아이템25 (톱레벨 클래스는 한 파일에 하나만 담아라)

소스 파일 하나에 톱레벨 클래스를 여러 개 정의하면 심각한 위험을 감수해야 한다. 한 클래스를 여러가지로 정의할 수 있으며 그 중 어느 것을 사용하는지 어느 소스파일을 먼저 컴파일하냐에 따라 달라지기 때문이다.

Utensil.java

```java
package main;

class Utensil {
    static final String NAME = "pan";
}

class Dessert {
    static final String NAME = "cake";
}
```

- Utensil과 Dessert가 정의된 Utensil.java 파일

Dessert.java

```java
package main;

class Utensil {
    static final String NAME = "pot";
}

class Dessert {
    static final String NAME = "cake";
}
```

- Utensil과 Dessert가 정의된 Dessert.java 파일

```
choiy@DESKTOP-KU2RVB8 MINGW64 ~/OneDrive/바탕 화면/StudyDir
$ javac Main.java
Main.java:7: error: cannot find symbol
        System.out.println(Utensil.NAME + Dessert.NAME);
                           ^
  symbol:   variable Utensil
  location: class Main
Main.java:7: error: cannot find symbol
        System.out.println(Utensil.NAME + Dessert.NAME);
                                          ^
  symbol:   variable Dessert
  location: class Main
2 errors
```

- javac Main.java 명령어로 컴파일

IDE에서도 빨간줄이 나타나고 Main.java 컴파일시 컴파일 오류가 발생한다.

```
choiy@DESKTOP-KU2RVB8 MINGW64 ~/One
$ javac Main.java Utensil.java
```

- javac Main.java Utensil.java 명령어로 컴파일

```
package main;

public class Main {
    public Main() {
    }

    public static void main(String[] var0) {
        System.out.println("pancake");
    }
}
```

- 결과

```
choiy@DESKTOP-KU2RVB8 MINGW64 ~/One
$ javac Dessert.java Main.java
```

- javac Dessert.java Main.java 명령어로 컴파일

```
public class Main {
    public Main() {
    }

    public static void main(String[] var0) {
        System.out.println("potcake");
    }
}
```

- 결과

건내주는 명령어에 따라 실행 결과가 달라지는 것을 볼 수 있다. 이처럼 컴파일러에 어느 소스파일을 먼저 건네느냐에 따라 동작이 달라지므로 반드시 바로잡아야 한다.

해결책

해결책은 톱레벨 클래스들을 서로 다른 소스파일로 분리하는 것이다.

정리

소스파일 하나에는 반드시 톱레벨 클래스(혹은 톱레벨 인터페이스)를 하나만 담자. 이 규칙을 따르면 컴파일러가 한 클래스에 대한 정의를 여러 개 만들어 내는 일이 사라지고 어떤 소스파일 순서로 컴파일하든 바이너리 파일이나 프로그램의 동작이 달라지지 않는다.