14. API

1) JDK 라이브러리

자바 환경에서 돌아가는 프로그램을 개발하는 데 필요한 도구들을 모아놓은 패키지이다. 기존 프로그램 작업을 하면서 자동적으로 실행할 수 있었던 명령어들은 자바를 실행할 때 기본으로 연동되는 인터페이스와 라이브러리 클래스들 덕분이었다. JDK에서 제공하는 라이브러리들은 대표적으로

- java.lang : 자바프로그램의 기본적인 기능을 제공. 명시적으로 지정하지 않아도 모든 자바 프로그램에 포함되는 패키지이다. Ex) java.lang.String나 java.lang.Exception은 모두 java.lang.을 생략 가능
- java.util : 유용한 유틸리티 클래스를 제공
- java.io : 입출력 기능을 제공하는 패키지
- java.awt : 그래픽 유저인터페이스(GUI)를 구축하기 위한 다양한 컴포넌트를 제공하는 패키지
- java.awt.event awt :컴포넌트들의 이벤트를 제어하는 패키지

2) API(Application Programming Interface)

자바 프로그램에서 누군가가 만들어 놓은 자주 사용하는 클래스 및 인터페이스의 모음이라고 할 수 있다. 자바에는 3,000여개의 API 클래스가 있으며 보통 오라클사에서 제공하는 API document 페이지를 이용하여 참고하여 사용한다. - https://docs.oracle.com/javase/8/docs/api/

3) String

String은 다른 기초 데이터 자료형(int, double, float ..)과는 다르게 객체 자료형 데이터로써 보통의 클래스들 처럼 객체를 만들어서 사용해야한다. 하지만 String은 예외로 아래처럼 생략하여 사용 가능하게 했다.

Ex)

```
1 String string = “HelloWorld”;
2
3 String string = new String(“HelloWorld~~”); // 이 두 경우를 모두 사용할수있다.
```

또한 String형의 변수를 정의할 때 String은 객체이기 때문에 해당 변수를 직접적으로 가르키지 않고 값의 위치를 가르키는 주소를 보여준다. 이 같은 특성 때문에 String변수를 정의할 때 기존 정의한 변수를 수정하려고

Ex)

```
String i = "HelloWorld";

i = "HellowWorld~" // 기존 i에 대입하지 않고 새로운 객체를 생성하여 대입
```

라고 수정했을 때 수정 전 i의 값에 변동된 값을 대입하지 않고 새로운 객체를 생성하여 대입한다.

4) String의 주요 기능들

String 클래스에서 사용하는 주요 기능들이다. 무수히 많지만 주로 사용되는 필수 명령어 몇 개만 예로 들었다.

Ex)

```
String str1 = "abcXabc";
String str2 = new String("ABCXabc");
String str3 = "       ja    va       ";

 System.out.println("1."+str1.concat(str2));           // 문자열 결합 | abcXabcABCXabc
 System.out.println("2."+str1.substring(3));             // 3번째부터 끝까지 출력 | Xabc
 System.out.println("3."+str1.substring(3,5));             //3번째부터 5번째앞까지 | Xa
 System.out.println("4."+str1.length());                              //글자 길이 | 7
 System.out.println("5."+str1.toUpperCase());                     // 대문자로 | ABCXABC
 System.out.println("6."+str1.toLowerCase());                     // 소문자로 | abcxabc
 System.out.println("7."+str1.charAt(3));                          // 3번째 문자 | X
 System.out.println("8."+str1.indexOf('b'));       // 첫번째 'b'가 나오는 인덱스 위치값 | 1
 System.out.println("9."+str1.indexOf('b', 3)); // 3번째부터 검색해서 첫번째 'b' 위치값 | 5
 System.out.println("10."+str1.indexOf("abc"));          // 첫번째 "abc"나오는 위치 | 0
 System.out.println("11."+str1.indexOf("abc",3));     // 3번째부터 검색해서 "abc"위치 | 4
 System.out.println("12."+str1.indexOf('z'));             //해당 값이 없으면 -1  | -1
 System.out.println("13."+str1.lastIndexOf('b'));                 //마지막 'b' 위치 | 5
 System.out.println("14."+str1.lastIndexOf('b',3));  //3번째부터 맨 마지막 'b'의 위치 | 1
 System.out.println("15."+str1.equals(str2));   // str1과 str2가 같은 문자열인지 | false
 System.out.println("16."+str1.equalsIgnoreCase(str2));  //대소문자 구분없이 비교 | true
 System.out.println("17."+str3.trim());                    // 앞뒤 공백 제거 | ja   va
 System.out.println("18."+str1.replace('a', '9'));        //'a'를 '9'로 수정 | 9bcX9bc
 System.out.println("19."+str1.replace("abc", "#"));     // "abc"을 "#"으로 수정 | #X#
 System.out.println("20."+str1.replaceAll("abc","Z"));          // "abc"을 "z" | ZXZ

String str = "안녕Hello";
 System.out.println(str.replaceAll("[a-zA-Z]", "*"));   //알파벳을 ""로 수정 | 안녕*****
 System.out.println(str.replaceAll("[가-힣]", "*"));    // 한글문자를 ""로 수정 | **Hello
```

5) String의 문제점

위에서 언급했듯이 String클래스는 객체자료형 데이터이기 때문에 어떤 변수의 수정이나 값의 변동이 있을 때 기존의 메모리를 수정하여 대입하지 않고 새로 만들어 대입한다. 이는 메모리를 과소비한다고 볼 수 있으며, 때문에 String형식의 무수히 많은 변화를 주는 로직은 메모리에 큰 비중을 차지하게 되고 프로그램이 느려진다. 이런 단점을 보완하고자 문자열 변수의 조작에 적합하게 만든 StringBuilder클래스와 StringBuffer클래스가 있다.

이 클래스들은 모두 객체 내부에 있는 버퍼(buffer, 데이터를 임시로 저장하는 메모리)에 문자열 내용을 저장해 두고 그 안에서 추가, 수정, 삭제 작업을 진행하여 기존 String처럼 새로운 객체를 생성하지 않고 기존 객체를 유지하며 작업한다.

1. StringBuilder, StringBuffer

두 가지 클래스는 모두 쓰임과 메소드가 같지만, StringBuffer는 여러곳에서 동시에 같은 문자열 인스턴스에 접근할 때 중복 점유를 막을 수 있는 장치가 되어 있다(동기화 처리). 때문에 StringBuilder에 비해 좀 더 무거우며. 따라서 특별한 이유가 없다면 StringBuilder를 사용하는 것이 일반적이라고 한다. StringBuilder가 좀 더 빠르다

Ex)

```
StringBuilder strBuilder = new StringBuilder("abc");
 //수정될때마다 객체를 새로 만들지않음 String 기존값에 추가
        System.out.println("해시코드 결과 :" +strBuilder.hashCode());
        //내용을 수정했을때 새로 생생되지않는것을 확인하기위한 해시값  // 1159190947
        System.out.println("strBuilder :" + strBuilder); //abc

        strBuilder.append("def");//기존의 "abd"에 "def"를 추가 객체 자체에 추가할당돼있음
        System.out.println("strBuilder :" + strBuilder); // abcdef

        strBuilder.insert(3, "AAA");//3번째에 "AAA"추가
        System.out.println("strBuilder :" + strBuilder);//abcAAAdef

        strBuilder.delete(3, 5);//3번쨰 부터 5번째 앞까지 삭제
        System.out.println("strBuilder :" + strBuilder); // abcAdef
        System.out.println("해시코드 결과 :" +strBuilder.hashCode());//1159190947

        strBuilder.deleteCharAt(3); //3번째 문자만 삭제 == delete(3,4)
        System.out.println("strBuilder :" + strBuilder.toString());// abcdef
        System.out.println("해시코드 결과 :" +strBuilder.hashCode());//1159190947

        System.out.println(strBuilder.capacity()); // 19
        /* 실제 데이터의 길이가 들어있는 length()와는 다른 capacity()는 현재 배열 사이즈
         * String과는 다르게 StringBuilder는 사이즈를 여유있게 잡아둠
         * append()등 문자열 조정할때마다 배열 사이즈가 자동으로 조절됨
         */
```

위 해시값 처럼 StringBuilder, StringBuffe는 새로 생성하지 않고 기존 객체를 수정하면서 작동한다.

2. 개발 테스트에 많이 쓰이는 System.currentTimeMillis()

String과 StringBuilder, StringBuffer의 작업속도를 차이를 보여주기 위해 10만번의 로직을 수행 후 currentTimeMillis(1970년도부터 현재까지의 밀리세컨 - 1/1,000초)함수를 통해 시간을 측정 보면

Ex)

```
    //String 변경 (10만번)
    String str = "A";
    long startTime = System.currentTimeMillis(); // 로직 시작시점의 밀리세컨
    for (int i = 0; i < 100000; i++) {
        str = str + "a";
    }
    long endTime = System.currentTimeMillis(); //로직 끝나는 시점의 밀리세컨
    System.out.println((endTime - startTime);   // 1512

    // StringBuffer 변경(10만번)
    StringBuffer strBuf = new StringBuffer("A");
    startTime = System.currentTimeMillis();
    for(int i=0; i<100000; i++) {
        strBuf.append("a");
    }
    endTime = System.currentTimeMillis();
    System.out.println(endTime - startTime);    // 5

    // StringBuilder 변경(10만번)
    StringBuilder strBuld = new StringBuilder("A");
    startTime = System.currentTimeMillis();
    for(int i=0; i<100000; i++) {
        strBuld.append("a");
    }
    endTime = System.currentTimeMillis();
    System.out.println(endTime - startTime);    // 3
    }
```

컴퓨터 별로 성능 차이로 인한 작동시간 차이가 있겠지만 해당 컴퓨터는 약1.5의 차이를 보였다.

3. StringTokenizer

StringTokenizer는 하나의 String 문자열을 문자별로 분리(토큰화) 해주는 클래스이다.

```
String str1 = "박보검 설현 수지 고소영 장동건";
String str2 = "2022/03/28";
StringTokenizer tokenizer1 = new StringTokenizer(str1);  // 토큰화 하고싶은 변수 입력
//클래스이기 때문에 객체를 생성하여 사용해야하며
//기본적으로 토큰화 하고싶은 변수를 넣을때 변수명 뒤에 아무것도 입력하지않으면 space기준으로 문자열을 분할
System.out.println("tokenizer1의 토큰 갯수 " + tokenizer1.countTokens());

while(tokenizer1.hasMoreTokens()) {  //hasMoreTokens() 해당 객체에 토큰값이 있는지 불린으로 리턴
System.out.println(tokenizer1.nextToken()); //nextToken() 매개변수의 토큰을 출력하는 함수
} /*박보검
   *설현
   *수지
   *고소영
   *장동건
   */
StringTokenizer tokenizer2 = new StringTokenizer(str2, "/");
 // "/"기준으로 문자열 분할 "2022" String으로 분할
System.out.println("tokenizer2의 토큰 갯수 : " + tokenizer2.countTokens() );

while(tokenizer2.hasMoreTokens()) {
    System.out.println(tokenizer2.nextToken());
}/*2022
  *03
  *28
  */
```