(디지털컨버전스) 스마트 콘텐츠와 웹 융합 응용SW개발자 양성과정

2021년 6월 3일
[19일차 복습]

- 수강생 : 김 민 규
- 강의장 : 강 남 C
- 수강 기간 : 2021. 05. 07 ~ 2021. 12. 08
- 수강 시간 : 15:30 ~ 22:00
- 이상훈 강사님 | 이은정 취업담임

▶ 내용 : 인터페이스(interface)_implements

```
+ class Lamp implements LampMethod {
+     @Override
+     public void lightOn() {
+         System.out.println("Lamp를 킵니다.");
+     }
+
+     @Override
+     public void lightOff() {
+         System.out.println("Lamp를 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();
    }
}
```

```
6   class Lamp {
7       LampMethod lamp = new LampMethod() {
8           @Override
9           public void lightOn() {
10              System.out.println("Lamp를 킵니다.");
11          }
12
13          @Override
14          public void lightOff() {
15              System.out.println("Lamp를 끕니다.");
16          }
17      };
18  }
```

```
48  public class g_Quiz54_Interface {
49      public static void main(String[] args) {
50          Lamp lamp = new Lamp();
51
52          lamp.lamp.lightOn();
53          lamp.lamp.lightOff();
54
55          StreetLamp streetLamp = new StreetLamp();
56
57          streetLamp.streetLamp.lightOn();
58          streetLamp.streetLamp.lightOff();
59
60          Led led = new Led();
61
62          led.led.lightOn();
63          led.led.lightOff();
64      }
65  }
```

주요 내용

--

Implements를 사용하게 된다면,
중간 매개체를 지워도된다.

Class 클래스명 impements 인터페이스명

--

▶ 내용 :

```
+ class LampMethod2 {
+     public void lightOn() {
+         System.out.println("독립은 무슨 종속이다!");
+     }
+     public void lightOff() {
+         System.out.println("무조건 우리말에 따르라 ~~!!~!");
+     }
+ }
+
+ // 저번에는 클래스 내부에 인터페이스에 대한 객체를 생성한 반면
+ // 이번에는 implements를 사용하여 해당 클래스에서
+ // 인터페이스 내부의 미구현 매서드를 구현해줌으로서 동작을 하게 된다.
+ class Lamp2 extends LampMethod2 {
+
+ }
+
+ class StreetLamp2 extends LampMethod2 {
+
+ }
+
+ class Led2 extends LampMethod2 {
+
```

```
+ public class InterfaceVersusExtendsTest {
+     public static void main(String[] args) {
+         Lamp2 lamp = new Lamp2();
+
+         lamp.lightOn();
+         lamp.lightOff();
+
+         StreetLamp2 streetLamp = new StreetLamp2();
+
+         streetLamp.lightOn();
+         streetLamp.lightOff();
+
+         Led2 led = new Led2();
+
+         led.lightOn();
+         led.lightOff();
+     }
+ }
```

주요 내용

--

LampMethod2에 종속되어

Lamp2 와 StrretLamp2와 Led2는

LampMethod2의 메서드를 동작(출력)하게된다.

--

▶ 내용 : HashSet_ addAll 과 retainAll

```
+ 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<String>(s1);
+         intersection.retainAll(s2);
+
+         System.out.println("합집합: " + union);
+         System.out.println("교집합: " + intersection);
+     }
+ }
```

주요 내용

--

HashSet
Java내의 collection 중 가장 빠른 속도를 자랑함

또한 Set(집합)의 특성에 따라
중복을 허용하지 않는다.

--

순서가 중요할 경우 ArrayList 사용
빠른 처리만 원한다면 HashSet 권장

--

addAll – 합집함

retainAll –교집합

▶ 내용 : HashSet_ add

```
1  + import java.util.HashSet;
2  + import java.util.Set;
3  +
4  + public class HowToUseHashSet {
5  +     public static void main(String[] args) {
6  +         Set<String> s = new HashSet<String>();
7  +
8  +         String[] sample = {"안녕", "하이", "헬로", "안녕", "안녕"};
9  +
10 +         // 집합의 특성: 중복 허용 x
11 +         for (String str : sample) {
12 +             if (!s.add(str)) {
13 +                 System.out.println("중복되었습니다: " + str);
14 +             }
15 +         }
16 +
17 +         // size()는 원소의 개수
18 +         System.out.println(s.size() + " 중복을 제외한 단어: " + s);
19 +     }
20 + }
```

주요 내용

--
HashSet은 집합의 특성으로 중복을 허용하지 않음

if (!s.add(str)) 에서
s.add(str)는 str에
지정된 값들이 있을 경우 True이지만

!s.add(str)이므로
지정되지 않은 값들에 대하여 출력이 이루어짐.

즉, 중복된 값들에 대해서만
출력이 이루어짐.

--
Size는 원수의 갯수를 의미함.
Length와 유사.

▶ 내용 :

```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 + '\'' +
                '}';
    }
}

public class HashMapTest {
    public static void main(String[] args) {
        // Map의 특성중 하나가 key와 value가 분리됨
        // Map<Key, Value>
        // 특별히 특정 데이터타입을 지켜줘야 하는 것은 없다.
        // 나는 "열쇠"를 키로 사용하고 "으아아앜!"을 값으로 쓸거야! 하면 쓰면 된다.
        Map<Integer, Student> st = new HashMap<Integer, Student>();

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

        st.remove(2);

        System.out.println(st);

        st.put(3, new Student(77, "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);
        }

        // 나는 "열쇠"를 키로 사용하고 "으아아앜!"을 값으로 쓸거야! 하면 쓰면 된다.
        Map<String, String> strMap = new HashMap<String, String>();

        strMap.put("열쇠", "으아아앜!");

        // HashMap을 사용할때는 이 방식이 변하지 않습니다.
        // 추상화의 연장선 관점에서 아래 사항을 준수하며 코딩하면 어떤 상황에서든 key, value 값을 얻을 수 있습니다.
        // Entry<키 데이터타입, 밸류 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);
        }
```

주요 내용

--

Map
Map의 특성 중 하나가
Key와 value가 분리됨
Map〈Key, Value〉

--

put
객체.put(a, b) -〉 a는 열쇠이며, b는값

Put을 통해 값을 추가하거나
수정할 수 있음

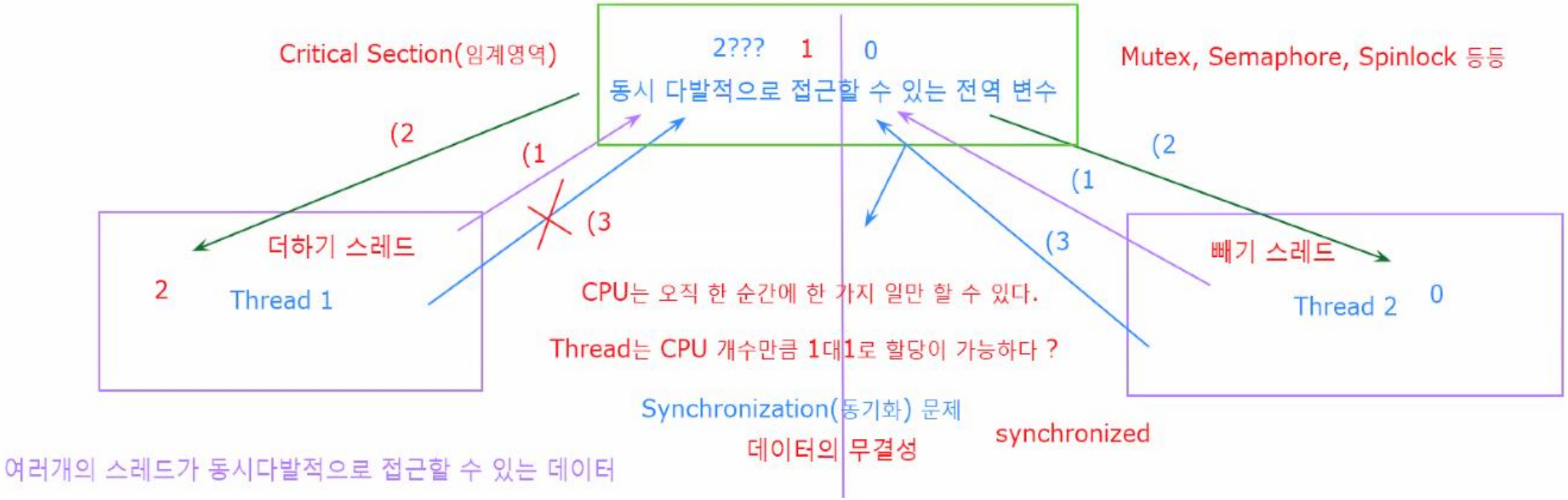
--

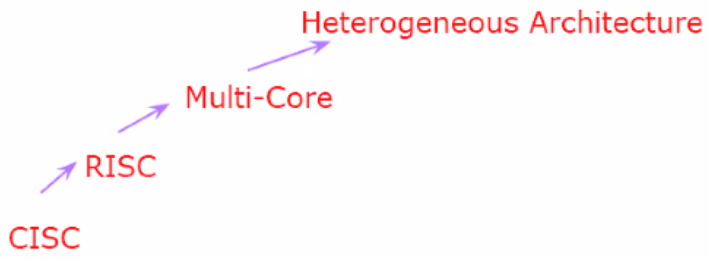
# remove
St.remove(7)
St 객체의 키값 7의 내용을 지움

--

Entry
Etnry〈키데이터타입, 벨류데이터타입〉
형식을 지켜줘야함.

▶ 내용 : 스레드(thread)





스레드의 이론적인 이해를 위해 알아야 하는 것
Multi-Tasking의 원리
Context-Switching
실제 컴퓨터가 연산하는 방식

*Critical Section(임계영역) : 여러 개의 스레드가 동시다발적으로 접근할 수 있는 데이터 (여러 개의 데이터가 접근하고 있지 않은 상태면 임계영역(Critcal Section)이라 볼 수 없다.)
→ 이로인해 동기화문제가 발생 → 동기화문제로 인하여 데이터의 무결성 문제 발생 → Synchronized, Mutex, Semaphore, Spinlock 등이 이러한 문제를 해결해주기위해 사용됨.

▶ 내용 : 스레드(thread)

스레드의 이론적인 이해를 위해 알아야 하는 것

Multi-Tasking의 원리 : 아주 빠르게 순차적으로 일어날 뿐, 실제로는 동시에 처리되는 것은 아니다.
(하나의 프로그램?에 대해서 0.1초 등과 같이 순식간에 처리하고 다음것을 연이어 처리함)
여기서 Context-Switching이 포함되어있음.

Context-Switching : 멀티프로세스 환경에서 **CPU가 어떤 하나의 프로세스를 실행하고 있는 상태에서 인터럽트 요청에 의해 다음 우선 순위의 프로세스가 실행되어야 할 때 기존의 프로세스의 상태 또는 레지스터 값(Context)을 저장하고 CPU가 다음 프로세스를 수행하도록 새로운 프로세스의 상태 또는 레지스터 값(Context)를 교체하는 작업**을 Context Switch(Context Switching)라고 한다.

▶ 내용 : 실제컴퓨터가 연산하는 방식

Threa1

```
Move eax, 3   // eax = 3                (1)
Move ebx, 4   // eax = 4                (4)
Add eax, ebx // eax = eax + ebx         (5) 7 + 4 = 11??   ➔ 데이터의 무결성이 깨짐
```

Thread2

```
Move eax, 7    // eax = 7               (2)
Move ebx, 2    // ebx = 2               (3)
Add ead, ebx  //  eax = eax + ebx
```

Context Switching의 도입이 없는 상황의 경우 이러한 식으로 프로그램이 실행될 수 있음.

Threa1

```
Move eax, 3   // eax = 3                (1)  ➔ 제어권을 넘기기 전에 현재 하드웨어 레지스터 정보를 메모리에 저장함
                                              (이 정보는 운영체제가 관리하고 있어서 찾을 수 있음)
Move ebx, 4   // eax = 4                (4) ← 다시 돌아와서 이전에 저장한 정보를 메모리에서 찾아서 복원함.
Add eax, ebx // eax = eax + ebx         (5) 3 + 4 = 7 (데이터의 무결성을 보장함)
```

Thread2

```
Move eax, 7    // eax = 7               (2)
Move ebx, 2    // ebx = 2               (3)  ➔ 자신의 제어권이 역시 넘어가기 전에 하드웨어 레지스터 정보를 백업함.
Add ead, ebx  //  eax = eax + ebx
```

Context Switching의 도입한 경우.

주요 내용

--

면접에서 자주 나오는 질문
Mutex는
Context Switching을 하고

Spinlock은
Context Switching을 하지 않는다.

복잡한 작업의 경우 Mutex가 좋음.
Context Switching을 사용하기 때문에.

스레드가 처리하는 작업이 단순할 때에는
Spinlock이 좋음.
단순한 작업에서 Context Switching을
사용할 경우 메모리가 더 많이 소요되므로.

--