘어갑니다. cd(Change Directory) 명령어로 자바의 설치 경로로 이동합니다. cd /Library/Java/JavaVirtualMachines/jdk-9.0.4.jdk/Contents/Home pwd 명령어로 이동이 되었는지 확인할 수 있습니다. ls(list segments) 명령어로 현재 경로에 있는 파일들을 목록을 확인합니다. 그 중 bin 폴더로 이동합니다. cd bin ls 명령어로 파일 목록을 확인하면 java와 javac가 있는 것을 확인할 수 있습니다. javac 명령어는 여기에서의 javac 파일이 실행되는 것입니다.

Linux :

```
readlink -f $(which java)       // java 경로 확인
readlink -f $(which javac)      // javac 경로 확인
```

- 환경 변수 경로 확인 : 그렇다면 javac의 경로를 운영체제에서 어떻게 알고 javac만 입력해도 실행할 수 있는 것일까요? 환경변수에 이 경로가 이미 입력되어 있기 때문입니다. 각 운영체제에서 환경변수를 설정하는 법을 알아봅니다.

Windows : 윈도우 탐색기에서 내 PC 항목을 오른쪽 버튼으로 클릭하여 나오는 팝업창에 속성을 클릭합니다. 고급 시스템 설정을 클릭합니다. 환경 변수를 클릭하면 아래와 같이 환경변수 목록을 확인할 수 있습니다. Path를 더블클릭하면 아래와 같이 수정할 수 있습니다. 이렇게 환경 변수의 Path에 javac의 경로(자바 설치 경로\bin)가 입력되어 있기 때문에 명령 프롬프트에서 javac 명령어를 바로 이용할 수 있습니다. 운영체제는 javac 명령어를 입력받으면 Path에 지정되어 있는 경로들 안에 javac 명령어가 있는지 확인하여 실행합니다.

MacOS / Linux : echo $PATH를 입력하면 환경 변수의 Path 정보를 확인할 수 있습니다. 만약 javac의 경로가 포함되어 있지 않다면 nano ~/.bash_profile 명령어로 추가할 수 있습니다. 파일의 편집기가 열리면 마지막에 아래처럼 입력하여 java의 경로를 입력합니다.

export PATH=$PATH:/Library/Java/JavaVirtualMachines/jdk-9.0.4.jdk/Contents/Home/bin

https://java.com/ko/download/help/path.html

3) 컴파일과 실행하기

- 프로젝트 디렉토리로 이동하기

Windows : 명령 프롬프트에서 cd 명령어를 통해 프로젝트 디렉토리로 이동합니다. cd C:\Users\egoing\Desktop\java1\Programming dir 명령어로 프로젝트 디렉토리 내부의 구조를 확인할 수 있습니다.

MacOS(Linux) : 파인더에서 프로젝트 디렉토리에 들어가면 하단에 경로를 복사할 수 있는 기능이 있습니다. 윈도우와 마찬가지로 터미널에서 cd 명령어를 이용하여 프로젝트 디렉토리로 이동합니다. cd /Users/live/Desktop/java1/Programming ls 명령어로 프로젝트 디렉토리 내부의 구조를 확인할 수 있습니다.

- 자바 파일 컴파일하기 : 터미널에 javac를 입력하면 javac 명령어의 사용 방법을 보여줍니다. Program.java를 컴파일합니다. javac Program.java 그리고 나서 윈도우에서는 dir, macOS나 Linux에서는 ls 명령어를 입력하여 내부 구조를 살펴보면, Program.class가 생성된 것을 확인할 수 있습니다. (기존에 있던 class 파일을 지우고 실행해 봅니다.) 혹시 에러가 난다면 다음과 같이 실행해 봅니다. javac -cp "." Program.java

- 실행하기 : 컴파일된 클래스 파일을 실행하기 위해서는 java 명령어를 이용합니다. java 명령어로 컴파일된 파일을 가상 머신에서 실행하는 작업을 합니다. 실행할 때는 .class를 붙이지 않습니다. java Program 만약 에러가 난다면 다음과 같이 입력합니다. java -cp "." Program

4) 라이브러리 이용하기

- 라이브러리를 이용하는 프로그램을 컴파일하기

OkJavaGoInHome.java

```
import org.opentutorials.iot.Elevator;
import org.opentutorials.iot.Lighting;
import org.opentutorials.iot.Security;

public class OkJavaGoInHome {
```

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

        ....
    }

}
```

import 구문을 통해 OkJavaGoInHome 클래스가 있는 폴더의 org.opentutorials.iot의 클래스들을 불러들이고 있기 때문에 컴파일을 실행할 때 자동적으로 org.opentutorials.iot의 Elevator, Lighting, Security 클래스들도 컴파일을 합니다. 이때 lib 폴더를 새로 만들어서 org 폴더를 그 안으로 이동시킵니다. 이제 org.opentutorials.iot 패키지는 lib 라이브러리의 일부가 되었습니다. 다시 OkJavaGoInHome를 컴파일한다면 잘 될까요?

- 외부 라이브러리도 포함해서 컴파일하기 : 외부 라이브러리를 포함해서 컴파일하기 위해서는 javasc 명령어의 옵션 중 --class-path(-cp) 옵션을 이용해서 외부 라이브러리도 함께 지정해야 합니다.

```
javac -cp ".;lib" OkJavaGoInHome.java
```

macOS나 Linux의 경우에는 ".;lib" 부분의 세미콜론(;)을 콜론(:)으로 바꿔서 입력합니다. 콜론(:)이나 세미콜론(;)은 구분자의 의미를 갖습니다.

```
javac -cp ".:lib" OkJavaGoInHome.java
```

--class-path 옵션은 -cp로 줄여서 표현할 수 있습니다. -cp ".:lib"은 자바 파일이 있는 현재 폴더(.)와 lib 폴더에서 필요한 자바 파일들을 컴파일하라는 의미입니다.

- 외부 라이브러리도 포함해서 실행하기 : 외부 라이브러리의 클래스들도 함께 사용하는 프로그램을 실행하기 위해서는 컴파일했을 때와 마찬가지로 --class-path 옵션에 외부 라이브러리도 포함해서 실행해야 합니다.

윈도우에서는 아래와 같이 입력합니다.

```
java -cp ".;lib" OkJavaGoInHome
```

macOS와 Linux에서는 세미콜론이 아닌 콜론으로 바꿔서 입력합니다.

```
java -cp ".:lib" OkJavaGoInHome
```

5) 입력과 출력

- 실행할 때 아규먼트 입력하기 : org.opentutorials.iot 패키지는 다시 lib 폴더 밖으로 꺼냅니다.

OkJavaGoInHomeInput.java를 컴파일합니다.

```
javac OkJavaGoInHomeInput.java
 OkJavaGoInHome.java
import org.opentutorials.iot.DimmingLights;
import org.opentutorials.iot.Elevator;
import org.opentutorials.iot.Lighting;
import org.opentutorials.iot.Security;

public class OkJavaGoInHomeInput {
```

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

        String id = args[0];
        String bright = args[1];

        ....

    }

}
```

id 변수와 bright 변수는 args 파라미터를 이용해서 값을 받고 있습니다. 이를 실행하기 위해서 이클립스에서는 Run Configuration의 Argument 탭에서 입력을 해서 실행했었습니다. 터미널에서 아규먼트를 주기 위해서는 어떻게 해야 할까요? 실행할 클래스 파일 이름 다음에 연달아서 입력합니다.

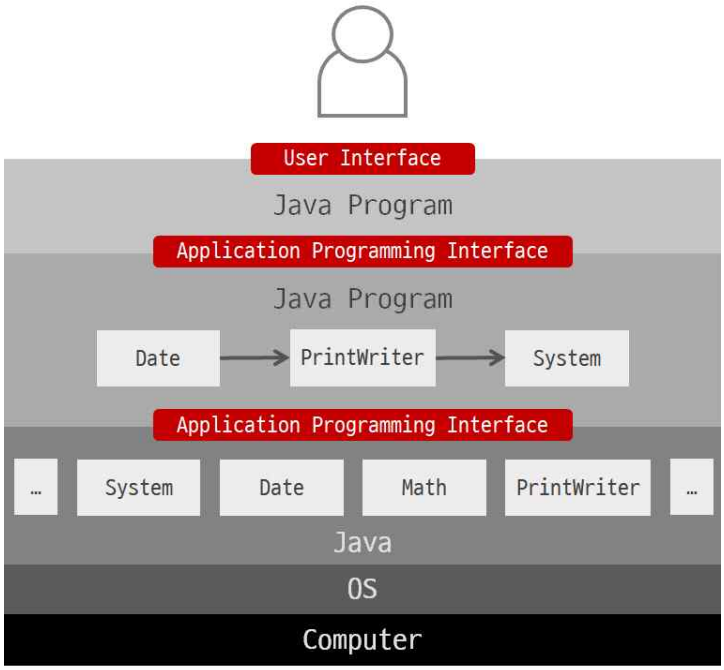
java OkJavaGoInHomeInput "JAVA APT 507" 15.0

(2번째 인수인 15.0은 큰따옴표 안에 넣어도 되고 소수점은 생략할 수 있습니다.)

13. 자바 문서 보는 법

1) API VS UI

- API와 UI



우리가 자바 프로그램을 만들고 사용할 때의 단계이다. 컴퓨터와 운영체제 위에 우리는 자바를 설치했습니다. 자바의 라이브러리에는 다양한 도구들이 있는데, 모니터에 출력했을 때 사용했던 System 객체를 비롯하여 Date, Math, PrintWriter 등 다양한 도구들이 있습니다. 우리는 자바 프로그램을 만들 때 자바의 도구들을 응용해서 우리가 원하는 작업을 시간적 순서

에 따라 동작하도록 만들었습니다. 작업들의 시간적 순서에 주목해서 우리는 프로그램(Program)이라고 부르고, 도구의 응용에 주목해서 우리는 애플리케이션(Application)이라고 부릅니다. 이 때 자바는 자바의 도구들을 응용해서 사용하기 위해서 일정한 조작 장치를 구성하였는데, 이것을 자바 API(Application Programming Interface)라고 합니다. 자바 프로그램은 또 다른 자바 프로그램에서 사용될 수도 있고, 다른 프로그램에서 사용할 수 있도록 만들어둔 장치 역시 API입니다. 이렇게 만들어진 자바 프로그램은 사용자가 사용할 수 있습니다. 이 때 사용자가 자바 프로그램을 사용할 수 있도록 만들어둔 장치들을 UI(User Interface)라고 합니다. 예를 들면, 커맨드 라인 시스템의 아규먼트, 데스크톱 앱의 버튼, 웹 페이지의 링크 등이 있습니다.

https://dydrlaks.medium.com/api-%EB%9E%80-c0fd6222d34c

https://docs.oracle.com/javase/8/docs/api/

2) 패키지, 클래스, 변수, 메소드

- Java API Documentation 보는 방법 : 자바 프로그램을 만들기 위해서는 자바의 도구들을 사용할 수 있게 만든 장치인 자바 API를 이용합니다. 오라클에서는 자바 API의 설명서인 Java API documentation을 제공하고 있습니다.

포털 검색을 통해 자바 API Documentation 페이지를 열어 봅니다. 왼쪽의 위쪽 섹션에는 패키지들에 대한 정보를 담고 있고, 왼쪽의 아래 섹션에는 클래스에 대한 정보를 담고 있습니다. 왼쪽의 아래 섹션에서 우리가 전에 사용해 보았던 Math 클래스를 클릭해서 정보를 확인해 봅니다. Math 클래스의 설명서를 보면 위와 같은 형식으로 구성되어 있습니다. Math 클래스는 java.lang 패키지에 속해 있다는 것을 알 수 있습니다. java.lang 패키지에는 Math 외에 String과 같은 다른 객체들도 존재합니다. 더 스크롤해서 보면 PI와 같은 변수(Variable/Field)들에 대해서도 나와 있고, 전에 사용했던 floor, ceil 등 메소드(Method)들의 정보들도 나와 있습니다.

- 클래스, 변수, 메소드, 패키지 : Math 클래스와 같이 클래스 안에는 PI와 같은 변수, floor, ceil과 같은 메소드들이 포함되어 있습니다. 패키지는 이러한 클래스(들)을 하나의 묶음으로 정리한 것입니다.

https://docs.oracle.com/javase/tutorial/java/concepts/class.html

https://docs.oracle.com/javase/tutorial/java/concepts/package.html

3) 클래스

- 클래스

Programming 프로젝트에서 ClassApp 클래스를 생성합니다.

ClassApp.java

```
public class ClassApp {

    public static void main(String[] args) {

        System.out.println(Math.PI);
        System.out.println(Math.floor(1.6));
        System.out.println(Math.ceil(1.6));

    }
```

}

Math 클래스에는 수학과 관련된 여러 변수들과 메소드들이 있습니다. 우선 PI 변수는 원주율이 적절한 정밀도로 저장되어 있는 변수입니다. floor 메소드는 특정 소수점 이하에 대해서 버림한 값을 산출합니다. ceil 메소드는 특정 소수점 이하에 대해서 올림한 값을 산출합니다.

https://docs.oracle.com/javase/tutorial/java/concepts/class.html

4) 인스턴스

- 인스턴스 : 인스턴스는 클래스를 컴퓨터 상에서 실체화한 것입니다.

Programming 프로젝트에서 InstanceApp 클래스를 생성합니다.

InstanceApp.java

```
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.PrintWriter;
public class InstanceApp {

    public static void main(String[] args) throws IOException{

        PrintWriter p1 = new PrintWriter("result1.txt");
        p1.write("Hello 1");
        p1.close();

        PrintWriter p2 = new PrintWriter("result2.txt");
        p2.write("Hello 2");
        p2.close();

    }

}
```

Math 클래스와 달리 PrintWriter 객체는 new 키워드를 통해 인스턴스를 생성하여 사용합니다.

PrintWriter p1 = new PrintWriter("result1.txt");

result1.txt 파일에 쓰기가 가능한 PrintWriter 클래스를 실제로 생성해서 이름을 p1으로 붙였습니다. PrintWriter 클래스는 java.io 패키지에 속해 있어서 사용하기 위해서는 import 구문을 이용해서 불러들여야 합니다. PrintWriter 인스턴스를 생성할 때는 사용할 파일이 실제로 존재하지 않는다면 오류를 내기 때문에 이에 대한 처리를 해야 합니다. Add throws declaration을 이용해서 에러 핸들링을 할 수 있습니다.

- 인스턴스를 만드는 이유 : 인스턴스는 객체를 다양한 상태에서 사용하고, 기능을 재사용할 경우가 많은 상황에서 유용한 방식입니다. 인스턴스를 생성할 수 있는 클래스는 자바 API Documentation에 Constructor Summary가 존재합니다.

https://gmlwjd9405.github.io/2018/09/17/class-object-instance.html

https://docs.oracle.com/javase/tutorial/java/concepts/class.html

5) 상속

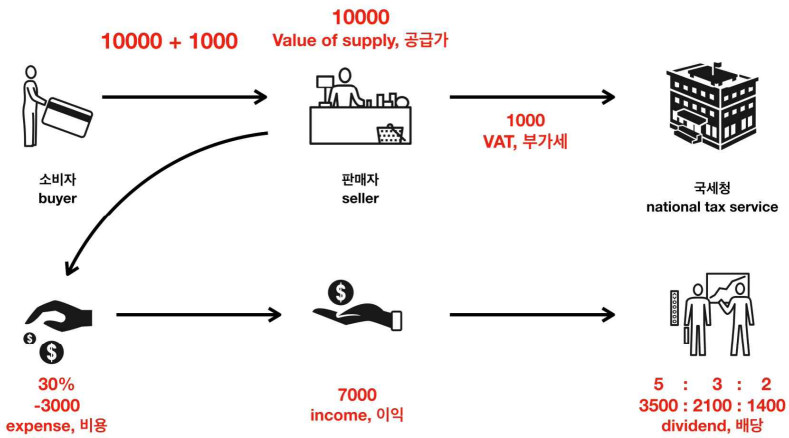
- 클래스의 상속관계 : 클래스 간에는 서로 계층적인 관계를 갖고 있을 수 있습니다. 들여쓰기되어 표현된 각각의 클래스 간의 관계는 상속 관계를 나타냅니다. PrintWriter 클래스는 Writer 클래스에서 상속을 받았고, Writer 클래스는 Object 클래스로부터 상속을 받았다는 것을 나타냅니다.
- 클래스 간의 상속관계의 특성 : 자식에 해당하는 클래스는 부모에 해당하는 클래스의 모든 변수와 메소드를 사용할 수 있습니다. 이클립스 안에서 Open Type hierarchy를 이용하여 클래스의 상속관계를 확인할 수 있습니다. PrintWriter는 Writer를 확장해서 만들어진 클래스이고, Writer는 Object를 확장해서 만들어진 클래스입니다. 그래서 PrintWriter 클래스는 Writer의 write 메소드를 사용할 수 있고, Object의 toString 메소드를 사용할 수 있습니다. Object 클래스는 모든 클래스의 부모로 모든 클래스는 Object의 변수와 메소드를 상속받습니다. 자바 API 설명서에도 클래스들 간의 계층적인 구조, 부모 클래스로부터 상속받은 변수와 메소드에 대한 설명이 나와 있습니다.

https://docs.oracle.com/javase/tutorial/java/concepts/inheritance.html

14. 나만의 앱 만들기

1) 오리엔테이션

- 우리의 문제 현상은 무엇인가?



위와 같은 상황에서 돈의 흐름을 빠르게 계산해야 하는 문제가 있다고 합시다. 계산기를 두드리자니 너무 손이 많이가고 프로그램을 이용하자니 대기업을 위한 프로그램이라서 너무 복잡합니다. 우리가 배운 자바를 이용해서 이 문제를 해결해 봅시다.

우선 우리가 해야할 일은 현실을 분석하는 것입니다. 판매자인 우리는 소비자에게 물건을 공급할때 공급가(Value of supply)를 받습니다. 그리고 공급가의 10%만큼 소비자는 부가가치세(Value added tax)를 더 지불하고 판매자는 국세청에 부가가치세를 납부합니다. 이 때 물건을 판매하기까지는 30%의 비용이 들고 나머지 70%가 이익이라고 할 때, 이익은 투자한 동업자들에게 5:3:2의 비율로 배당합니다. 이 문제를 자바로 어떻게 해결할 수 있을까?

2) 기본 기능 구현

- MyApp 만들기 : 문제를 해결하기 위해 새로운 프로젝트에서부터 시작해 봅시다. 새로운

프로젝트 이름을 MyApp으로 해 생성합니다. MyApp 프로젝트에 AccountingApp 클래스를 생성하여 물건을 판매하면서 이루어지는 여러 계산을 알아봅시다.

AccountingApp.java

```
public class AccountingApp {

	public static void main(String[] args) {

		System.out.println("Value of supply : " + 10000.0);
	System.out.println("VAT : " + (10000.0 * 0.1));
	System.out.println("Total : " + (10000.0 + (10000.0 * 0.1)));
	System.out.println("Expense : " + (10000.0 * 0.3));
	System.out.println("Income : " + (10000.0 - (10000.0 * 0.3)));
	System.out.println("Dividend 1 : " + (10000.0 - (10000.0 * 0.3)) * 0.5);
	System.out.println("Dividend 2 : " + (10000.0 - (10000.0 * 0.3)) * 0.3);
	System.out.println("Dividend 3 : " + (10000.0 - (10000.0 * 0.3)) * 0.2);

	}

}
```

공급가를 10000.0으로 잡고 시작합니다. 부가가치세는 10000.0에 10%인 0.1을 곱한 10000.0 * 0.1입니다. 총 판매가는 공급가와 부가가치세를 합한 10000.0 + (10000.0 * 0.1)입니다. 비용은 판매가의 30%이므로 10000.0 * 0.3입니다. 수익은 판매가에 비용을 뺀 값이므로 10000.0 - (10000.0 * 0.3)입니다. 첫 번째 배당자는 수익의 50%를 배당받으므로 (10000.0 - (10000.0 * 0.3)) * 0.5입니다. 두 번째 배당자는 수익의 30%를 배당받으므로 (10000.0 - (10000.0 * 0.3)) * 0.3입니다. 세 번째 배당자는 수익의 20%를 배당받으므로 (10000.0 - (10000.0 * 0.3)) * 0.2입니다. 공급가인 10000.0을 모두 다른 값으로 교체하면 공급가에 따른 여러 값들을 계산할 수 있습니다.

3) 변수 도입

- 변수를 적용하기 : 변수는 코드의 재사용성을 높이고 의미 있는 이름을 높여서 코드의 가독성을 높일 수 있습니다. 그러면 우리 프로그램에도 변수를 도입해서 이러한 장점을 취해보도록 합니다.

AccountingApp.java

```
public class AccountingApp {

    public static void main(String[] args) {

	double valueOfSupply = 10000.0;
	double vatRate = 0.1;
	double expenseRate = 0.3;
	double vat = valueOfSupply * vatRate;
	double total = valueOfSupply + vat;
```

```
        double expense = valueOfSupply * expenseRate;
        double income = valueOfSupply - expense;
        double dividend1 = income * 0.5;
        double dividend2 = income * 0.3;
        double dividend3 = income * 0.2;

        System.out.println("Value of supply : " + valueOfSupply);
        System.out.println("VAT : " + vat);
        System.out.println("Total : " + total);
        System.out.println("Expense : " + expense);
        System.out.println("Income : " + income);
        System.out.println("Dividend 1 : " + dividend1);
        System.out.println("Dividend 2 : " + dividend2);
        System.out.println("Dividend 3 : " + dividend3);

    }

}
```

특정 값을 지역 변수로 바꾸기 위해 Extract Local Variable 기능을 이용할 수 있습니다. 특정 값을 블록으로 지정한 수 Refactor - Extract Local Variable을 클릭합니다. 변수의 이름을 지정하고 OK를 누르면 변수가 생성됩니다. OK 옆의 Preview 버튼을 누르면 변수 생성 전후를 비교할 수 있습니다. 위와 같은 경우 2번째 동업자에게 할당된 배당 비율도 expenseRate로 바뀌게 된다는 것을 확인할 수 있습니다. 배당 비율은 비용 비율과 다르므로 취소합니다. 또는 변수로 지정하고자 하는 값을 지우고 이름을 바로 입력하면 이클립스에서 변수 생성을 도와줍니다. Create local variable 'expenseRate'를 클릭하면 변수를 생성해 줍니다. 이런 식으로 변수를 지정하여 바꾸어 줍니다. 이제 공급가 값만 바꾸어 주면 다른 값들은 실행할 때 자동으로 계산되는 것을 볼 수 있습니다.

4) 입력값 도입

- 아규먼트를 받는 프로그램으로 수정하기 : AccountingApp.java를 아래와 같이 수정합니다.

```
public class AccountingApp {

        public static void main(String[] args) {

                double valueOfSupply = Double.parseDouble(args[0]);

        …

        }
}
```

- 터미널에서 실행하기 : 터미널을 실행하여 MyApp 프로젝트 폴더로 이동합니다(cd:

Change Directory).
 javac AccountingApp.java
javac 명령어로 AccountingApp.java를 컴파일합니다.
 java AccountingApp 33333.0
java 명령어로 AccountingApp을 실행하고 아규먼트로 원하는 공급가액을 입력하여 실행합니다.
 - 만약 실행하고자 하는 컴퓨터에 자바조차 깔려있지 않다면...
lauch4j(http://launch4j.sourceforge.net/)와 같은 솔루션을 이용하면 자바 프로그램을 JRE까지 포함한 실행파일로 변환하여, 자바가 깔려있지 않은 컴퓨터라도 애플리케이션을 실행할 수 있게 합니다.

5) 조건문

 - 조건문 : 제어문은 프로그램의 실행 과정을 조건에 따라 바꾸는 것입니다. 서울 지하철을 예로 들어 보면 같은 길로 가던 1호선 열차도 방향에 따라 구로역에서 인천 방향과 천안 방향 노선이 분기하고 2호선처럼 성수역이 나올 때까지 빙글빙글 돌기도 합니다. 자바에는 두 가지 제어문이 있습니다. 조건문과 반복문이 있는데 이번 시간에는 조건문을 이용하여 프로그램을 만들어 봅니다. MyApp 프로젝트에 AccountingIFApp 클래스를 생성합니다.
 AccountingIFApp.java

```
public class AccountingIFApp {

    public static void main(String[] args) {

        double valueOfSupply = Double.parseDouble(args[0]);
        double vatRate = 0.1;
        double expenseRate = 0.3;
        double vat = valueOfSupply * vatRate;
        double total = valueOfSupply + vat;
        double expense = valueOfSupply * expenseRate;
        double income = valueOfSupply - expense;

        double dividend1;
        double dividend2;
        double dividend3;

        if(income > 10000.0) {
            dividend1 = income * 0.5;
            dividend2 = income * 0.3;
            dividend3 = income * 0.2;
        } else {
            dividend1 = income * 1.0;
            dividend2 = income * 0;
            dividend3 = income * 0;
```

```
        }

        System.out.println("Value of supply : " + valueOfSupply);
        System.out.println("VAT : " + vat);
        System.out.println("Total : " + total);
        System.out.println("Expense : " + expense);
        System.out.println("Income : " + income);
        System.out.println("Dividend 1 : " + dividend1);
        System.out.println("Dividend 2 : " + dividend2);
        System.out.println("Dividend 3 : " + dividend3);

    }

}
```

우리의 문제상황에 새로운 조건이 하나 생기게 되었다고 해 봅시다. 수익(income)이 1만원이 넘을 경우에만 동업자들에게 기존의 배당률대로 수익을 배당하고, 수익이 1만원 이하일 경우에는 1번 동업자에게만 모든 수익을 배당하게 되었습니다. 이를 우리가 배운 내용만 가지고 만들기 위해서는 수익이 1만원이 넘을 경우에 사용할 프로그램 하나, 수익이 1만원 이하일 경우에 사용할 프로그램 하나, 총 두개의 프로그램을 만들어야 할 것입니다. 이런 경우 조건문을 사용하게 되면 프로그램을 두 개 만들 필요 없이 하나의 프로그램으로 모든 문제를 해결할 수 있게 됩니다.

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/if.html

6) 배열

- 배열 : 우리는 이미 배열을 다뤘던 적이 있습니다. main 메소드에 args 파라미터가 있었습니다. args는 String 자료를 여러 개 담을 수 있는 문자열 배열입니다. 배열은 같은 자료형을 여러 개 담을 수 있는 객체입니다. 다시 조건문에서 한 발자국 뒤로 돌아가서 AccountingApp 애플리케이션에 배열을 적용해 봅니다. MyApp 프로젝트에 AccountingArrayApp 클래스를 생성합니다.

AccountingArrayApp.java

```
public class AccountingArrayApp {

    public static void main(String[] args) {

        double valueOfSupply = Double.parseDouble(args[0]);
        double vatRate = 0.1;
        double expenseRate = 0.3;
        double vat = valueOfSupply * vatRate;
        double total = valueOfSupply + vat;
        double expense = valueOfSupply * expenseRate;
        double income = valueOfSupply - expense;
```

```
        double[] dividendRates = new double[3];
        dividendRates[0] = 0.5;
        dividendRates[1] = 0.3;
        dividendRates[2] = 0.2;

        double dividend1 = income * dividendRates[0];
        double dividend2 = income * dividendRates[1];
        double dividend3 = income * dividendRates[2];

        System.out.println("Value of supply : " + valueOfSupply);
        System.out.println("VAT : " + vat);
        System.out.println("Total : " + total);
        System.out.println("Expense : " + expense);
        System.out.println("Income : " + income);
        System.out.println("Dividend 1 : " + dividend1);
        System.out.println("Dividend 2 : " + dividend2);
        System.out.println("Dividend 3 : " + dividend3);

    }

}
```

우리의 애플리케이션에서 배당률에 해당하는 부분을 변수화 해 봅니다.

```
public class AccountingArrayApp {

    public static void main(String[] args) {

        …

        double rate1 = 0.5;
        double rate2 = 0.3;
        double rate3 = 0.2;

                double dividend1 = income * rate1;
        double dividend2 = income * rate2;
                double dividend3 = income * rate3;

        …

    }

}
```

이럴 경우 수익을 배당할 동업자가 점점 늘어나게 되었을 때 배당률 변수를 하염없이 계속 추가해야 합니다. 배당률 변수를 하나로 묶어서 표현하면 변수를 무한정 늘리지 않아도 됩니다. 이럴 경우 사용하는 것이 배열입니다.

…

```
double[] dividendRates = new double[3];
dividendRates[0] = 0.5;
dividendRates[1] = 0.3;
dividendRates[2] = 0.2;

        double dividend1 = income * dividendRates[0];
double dividend2 = income * dividendRates[1];
        double dividend3 = income * dividendRates[2];

…
```

배열은 데이터 타입 옆에 대괄호([])를 붙여서 표현하고, 인스턴스를 만들 때는 배열의 길이를 지정하여 생성합니다. 배열 내의 데이터는 인덱스를 이용하여 접근할 수 있습니다. 인덱스는 0부터 시작합니다. 프로그램의 기능은 변하지 않았지만, 배열로 묶어서 표현함으로써 각각의 배당률이 서로 연관된 정보라는 것을 분명히 할 수 있게 되었고, 변수의 개수도 줄어들게 되었습니다.

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html

7) 반복문

- 반복문 : 반복문은 조건이 참인 한, 해당되는 구간을 계속 반복합니다. 배열과 반복문을 함께 이용하면 프로그램을 훨씬 간결하게 만들 수 있습니다. MyApp 프로젝트에 AccountingArrayLoopApp 클래스를 생성합니다.

AccountingArrayLoopApp.java

```
public class AccountingArrayLoopApp {

    public static void main(String[] args) {

        double valueOfSupply = Double.parseDouble(args[0]);
        double vatRate = 0.1;
        double expenseRate = 0.3;
        double vat = valueOfSupply * vatRate;
        double total = valueOfSupply + vat;
        double expense = valueOfSupply * expenseRate;
        double income = valueOfSupply - expense;



        System.out.println("Value of supply : " + valueOfSupply);
```

```
        System.out.println("VAT : " + vat);
        System.out.println("Total : " + total);
        System.out.println("Expense : " + expense);
        System.out.println("Income : " + income);

        double[] dividendRates = new double[3];
        dividendRates[0] = 0.5;
        dividendRates[1] = 0.3;
        dividendRates[2] = 0.2;


        int i = 0;
        while(i < dividendRates.length) {
            System.out.println("Dividend : " + (income*dividendRates[i]) );
            i = i + 1;
        }


    }

}
```

while 문은 괄호 안의 조건이 참인 한 블록 내의 작업을 계속 반복합니다. 이번에는 배열이 길이만큼 반복하여 동업자에게 배당 금액을 계산하는 반복문을 만들어 봅니다.

```
        ....

        double[] dividendRates = new double[3];
        dividendRates[0] = 0.5;
        dividendRates[1] = 0.3;
        dividendRates[2] = 0.2;

        double dividend1 = income * dividendRates[0];
        double dividend2 = income * dividendRates[1];
        double dividend3 = income * dividendRates[2];

        System.out.println("Dividend 1 : " + dividend1);
        System.out.println("Dividend 2 : " + dividend2);
        System.out.println("Dividend 3 : " + dividend3);
```

기존의 프로그램에서 동업자에게 배당하는 금액은 수익 * 배당률로 세 번씩 반복하고 있는 것을 알 수 있습니다. 마찬가지로 그 배당 금액을 출력하는 작업을 세 번씩 반복하고 있습니다.

```
        ....
```

```
        double[] dividendRates = new double[3];
        dividendRates[0] = 0.5;
        dividendRates[1] = 0.3;
        dividendRates[2] = 0.2;

        int i = 0;
        while (i < dividendRates.length) {
            System.out.println("Dividend : " + income * dividendRates[i]);
            i = i + 1;
        }

//          double dividend1 = income * dividendRates[0];
//          double dividend2 = income * dividendRates[1];
//          double dividend3 = income * dividendRates[2];

//          System.out.println("Dividend 1 : " + dividend1);
//          System.out.println("Dividend 2 : " + dividend2);
//          System.out.println("Dividend 3 : " + dividend3);
```

while 문을 이용하여 반복적으로 동작하던 작업을 간결하게 바꿀 수 있게 되었습니다. 반복문의 진가는 반복할 횟수가 늘어나게 되면 드러납니다. 지금은 동업자가 3명뿐이지만 백만 명으로 늘어난다면, 기존의 방식으로는 2억줄이 넘는 코드가 늘어져 있을 것입니다. 하지만 반복문을 사용하게 되면 3명이든 백만 명이든 언제나 5줄의 코드일 것입니다.

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/while.html

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/for.html

8) 메소드

- 메소드 : 클래스의 동작을 나타내는 함수입니다. 이전에 다루어 보았던 Math의 floor, ceil, PrintWriter의 write, close 등이 메소드입니다. 그 메소드를 이용해서 정리한느 방법을 알바봅니다. MyApp 프로젝트에서 AccountringMethodApp 클래스를 생성합니다.

AccountingMethodApp.java

```
public class AccountingMethodApp {
    public static double valueOfSupply;
    public static double vatRate;
    public static double expenseRate;
    public static void main(String[] args) {
        valueOfSupply = 10000.0;
        vatRate = 0.1;
        expenseRate = 0.3;
        print();
    }

    public static void print() {
```

```
        System.out.println("Value of supply : " + valueOfSupply);
        System.out.println("VAT : " + getVAT());
        System.out.println("Total : " + getTotal());
        System.out.println("Expense : " + getExpense());
        System.out.println("Income : " + getIncome());
        System.out.println("Dividend 1 : " + getDiviend1());
        System.out.println("Dividend 2 : " + getDiviend2());
        System.out.println("Dividend 3 : " + getDiviend3());
    }

    public static double getDiviend1() {
        return getIncome() * 0.5;
    }
    public static double getDiviend2() {
        return getIncome() * 0.3;
    }
    public static double getDiviend3() {
        return getIncome() * 0.2;
    }

    public static double getIncome() {
        return valueOfSupply - getExpense();
    }

    public static double getExpense() {
        return valueOfSupply * expenseRate;
    }

    public static double getTotal() {
        return valueOfSupply + getVAT();
    }

    public static double getVAT() {
        return valueOfSupply * vatRate;
    }

}
```

- 메소드로 추출하기 : 우선 vat를 메소드로 추출하는 법을 알아봅니다. vat의 값에 해당하는 부분을 블록으로 지정한 후 Refactor - Extract Method를 클릭합니다. 위와 같이 getVAT로 메소드의 이름을 지정하고 public 옵션을 선택한 후 OK를 클릭합니다.

```
public class AccountingMethodApp {
```

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

        …

        double vat = getVAT(valueOfSupply, vatRate);

        …
    }

        public static double getVAT(double valueOfSupply, double vatRate) {
                return valueOfSupply * vatRate;
        }

}
```

위와 같이 vat를 구하는 getVAT 메소드가 만들어져서 vat 변수는 getVAT 메소드를 통해 값을 가져옵니다. getVAT는 main 메소드의 내부의 변수(지역 변수)인 valueOfSupply와 vatRate를 파라미터로 사용합니다. 지역 변수를 전역 변수로 바꾸면 메소드의 파라미터를 없앨 수 있습니다. getVAT의 아규먼트로 주고 있는 valueOfSupply와 vatRate 변수에서 Refactor - Convert Local Variable to Filed를 클릭합니다. 전역 변수는 Field라고도 합니다. 변수 이름을 지정하고 public 변수로 만듭니다.

```
public class AccountingMethodApp {

    public static double valueOfSupply;

        public static void main(String[] args) {

        valueOfSupply = 10000.0;
        double vatRate = 0.1;

        …
}
```

valueOfSupply는 클래스 어디에서도 접근할 수 있는 변수가 되었고, 변수의 값은 main 메소드 안에서 할당하고 있습니다. 이제 getVAT의 파라미터를 없앨 수 있습니다.

```
public class AccountingMethodApp {

    public static double valueOfSupply;
        public static double vatRate;

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

```
        …

        double vat = getVAT();

       …

    }

        public static double getVAT() {
                return valueOfSupply * vatRate;
        }

}
```

마찬가지로 다른 변수들에 대해서도 메소드화를 만듭니다.

```
public class AccountingMethodApp {

    …

        print();

    }

        public static void print() {
                System.out.println("Value of supply : " + valueOfSupply);
        System.out.println("VAT : " + getVAT());
        System.out.println("Total : " + getTotal());
        System.out.println("Expense : " + getExpense());
        System.out.println("Income : " + getIncome());
        System.out.println("Dividend 1 : " + getDividend1());
        System.out.println("Dividend 2 : " + getDividend2());
        System.out.println("Dividend 3 : " + getDividend3());
        }

        …
}
```

출력하는 부분들도 메소드로 정리합니다. 이제 main 메소드 안의 지역 변수들은 필요가 없어졌기 때문에 모두 지워도 상관없습니다. main 메소드 안의 동작들은 단순해졌고, 세부적인 동작들은 다른 메소드 안에서 정의되어 프로그램을 사용하는 입장에서 보기 단정한 코드로 바뀌었습니다.

https://docs.oracle.com/javase/tutorial/java/javaOO/classvars.html

9) 클래스

- 클래스 : 서로 연관된 변수와 메소드를 묶어 놓은 것이라는 것을 지난 시간에 알아보았습니다. 이번 시간에는 클래스를 이용해서 프로그램을 정리하는 방법에 대해서 알아봅니다. MyApp 프로젝트에서 AccountingClassApp 클래스를 생성합니다.

AccountingClassApp.java

```
class Accounting{
    public static double valueOfSupply;
    public static double vatRate;
    public static double expenseRate;
    public static void print() {
        System.out.println("Value of supply : " + valueOfSupply);
        System.out.println("VAT : " + getVAT());
        System.out.println("Total : " + getTotal());
        System.out.println("Expense : " + getExpense());
        System.out.println("Income : " + getIncome());
        System.out.println("Dividend 1 : " + getDiviend1());
        System.out.println("Dividend 2 : " + getDiviend2());
        System.out.println("Dividend 3 : " + getDiviend3());
    }

    public static double getDiviend1() {
        return getIncome() * 0.5;
    }
    public static double getDiviend2() {
        return getIncome() * 0.3;
    }
    public static double getDiviend3() {
        return getIncome() * 0.2;
    }

    public static double getIncome() {
        return valueOfSupply - getExpense();
    }

    public static double getExpense() {
        return valueOfSupply * expenseRate;
    }

    public static double getTotal() {
        return valueOfSupply + getVAT();
    }
```

```
    public static double getVAT() {
        return valueOfSupply * vatRate;
    }
}
public class AccountingClassApp {

    public static void main(String[] args) {
        Accounting.valueOfSupply = 10000.0;
        Accounting.vatRate = 0.1;
        Accounting.expenseRate = 0.3;
        Accounting.print();
        // anotherVariable = ...;
        // anotherMethod = ...;
    }



}
```

이클립스의 상단 Window - Show View - Outline을 클릭합니다. 클래스에 포함되어 있는 변수와 메소드들이 정리되어 나타나 있는 것을 알 수 있습니다. 다시 편집기로 올라와서 맨 위에 Accounting 클래스를 생성합니다.

```
class Accounting {

        public static double valueOfSupply;
        public static double vatRate;
        public static double expenseRate;

        public static void print() {
                System.out.println("Value of supply : " + valueOfSupply);
        System.out.println("VAT : " + getVAT());
        System.out.println("Total : " + getTotal());
        System.out.println("Expense : " + getExpense());
        System.out.println("Income : " + getIncome());
        System.out.println("Dividend 1 : " + getDividend1());
        System.out.println("Dividend 2 : " + getDividend2());
        System.out.println("Dividend 3 : " + getDividend3());
        }

        public static double getDividend3() {
                return getIncome() * 0.2;
        }
```

```
	public static double getDividend2() {
		return getIncome() * 0.3;
	}

	public static double getDividend1() {
		return getIncome() * 0.5;
	}

	public static double getIncome() {
		return valueOfSupply - getExpense();
	}

	public static double getExpense() {
		return valueOfSupply * expenseRate;
	}

	public static double getTotal() {
		return valueOfSupply + getVAT();
	}

	public static double getVAT() {
		return valueOfSupply * vatRate;
	}

}

public class AccountingClassApp {
	....
}
```

이와 같이 Outline이 바뀌게 됩니다.

```
public class AccountingClassApp {



	public static void main(String[] args) {

		Accounting.valueOfSupply = 10000.0;
		Accounting.vatRate = 0.1;
		Accounting.expenseRate = 0.3;
		Accounting.print();
```

```
    }

}
```

AccountingClassApp의 main 메소드를 위와 같이 수정하면 똑같이 동작하는 것을 확인할 수 있습니다.

https://docs.oracle.com/javase/tutorial/java/javaOO/classes.html

10) 인스턴스

- 인스턴스 : PrintWriter 클래스를 이용해서 인스턴스에 대해서 알아 보았습니다. 클래스는 실제로 실행시킨 실체화된 클래스라고 할 수 있습니다. 이를 통해 다양한 상태에 있는 여러 개의 클래스를 돌릴 수 있습니다. 지난 시간에 이어 AccountingClassApp 클래스를 그대로 이용합니다

AccountingClassApp.java

```
class Accounting{
    public double valueOfSupply;
    public double vatRate;
    public double expenseRate;
    public void print() {
        System.out.println("Value of supply : " + valueOfSupply);
        System.out.println("VAT : " + getVAT());
        System.out.println("Total : " + getTotal());
        System.out.println("Expense : " + getExpense());
        System.out.println("Income : " + getIncome());
        System.out.println("Dividend 1 : " + getDiviend1());
        System.out.println("Dividend 2 : " + getDiviend2());
        System.out.println("Dividend 3 : " + getDiviend3());
    }

    public double getDiviend1() {
        return getIncome() * 0.5;
    }
    public double getDiviend2() {
        return getIncome() * 0.3;
    }
    public double getDiviend3() {
        return getIncome() * 0.2;
    }

    public double getIncome() {
        return valueOfSupply - getExpense();
    }
```

```
    public double getExpense() {
        return valueOfSupply * expenseRate;
    }

    public double getTotal() {
        return valueOfSupply + getVAT();
    }

    public double getVAT() {
        return valueOfSupply * vatRate;
    }
}
public class AccountingClassApp {

    public static void main(String[] args) {
        // instance
        Accounting a1 = new Accounting();
        a1.valueOfSupply = 10000.0;
        a1.vatRate = 0.1;
        a1.expenseRate = 0.3;
        a1.print();

        Accounting a2 = new Accounting();
        a2.valueOfSupply = 20000.0;
        a2.vatRate = 0.05;
        a2.expenseRate = 0.2;
        a2.print();

        a1.print();
    }
}
```

.여러 조건에 따라서 다른 출력을 내고 싶다고 가정해 봅니다.

1. OO 상품
공급가 : 10000
부가가치세율 10%
공급가 대비 비용률 : 30%
2. XX 상품
공급가 : 20000
부가가치세율 5%
공급가 대비 비용률 : 20%

and so on...
이럴 경우 기존의 방식으로 프로그램을 실행하게 된다면, 변수를 조정하는 과정에서 프로그램이 꼬일 수 있습니다. 이 때 각각의 상태에 따라 인스턴스를 생성해서 실행하는 것이 방법입니다. 우선 Accounting 클래스의 static 키워드를 모두 제거합니다.

```
class Accounting {

	public double valueOfSupply;
	public double vatRate;
	public double expenseRate;

	public void print() {
		System.out.println("Value of supply : " + valueOfSupply);
	System.out.println("VAT : " + getVAT());
	System.out.println("Total : " + getTotal());
	System.out.println("Expense : " + getExpense());
	System.out.println("Income : " + getIncome());
	System.out.println("Dividend 1 : " + getDividend1());
	System.out.println("Dividend 2 : " + getDividend2());
	System.out.println("Dividend 3 : " + getDividend3());
	}

	public double getDividend3() {
		return getIncome() * 0.2;
	}

	public double getDividend2() {
		return getIncome() * 0.3;
	}

	public double getDividend1() {
		return getIncome() * 0.5;
	}

	public double getIncome() {
		return valueOfSupply - getExpense();
	}

	public double getExpense() {
		return valueOfSupply * expenseRate;
	}
```

```
	public double getTotal() {
		return valueOfSupply + getVAT();
	}

	public double getVAT() {
		return valueOfSupply * vatRate;
	}

}
```

이번 상황에서는 두 가지 상태가 있기 때문에 두 개의 인스턴스를 생성합니다.

....

```
public class AccountingClassApp {



	public static void main(String[] args) {


		Accounting a1 = new Accounting();
		Accounting a2 = new Accounting();

	....
```

그리고 각각의 인스턴스를 상태에 맞추어 조절합니다.

....

```
Accounting a1 = new Accounting();
a1.valueOfSupply = 10000.0;
a1.vatRate = 0.1;
a1.expenseRate = 0.3;
a1.print();

Accounting a2 = new Accounting();
a2.valueOfSupply = 20000.0;
a2.vatRate = 0.05;
a2.expenseRate = 0.2;
a2.print();
```

....

a1 인스턴스는 a2 인스턴스의 작업이 끝난 후에도 여전히 남아 있습니다. 그래서 마지막에 a1 인스턴스의 print 메소드를 실행해도 같은 결과를 냅니다.

https://docs.oracle.com/javase/tutorial/java/javaOO/objectcreation.html