제 6 장
유용한 참조타입과 배열
1부-String

참조타입

- 참조(reference)타입, 객체(object)타입의 변수
  - 정보가 저장된 (위치를 가리키는) 주소를 저장하고 있는 유형의 변수 (cf. 기본 데이터 타입)
  - 모든 객체는 참조타입의 변수로써 할당된다
- 자주 사용하는 참조 타입의 객체
  - **String**
  - **Number**
    - wrapper 클래스 : 기본 데이터 타입을 참조타입의 객체로 변환
    - 자동박싱(Autoboxing)과 언박싱(Unboxing)
  - **Math** : 각종 연산 메소드를 포함하고 있는 클래스
  - **배열** 객체
  - **Enum** 객체

String 타입

- 문자들의 순차적 나열로서, 참조타입으로 정의되는 객체
- **String** 클래스
  - 매우 다양한 메소드를 포함하고 있다
  - 다양한 방법으로 스트링을 생성할 수 있도록 13개의 생성자 메소드를 지원
  - 교재 p.101의 [표 6.1] ~ [표 6.5]를 참고
    - 스트링 내의 문자 조작
    - 스트링의 분할, 연결, 탐색 … 등
- **스트링의 생성**
  - 가장 직접적이면서 기본적인 참조타입의 스트링 생성
    - **String greeting = "Hello World !";**
  - 스트링은 한번 생성되면 변경이 불가능

- 스트링 생성 예
  - char[] helloArray = { 'h', 'e', 'l', 'l', 'o', '.' };
  - String helloString = **new** String(helloArray);
  - System.out.println(helloString);
- String.format()

```
System.out.printf("The value of the float " + "variable is %f, while " +  "the value of the "
+ "integer variable is %d, " + "and the string is %s", floatVar, intVar, stringVar);
```

```
String fs;        // 스트링 변수의 선언
// 형식화된 스트링의 생성
fs = String.format("The value of the float " + "variable is %f, while " +  "the value of the
" + "integer variable is %d, " + " and the string is %s", floatVar, intVar, stringVar);
System.out.println(fs);
```

스트링 길이 정보: length()

- accessor 메소드

```
String palindrome = "Dot saw I was Tod";
int len = palindrome.length();              // 17
```

```
public class StringTest {
    public static void main(String[] args) {
        String palindrome = "Dot saw I was Tod";
        int len = palindrome.length();
        char[] tempCharArray = new char[len];          // 문자 배열의 생성
        char[] charArray = new char[len];                              // 문자 배열의 생성
        // 원본 스트링의 임시 보관
        for (int i = 0; i < len; i++) {
            tempCharArray[i] = palindrome.charAt(i);
        }
        // 문자 배열의 역순화
        for (int j = 0; j < len; j++) {
            charArray[j] = tempCharArray[len - 1 - j];
        }
        String reversePalindrome = new String(charArray);
        System.out.println(reversePalindrome);
    }
}
```

- **스트링의 연결: String.concat()**
  - 2개의 스트링 객체 string1과 string2를 연결
    - String1.concat(string2)
  - 2개의 스트링 리터럴을 연결
    - "My name is ".concat("Won Ho Chung");
  - println() 메소드에서는 문자연결 연산자인 '+'를 많이 사용
    - String string1 = "saw I was ";
      System.out.println("Dot " + string1 + "Tod");
    - 자바는 2줄 이상의 스트링 리터럴을 허용하지 않음
      - String quote = "Now is the time for all good " +
        "men to come to the aid of their country.";

문자처리

- 스트링 내의 문자 검색, 서브스트링 검색, 대소문자 변환 등...
- String.charAt()
  - 스트링 내의 특정 인덱스 위치의 문자 액세스

```
String anotherPalindrome = "Niagara. O roar again!";
char aChar = anotherPalindrome.charAt(9);
```

- String.substring()
  - 스트링 내의 일부를 구성하는 서브스트링 액세스
  - 교재 p.101의 [표 6.1] 참조

```
String anotherPalindrome = "Niagara. O roar again!";
String roar = anotherPalindrome.substring(11, 15);
```

- 기타 서브스트링 처리 메소드 : 교재 p.102의 [표 6.2]
- 기타 문자 및 서브스트링 검색 메소드 : 교재 p.103의 [표 6.3]
- 기타 문자 및 서브스트링 대체 메소드 : 교재 p.104의 [표 6.4]

[표 6.1] 메소드 substring()

메소드	설명
String **substring**(int beginIndex, int endIndex)	주어진 스트링의 beginIndex 위치의 문자에서 시작해서 (endIndex-1) 위치까지의 문자들로 구성된 서브스트링인 새로운 스트링을 리턴.
String **substring**(int beginIndex)	주어진 스트링의 beginIndex 위치의 문자로 시작하여 주어진 스트링의 끝으로 구성된 서브스트링인 새로운 스트링을 리턴

[표 6.2] 유용한 스트링 처리 메소드

메소드	설명
String[] split(String regex) String[] split(String regex, int limit)	String 타입의 정규표현식(regex)으로 명시된 것과 일치하는 서브스트링을 찾아서, 이들을 regex에 따라 분리하여 스트링 배열로 리턴. 매개변수 limit는 리턴되는 배열의 최대 크기를 명시한다
CharSequence subSequence(int beginIndex, int endIndex)	beginIndex 위치의 문자부터 (endIndex-1) 위치의 문자까지로 구성되는 문자 시퀀스를 리턴
String trim()	주어진 스트링의 앞과 뒤 부분의 공백을 제거한 스트링의 복사본을 리턴
String toLowerCase() String toUpperCase()	주어진 스트링을 구성하는 문자들을 소문자 혹은 대문자로 변환하여 리턴. 변환이 필요없으면 원래 스트링을 리턴

[표 6.3] 각종 검색 메소드

메소드	설명
int indexOf(int ch) int lastIndexOf(int ch)	스트링 내에서 명시된 문자 ch가 나타나는 첫 번째(마지막) 인덱스를 리턴
int indexOf(int ch, int fromIndex) int lastIndexOf(int ch, int fromIndex)	스트링 내에서 명시된 문자 ch가 나타나는 첫 번째(마지막) 인덱스를 리턴. 탐색은 fromIndex로 명시된 위치부터 시작하여 순방향(역방향)으로 수행
int indexOf(String str) int lastIndexOf(String str)	스트링 내에서 명시된 서브스트링 str이 나타나는 첫 번째(마지막) 인덱스를 리턴
int indexOf(String str, int fromIndex) int lastIndexOf(String str, int fromIndex)	스트링 내에서 명시된 서브스트링 str이 나타나는 첫 번째(마지막) 인덱스를 리턴. 탐색은 fromIndex로 명시된 위치부터 시작하여 순방향(역방향)으로 수행
boolean contains(**CharSequence** s)	스트링이 명시된 문자 시퀀스 s를 포함하고 있으면 true를 리턴

[표 6.4] 스트링 대체 및 조작 메소드

메소드	설명
String replace(char oldChar, char newChar)	스트링 내의 모든 oldChar를 newChar로 대체한 새로운 스트링을 리턴
String replace(CharSequence target, CharSequence replacement)	스트링 내의 target으로 명시된 문자 시퀀스와 일치하는 각 서브스트링을 replacement로 명시된 문자 시퀀스로 대체한 새로운 스트링을 리턴
String replaceAll(String regex, String replacement)	스트링 내의 정규표현식(regex)과 일치하는 모든 서브스트링을 명시된 replacement 스트링으로 대체한 새로운 스트링을 리턴
String replaceFirst(String regex, String replacemen)	스트링 내의 정규표현식(regex)과 일치하는 첫 번째 서브스트링을 명시된 replacement 스트링으로 대체한 새로운 스트링을 리턴

파일 이름의 각 부분을 구분하기 위해서 lastIndexOf()와 substring() 메소드를 사용하는 예.
여기서 파일명으로 주어지는 인수는 절대 경로명과 확장자를 가지고 있다는 것을 가정한다.

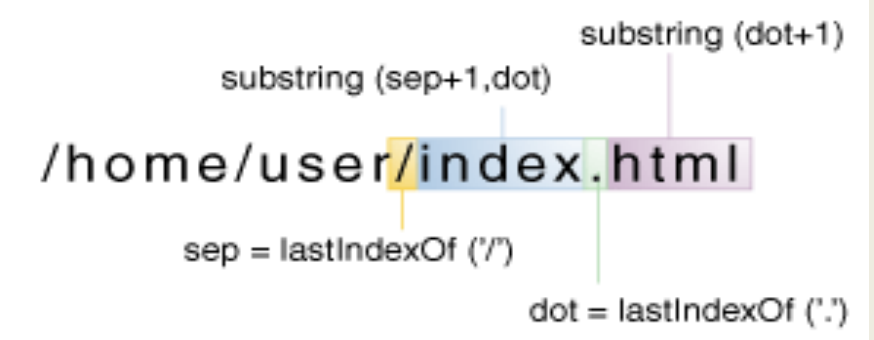
```
public class Filename {
    private String fullPath;
    private char pathSeparator, extensionSeparator;
    public Filename(String str, char sep, char ext) {
        fullPath = str;
        pathSeparator = sep;
        extensionSeparator = ext;
    }
    //파일 확장자
    public String extension() {
        int dot = fullPath.lastIndexOf(extensionSeparator);
        return fullPath.substring(dot + 1);   // dot+1 부터 스트링 끝
    }
    //파일 이름
    public String filename() {
        int dot = fullPath.lastIndexOf(extensionSeparator);
        int sep = fullPath.lastIndexOf(pathSeparator);
        return fullPath.substring(sep + 1, dot);
    }
    public String path() {
        int sep = fullPath.lastIndexOf(pathSeparator);
        return fullPath.substring(0, sep);
    }
}
```

```
public class FilenameTest {
    public static void main(String[] args) {
        final String FPATH = "/home/user/index.html";
        Filename myHomePage = new Filename(FPATH, '/', '.');
        System.out.println("Extension = " + myHomePage.extension());
        System.out.println("Filename = " + myHomePage.filename());
        System.out.println("Path = " + myHomePage.path());
    }
}
```

Extension = html

Filename = index

Path = /home/user



스트링 비교

메소드	설명
boolean endsWith(String suffix) boolean startsWith(String prefix)	어떤 스트링이 인수로 주어진 스트링 suffix/prefix로 끝나거나/시작하면 true 리턴
boolean startsWith(String prefix, int offset)	어떤 스트링의 offset 번째가 인수로 주어진 스트링 prefix로 시작하면 true 리턴
int **compareTo**(String anotherString)	어떤 스트링을 인수로 주어진 anotherString과 사전적으로 비교하여(소문자 대문자 구별) 1) 어떤 스트링이 앞에 나오면 양의 정수를 2) 어떤 스트링과 일치하면 0을 3) 어떤 스트링이 뒤에 나오면 음의 정수를 리턴
int compareToIgnoreCase(String str)	소문자 혹은 대문자의 경우를 구별하지 않는다는 점만 제외하고 compareTo(String)과 동일
boolean **equals**(Object anObject)	주어진 인수가 어떤 객체와 문자들의 순서가 동일한 String 객체이면 true를 리턴하며, 그 역의 경우에도 true를 리턴(소문자 혹은 대문자 구별) 매개변수가 Object 타입으로 주어짐을 주의
boolean equalsIgnoreCase(String anotherString)	소문자 혹은 대문자의 경우를 구별하지 않는다는 점만 제외하고 equals(Object)와 동일
boolean **regionMatches** (int toffset, String other, int ooffset,int len)	어떤 스트링의 명시된 영역이 인수로 주어진 스트링의 명시된 영역이 서로 일치하는지 테스트 . 비교 영역은, 어떤 스트링의 시작은 toffset 번째부터, 인수로 주어진 스트링 other의 시작은 ooffset 번째이며 비교 길이는 len으로 명시
boolean **regionMatches** (boolean ignoreCase, int toffset, String other, int ooffset, int len)	boolean 타입의 매개변수 ifnoreCase를 통해 대소문자를 구별하는 지를 테스트한다는 것을 제외하고는 **regionMatches**(int, String, int, int)와 동일하며, ignoreCase가 true이면 대소문자 구별하지 않는다
boolean matches(String regex)	어떤 스트링이 인수로 주어지는 스트링 타입의 regular expression과 일치하는지를 테스트하여 일치하면 true 리턴

스트링 내의 서브스트링을 탐색하기 위해서 regionMatches() 메소드를 사용하고 있는 예

```
public class RegionMatchesTest {
    public static void main(String[] args) {
        String searchMe = "Green Eggs and Ham";
        String findMe = "Eggs";
        int searchMeLength = searchMe.length();
        int findMeLength = findMe.length();
        boolean found = false;
        for (int i = 0; i <= (searchMeLength - findMeLength); i++) {
            if (searchMe.regionMatches(i, findMe, 0, findMeLength)) {
                found = true;
                System.out.println(searchMe.substring(i, i + findMeLength));
                break;
            }
        }
        if (!found)
            System.out.println("No match found.");
    }
}
```

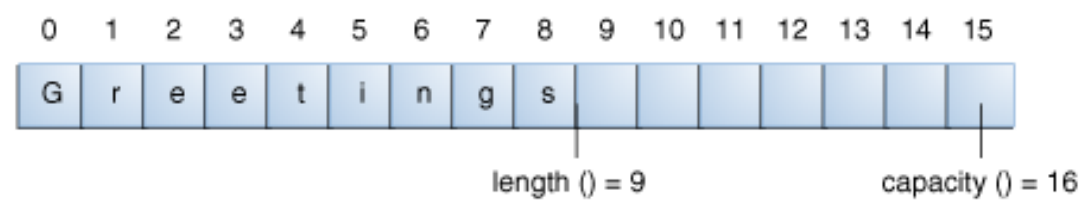
Eggs

StringBuilder 클래스

- StringBuilder는 객체가 수정될 수 있다는 점을 제외하고는 String 객체와 동일
- 내부에서 스트링 처리를 위해 버퍼를 두고 있음
- 객체를 버퍼를 사용하는 가변 길이의 배열(variable-length array)로 취급
- 해당 스트링의 길이와 내용은 메소드 구동을 통하여 변경 가능
- 코드를 단순하게, 혹은 성능 면에서 우수하지 않으면, 가능한 String 클래스을 사용하는 것이 바람직하다
  - 빈번하게 스트링들을 서로 연결해야 할 필요가 있는 경우, StringBuilder 객체를 사용하는 것이 효율적일 수 있다
  - 왜냐하면 스트링들의 연결로 인해 많은 garbage들이 생성되어 수행 속도에 영향을 줄 수 있기 때문이다

- 메소드 length와 capacity
  - length() : 객체 내에 있는 스트링의 길이를 리턴
  - capacity() : 수용할 수 있는 문자 공간의 크기(default값 = 16)
    - Capacity값 >= length값
    - StringBuilder 객체에 추가되는 만큼을 수용할 수 있도록, 필요한 만큼 자동으로 확장된다

```
// creates empty builder, capacity 16
StringBuilder sb = new StringBuilder();          // 디폴트 capacity는 16
// adds 9 character string at beginning
sb.append("Greetings");
```



  - 기타 유용한 메소드:
    - 교재 p.110, 111의 [표 6.7]과 [표 6.8]을 참조