Part 3. 라이브러리 활용

Chapter 12. java.base 모듈

12.1 API 도큐먼트

- 라이브러리 : 클래스와 인터페이스의 집합
- **API(Application Programming interface)** 도큐먼트 : 자바 표준 모듈에서 제공하는 라이브러리 사용하기 위한 방법 기술

다음 URL을 방문하면 JDK 버전별로 사용할 수 있는 API 도큐먼트를 볼 수 있다.

```
https://docs.oracle.com/en/java/javase/index.html
```

자바 버전 선택 후 왼쪽 메뉴에서 [API Document] 버튼을 클릭하면 각 버전에 따른 API 도큐먼트 페이지가 열린다.
여기서 **클래스 선언부, 구성 멤버, 필드, 생성자, 메소드** 등을 확인할 수 있다.

12.2 java.base 모듈

java.base는 모든 모듈이 의존하는 기본 모듈로, 모듈 중 유일하게 requires 하지 않아도 사용할 수 있다.

12.3 Object 클래스

자바의 모든 클래스는 Object의 자식이거나 자손 클래스이므로 Object가 가진 메소드는 모든 객체에서 사용할 수 있다.

객체 동등 비교

- Object의 `equals()` 메소드는 객체의 번지를 비교하고 `boolean` 값을 리턴.
- Object의 `equals()` 메소드는 재정의해서 동등 비교용으로 사용되는데, 동등 비교란 객체가 비록 달라도 내부의 데이터가 같은지를 비교하는 것을 말함.

Member.java

```
package ch12.sec03.exam01;

public class Member {
    public String id;

    public Member(String id) {
        this.id = id;
    }

    @Override
```

```
    public boolean equals(Object obj) {
        if(obj instanceof Member target) {
            if(id.equals(target.id)) {
                return true;
            }
        }
        return false;
    }
}
```

EqualsExample.java

```
package ch12.sec03.exam01;

public class EqualsExample {
    public static void main(String[] args) {
        Member obj1 = new Member("blue");
        Member obj2 = new Member("blue");
        Member obj3 = new Member("red");

        if(obj1.equals(obj2)) {
            System.out.println("obj1과 obj2는 동등합니다.");
        } else {
            System.out.println("obj1과 obj2는 동등하지 않습니다.");
        }

        if(obj1.equals(obj3)) {
            System.out.println("obj1과 obj3은 동등합니다.");
        } else {
            System.out.println("obj1과 obj3은 동등하지 않습니다.");
        }
    }
}
```

결과

```
obj1과 obj2는 동등합니다.
obj1과 obj3은 동등하지 않습니다.
```

객체 해시코드

- 객체 해시코드 : 객체를 식별하는 정수
- Object의 `hashCode()` 메소드는 객체의 메모리 번지를 이용해 해시코드를 생성하기 때문에 객체마다 다른 정수값을 리턴함.
- 두 객체가 동등한지를 비교할 때 주로 사용.

자바는 두 객체가 동등함을 비교할 때 `hashCode()`와 `equals()` 메소드를 같이 사용하는 경우가 많다.

Student.java

```
package ch12.sec03.exam02;

public class Student {
    private int no;
    private String name;

    public Student(int no, String name) {
        this.no = no;
        this.name = name;
    }

    public int getNo() { return no; }
    public String getName() { return name; }

    @Override
    public int hashCode() {
        int hashCode = no + name.hashCode();
        return hashCode;
    }

    @Override
    public boolean equals(Object obj) {
        if(obj instanceof Student target) {
            if(no == target.getNo() && name.equals(target.getName())) {
                return true;
            }
        }
        return false;
    }
}
```

HashCodeExample.java

```
package ch12.sec03.exam02;

public class HashCodeExample {

    public static void main(String[] args) {
        Student s1 = new Student(1, "홍길동");
        Student s2 = new Student(1, "홍길동");

        if(s1.hashCode() == s2.hashCode()) {
            if(s1.equals(s2)) {
                System.out.println("동등 객체입니다.");
            } else {
                System.out.println("데이터가 다르므로 동등 객체가 아닙니다.");
            }
        } else {
```

```
            System.out.println("해시코드가 다르므로 동등 객체가 아닙니다.");
        }
    }
}
```

결과

```
동등 객체입니다.
```

객체 문자 정보

- Object의 `toString()` 메소드는 객체의 문자 정보를 리턴.
- 객체의 문자 정보란 객체를 문자열로 표현한 값으로, *클래스명@16진수해시코드* 로 구성.
- 매개값이 기본 타입이거나 문자열일 경우 해당 값 그대로 출력, 객체라면 객체의 `toString()` 메소드 호출해서 리턴값 출력

레코드 선언

- 데이터 전달 위한 DTO(Data Transfer Object) 작성 시 반복적으로 사용되는 코드 줄이기 위해 도입
- `private fianl` 필드 자동 생성, 생성자 및 `Getter` 메소드 자동 추가
- `hashCode()`, `equals()`, `toString()` 메소드를 재정의한 코드 자동 추가

Member.java

```
package ch12.sec03.exam04;

public record Member(String id, String name, int age) {
}
```

RecordExample.java

```
package ch12.sec03.exam04;

public class RecordExample {
   public static void main(String[] args) {
      Member m = new Member("winter", "눈송이", 25);
      System.out.println(m.id());
      System.out.println(m.name());
      System.out.println(m.age());
      System.out.println(m.toString());
      System.out.println();

      Member m1 = new Member("winter", "눈송이", 25);
      Member m2 = new Member("winter", "눈송이", 25);
      System.out.println("m1.hashCode(): " + m1.hashCode());
      System.out.println("m2.hashCode(): " + m2.hashCode());
```

```
        System.out.println("m1.equals(m2): " + m1.equals(m2) );
    }
}
```

룸복 사용하기

- 룸복(Lombok) : 자동 코드 생성 라이브러리
- DTO 클래스 작성 시 `Getter`, `Setter`, `hashCode()`, `equals()`, `toString()` 메소드 자동 생성 (필드는 `final` 아님)

12.4 System 클래스

프로그램 종료, 키보드 입력, 콘솔(모니터) 출력, 현재 시간 읽기, 시스템 프로퍼티 읽기 등이 가능

12.5 문자열 클래스

클래스	설명
`String`	문자열을 저장하고 조작할 때 사용
`StringBuilder`	효율적인 문자열 조작 기능이 필요할 때 사용
`StringTokenizer`	구분자로 연결된 문자열을 분리할 때 사용

String 클래스

- 문자열 -> byte 배열 -> 다시 문자열

BytesToStringExample.java

```
package ch12.sec05;

import java.util.Arrays;

public class BytesToStringExample {
    public static void main(String[] args) throws Exception  {
        String data = "자바";

        //String -> byte 배열(기본: UTF-8 인코딩)
        byte[] arr1 = data.getBytes();
        //byte[] arr1 = data.getBytes("UTF-8");
        System.out.println("arr1: " + Arrays.toString(arr1));

        //byte 배열 -> String(기본: UTF-8 디코딩)
        String str1 = new String(arr1);
        //String str1 = new String(arr1, "UTF-8");
        System.out.println("str1: " + str1);

        //String -> byte 배열(EUC-KR 인코딩)
```

```
        byte[] arr2 = data.getBytes("EUC-KR");
        System.out.println("arr2: " + Arrays.toString(arr2));

        //byte 배열 -> String(기본: UTF-8 디코딩)
        String str2 = new String(arr2, "EUC-KR");
        System.out.println("str2: " + str2);
    }
}
```

결과

```
arr1: [-20, -98, -112, -21, -80, -108]
str1: 자바
arr2: [-64, -38, -71, -39]
str2: 자바
```

StringBuilder 클래스

문자열의 `+` 연산은 새로운 `String` 객체가 생성되고 이전 객체는 계속 버려지는 것이기 때문에 효율이 좋지 않다. 잦은 문자열 변경 작업을 해야 한다면 `String`보다는 내부 버퍼에 문자열을 저장해두고 그 안에서 추가, 수정, 삭제 작업을 하도록 설계된 `StringBuilder`를 사용하는 것이 좋다.

StringBuilderExample.java

```
package ch12.sec05;

public class StringBuilderExample {
    public static void main(String[] args) {
        String data = new StringBuilder()
                .append("DEF")
                .insert(0, "ABC")
                .delete(3, 4)
                .toString();
        System.out.println(data);
    }
}
```

결과

```
ABCEF
```

StringTokenizer 클래스

- 한 종류의 구분자만 있을 때 구분자 기준으로 문자열 분리 가능
- 첫 번째 매개값으로 전체 문자열, 두 번째 매개값으로 구분자 주기

- 구분자 생략시 공백이 기본 구분자 됨

StringTokenizerExample.java

```
package ch12.sec05;

import java.util.StringTokenizer;

public class StringTokenizerExample {
   public static void main(String[] args) {
      String data1 = "홍길동&이수홍,박연수";
      String[] arr = data1.split("&|,");
      for(String token : arr) {
         System.out.println(token);
      }
      System.out.println();

      String data2 = "홍길동/이수홍/박연수";
      StringTokenizer st = new StringTokenizer(data2, "/");
      while (st.hasMoreTokens()) {
         String token = st.nextToken();
         System.out.println(token);
      }
   }
}
```

12.6 포장 클래스

포장 객체는 포장하고 있는 기본 타입의 값을 변경할 수 없고, 단지 객체로 생성하는 데 목적이 있다.

박싱과 언박싱

- **박싱(boxing)** : 기본 타입의 값을 포장 객체로 만드는 과정
- 포장 클래스 변수에 기본 타입 값이 대입될 때 발생
- **언박싱(unboxing)** : 포장 객체에서 기본 타입의 값을 얻어내는 과정
- 기본 타입 변수에 포장 객체가 대입될 때 발생

BoxingUnBoxingExample.java

```
package ch12.sec06;

public class BoxingUnBoxingExample {
   public static void main(String[] args) {
      //Boxing
      Integer obj = 100;
      System.out.println("value: " + obj.intValue());

      //Unboxing
      int value = obj;
```

```
        System.out.println("value: " + value);

        //연산 시 Unboxing
        int result = obj + 100;
        System.out.println("result: " + result);
    }
}
```

12.7 수학 클래스

MathExample.java

```
package ch12.sec07;

public class MathExample {
    public static void main(String[] args) {
        //큰 정수 또는 작은 정수 얻기
        double v1 = Math.ceil(5.3);
        double v2 = Math.floor(5.3);
        System.out.println("v1=" + v1);
        System.out.println("v2=" + v2);

        //큰값 또는 작은값 얻기
        long v3 = Math.max(3, 7);
        long v4 = Math.min(3, 7);
        System.out.println("v3=" + v3);
        System.out.println("v4=" + v4);

        //소수 이하 두 자리 얻기
        double value = 12.3456;
        double temp1 = value * 100;
        long temp2 = Math.round(temp1);
        double v5 = temp2 / 100.0;
        System.out.println("v5=" + v5);
    }
}
```

결과

```
v1=6.0
v2=5.0
v3=7
v4=3
v5=12.35
```

RandomExample.java

```java
package ch12.sec07;

import java.util.Arrays;
import java.util.Random;

public class RandomExample {
   public static void main(String[] args) {
      //선택번호
      int[] selectNumber = new int[6];
      Random random = new Random(3);  // 종자값 3
      System.out.print("선택번호: ");
      for(int i=0; i<6; i++) {
         selectNumber[i] = random.nextInt(45) + 1;   // 1~45 정수
         System.out.print(selectNumber[i] + " ");
      }
      System.out.println();

      //당첨번호
      int[] winningNumber = new int[6];
      random = new Random(5);     // 종자값 5. 다른 종자값, 다른 난수
      System.out.print("당첨번호: ");
      for(int i=0; i<6; i++) {
         winningNumber[i] = random.nextInt(45) + 1;   // 1~45 정수
         System.out.print(winningNumber[i] + " ");
      }
      System.out.println();

      //당첨여부
      Arrays.sort(selectNumber);
      Arrays.sort(winningNumber);
      boolean result = Arrays.equals(selectNumber, winningNumber);
      System.out.print("당첨여부: ");
      if(result) {
         System.out.println("1등에 당첨되셨습니다.");
      } else {
         System.out.println("당첨되지 않았습니다.");
      }
   }
}
```

12.8 날짜와 시간 클래스

- 컴퓨터의 날짜 및 시각을 읽을 수 있도록 `java.util` 패키지에서 `Date`와 `Calendar` 클래스 제공
- 날짜와 시간을 조작할 수 있도록 `java.time` 패키지에서 `LocalDateTime` 등의 패키지 제공

클래스	설명
`Date`	날짜 정보를 전달하기 위해 사용
`Calendar`	다양한 시간대별로 날짜와 시간을 얻을 때 사용

클래스	설명
`LocalDateTime`	날짜와 시간을 조작할 때 사용

12.9 형식 클래스

Format 클래스	설명
`DecimalFormat`	숫자를 형식화된 문자열로 변환
`SimpleDateFormat`	날짜를 형식화된 문자열로 변환

12.10 정규 표현식 클래스

- 문자열이 정해져 있는 형식으로 구성되어 있는지 검증할 때 사용
- `java.util.regex` 패키지의 `Pattern` 클래스의 `matches()` 메소드로 정규 표현식 검증 가능.

PatternExample.java

```
package ch12.sec10;

import java.util.regex.Pattern;

public class PatternExample {
   public static void main(String[] args) {
      String regExp = "(02|010)-\\d{3,4}-\\d{4}";
      String data = "010-123-4567";
      boolean result = Pattern.matches(regExp, data);
      if(result) {
         System.out.println("정규식과 일치합니다.");
      } else {
         System.out.println("정규식과 일치하지 않습니다.");
      }

      regExp = "\\w+@\\w+\\.\\w+(\\.\\w+)?";
      data = "angel@mycompanycom";
      result = Pattern.matches(regExp, data);
      if(result) {
         System.out.println("정규식과 일치합니다.");
      } else {
         System.out.println("정규식과 일치하지 않습니다.");
      }
   }
}
```

12.11 리플렉션

- 자바는 클래스와 인터페이스의 메타 정보를 `Class` 객체로 관리
- 메타 정보 : 패키지 정보, 타입 정보, 멤버 (생성자, 필드, 메소드) 정보 등

- 이러한 메타 정보를 프로그램에서 읽고 수정하는 행위를 **리플렉션(reflection)** 이라고 함.

12.12 어노테이션

- 코드에서 @로 작성되는 요소
- 클래스 또는 인터페이스를 컴파일하거나 실행할 때 어떻게 처리해야 할 것인지를 알려주는 설정 정보.
- 용도
    1. 컴파일 시 사용하는 정보 전달
    2. 빌드 툴이 코드를 자동으로 생성할 때 사용하는 정보 전달
    3. 실행 시 특정 기능을 처리할 때 사용하는 정보 전달