# (디지털컨버전스) 스마트 콘텐츠와 웹 융합 응용 SW개발자 양성과정

## -18일차 학습 및 질문 노트-

강사 - Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 – Kyeonghwan Lee(이경환)
airtrade7@naver.com

# ▣interface implements

```java
interface LampMethod {
    public void lightOn();
    public void lightOff();
}
class Lamp implements LampMethod {
    @Override
    public void lightOn() { System.out.println("Lamp를 킵니다."); }
    @Override
    public void lightOff() { System.out.println("Lamp를 끕니다."); }
}
class StreetLamp implements LampMethod {
    @Override
    public void lightOn() { System.out.println("가로등을 킵니다."); }
    @Override
    public void lightOff() { System.out.println("가로등을 끕니다."); }
}
class Led implements LampMethod {
    @Override
    public void lightOn() { System.out.println("LED등을 킵니다."); }
    @Override
    public void lightOff() { System.out.println("LED등을 끕니다."); }
}
public class InterfaceTest {
    public static void main(String[] args) {
        Lamp lamp = new Lamp();
        lamp.lightOn();
        lamp.lightOff();
        StreetLamp streetLamp = new StreetLamp();
        streetLamp.lightOn();
        streetLamp.lightOff();
        Led led = new Led();
        led.lightOn();
        led.lightOff();
    }
}
```

17일차에는 클래스 내부에 인터페이스에 대한 객체를 생성한 반면 이번에는 Implements를 사용하여 해당 클래스에서 인터페이스 내부의 미 구현 매서드를 구현해줌으로서 동작을 하게 된다.

동일하게 출력

```
Lamp를 킵니다.
Lamp를 끕니다.
가로등을 킵니다.
가로등을 끕니다.
LED등을 킵니다.
LED등을 끕니다.

Process finished with exit code 0
```

# ▣HashSet

```java
import java.util.HashSet;

public class HashSetTest {
    public static void main(String[] args) {
        HashSet<String> set = new HashSet<~>();

        set.add("우유");
        set.add("빵");
        set.add("베이컨");
        set.add("소시지");
        set.add("HAM");
        set.add("파스타");
        set.add("계란");
        set.add("아메리카노");
        set.add("HAM");
        set.add("ham");
        // HashSet의 핵심 특성중 하나 Java내에 존재하는 Collection중 가장 빠른 속도를 자랑함
        // 또한 HashSet의 집합(Set)의 특성을 가지고 있어 중복을 허용하지 않는다.
        // 중요: 순서가 중요하다면 ArrayList를 사용하세요.
        //       순서가 별로 중요하지 않고 빠른 처리를 원한다면 HashSet을 권장합니다.

        System.out.println(set);
    }
}
```

**HashSet : "중복해서 저장하지 않는" 집합으로 사용할 수 있는 클래스.**

**사용법:HashSet<데이터타입> 객체명 = new HashSet<데이터타입>();**

- **HashSet의 핵심 특성 중 하나 Java내에 존재하는 Collection중 가장 빠른 속도를 자랑함**
- **HashSet의 집합(Set)의 특성을 가지고 있어 중복을 허용하지 않는다.**
- **순서가 중요하다면 ArrayList를 사용하세요.**
- **순서가 별로 중요하지 않고 빠른 처리를 원한다면 HashSet을 권장합니다**

**중복 값이 출력 되지 않는다.**

```
[빵, HAM, ham, 우유, 파스타, 베이컨, 계란, 아메리카노, 소시지]


Process finished with exit code 0
```

# ▣HashSet(union, intersection)

```java
import java.util.HashSet;
import java.util.Set;

public class SetFeatureTestWithHashSet {
    public static void main(String[] args) {
        Set<String> s1 = new HashSet<String>();
        Set<String> s2 = new HashSet<String>();

        s1.add("Apple");
        s1.add("Tesla");
        s1.add("Microsoft");

        s2.add("Tesla");
        s2.add("Alphabet");
        s2.add("Texas Instruments");

        Set<String> union = new HashSet<String>(s1);
        union.addAll(s2);

        Set<String> intersection = new HashSet<>(s1);
        intersection.retainAll(s2);

        System.out.println("합집합: " +union);
        System.out.println("교집합: "+ intersection);


    }
}
```

**union.addALL(): 합집합**
**Intersection.retainAll():교집합**

```
합집합: [Alphabet, Texas Instruments, Apple, Tesla, Microsoft]
교집합: [Tesla]

Process finished with exit code 0
```

# ▣HashMap

```java
import java.util.HashMap;
import java.util.Map;

class Student {
    int age;
    String name;

    public Student (int age, String name) {
        this.age = age;
        this.name = name;
    }

    @Override
    public String toString() {
        return "Student{" +
                "age=" + age +
                ", name='" + name + '\'' +
                '}';
    }
}
```

**HashMap : 키 값과 데이터를 사용하여 HashTable의 값을 찾음.**

**사용법:HaspMap<키의 타입, 데이터의 타입> 객체명 = new HaspMap<키의 타입, 데이터의 타입>();**

- **Key와 value가 분리됨**
- **Map<Key, value>**
- **특별히 특정 데이터타입을 지켜줘야 하는 것은 없다.**
- **앞에 오는 숫자는 인덱스가 아니다.(key값)**

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

        Map<Integer, Student> st = new HashMap<~>();

        // 앞에 오는 숫자는 인덱스가 아니다.
        // 단지 사물함을 여는데 필요한 열쇠일 뿐
        st.put(7, new Student( age: 42,  name: "Bob"));
        st.put(2, new Student( age: 33,  name: "Chris"));
        st.put(44, new Student( age: 27,  name: "Denis"));
        st.put(3, new Student( age: 29,  name: "David"));

        System.out.println(st);

        st.remove( key: 2);

        System.out.println(st);

        st.put(3, new Student( age: 77,  name: "Jesica"));

        System.out.println(st);

        for (Map.Entry<Integer, Student> s : st.entrySet()) {
            Integer key = s.getKey();
            Student value = s.getValue();
            System.out.println("key = " + key + ", value = " + value);
        }
        Map<String, String> strMap = new HashMap<~>();

        strMap.put("열쇠", "으아아앜!");
        // HashMap을 사용할때는 이 방식이 변하지 않습니다.
        // 추상화의 연장선 관점에서 아래 사항을 준수하여 코딩하면 어떤 상황에서든 key, value 값을 얻을 수 있습니다.
        // Entry<키 데이터타입, 벨류 데이터타입> 형식은 지켜주세요.
        for (Map.Entry<String, String> map : strMap.entrySet()) {
            String key = map.getKey();
            String value = map.getValue();
            System.out.println("key = " + key + ", value = " + value);
        }
    }
```

```
{2=Student{age=33, name='Chris'}, 3=Student{age=29, name='David'}, 7=Student{age=42, name='Bob'}, 44=Student{age=27, name='Denis'}}
{3=Student{age=29, name='David'}, 7=Student{age=42, name='Bob'}, 44=Student{age=27, name='Denis'}}
{3=Student{age=77, name='Jesica'}, 7=Student{age=42, name='Bob'}, 44=Student{age=27, name='Denis'}}
key = 3, value = Student{age=77, name='Jesica'}
key = 7, value = Student{age=42, name='Bob'}
key = 44, value = Student{age=27, name='Denis'}
key = 열쇠, value = 으아아앜!

Process finished with exit code 0
```

**입력 순서와 무관하게 key값의 오름차순 출력**

**remove(key): 해당 데이터 값(key) 삭제 후 정렬**

**put(key, 데이터): 해당 데이터 값(key) 수정**