Chapter 11. 메소드 오버로딩과 String 클래스

11-1. 메소드 오버로딩(Overloading)

매개변수 형(type)이 다르거나 개수가 다르거나

나 정말 JAVA를 공부한 적 없다고

- 메소드 오버로딩이란 동일한 이름의 메소드를 둘 이상 동시에 정의하는 것을 뜻한다.
- 메소드의 매개변수 선언(개수 또는 자료형)이 다르면 메소드 오버로딩 성립
- 오버로딩 된 메소드는 호출 시 전달하는 인자를 통해서 구분된다.

```
class AAA
{
    void isYourFunc(int n) { . . . }
    void isYourFunc(int n1, int n2) { . . . }
    void isYourFunc(int n1, double n2) { . . . }
    . . . .
}
```

오버로딩 된 메소드

```
AAA inst = new AAA();
inst.isYourFunc(10);
inst.isYourFunc(10, 20);
inst.isYourFunc(12, 3.15);
```

전달되는 인자의 유형을 통해서 호출되는 함수가 결정!

요런! 아주 기막히게 애매한 상황!

형변환의 규칙까지 적용해야만 메소드가 구분되는 기막히게 애매한 상황은 만들지 말자!

```
class AAA
{
    void isYourFunc(int n) { . . . }
    void isYourFunc(int n1, int n2) { . . . }
    void isYourFunc(int n1, double n2) { . . . }
     . . . .
}
```

오버로딩 된 메소드

```
AAA inst = new AAA();
inst.isYourFunc(10, 'a');
```

무엇이 호출되겠는가? 문자 'a'는 int형으로도, double형으로도 변환이 가능하다!

결론적으로, 형변환 규칙을 적용하되 가장 가까운 위치의 자료형으로 변환이 이뤄진다. 따라서 is...(int n1, int n2)가 호출된다.

생성자도 오버로딩의 대상이 됩니다.

생성자의 오버로딩은 하나의 클래스를 기반으로 다양한 형태의 인스턴스 생성을 가능하게 한다.

```
class Person
{
    private int perID;
    private int milID;
    public Person(int pID, int mID)
    {
        perID=pID;
        milID=mID;
    }
    public Person(int pID)
    {
        perID=pID;
        milID=0;
    }
    public void showInfo()
    {
        System.out.println("민번 : "+perID);
        if(milID!=0)
            System.out.println("군번 : "+milID+'\n');
        else
            System.out.println("군과 관계 없음 \n");
    }
}
```

군 필자를 위한 생성자

군 미필자를 위한 생성자

```
public static void main(String[] args)
{
    Person man=new Person(950123, 880102);
    Person woman=new Person(941125);
    man.showInfo();
    woman.showInfo();
}
```

군을 제대한 남성과 여성을 의미하는 인스턴스의 생성이 가능하다!

키워드 this를 이용한 다른 생성자의 호출

- 키워드 this를 이용하면 생성자 내에서 다른 생성자를 호출할 수 있다.
- 이는 생성자의 추가 정의에 대한 편의를 제공한다.
- 생성자마다 중복되는 초기화 과정의 중복을 피할 수 있다.

```
class Person
{
    private int perID;
    private int milID;
    private int birthYear;
    private int birthMonth;
    private int birthDay;

    public Person(int perID, int milID, int bYear, int bMonth, int bDay)
    {
        this.perID=perID;
        this.milID=milID;
        birthYear=bYear;
        birthMonth=bMonth;
        birthDay=bDay;
    }
    public Person(int pID, int bYear, int bMonth, int bDay)
    {
        this(pID, 0, bYear, bMonth, bDay);
    }
```

생성자의 재호출을 위한 키워드 this가 존재하지 않았다고 생각해 보자. 이 클래스의 생성자 정의에 어떠한 변화가 있어야 하는가?

인자로 pID, 0, bYear, bMonth, bDay를 전달받는 생성자의 호출문장

11-2. String 클래스

String 클래스의 인스턴스 생성

- JAVA는 큰 따옴표로 묶여서 표현되는 문자열을 모두 인스턴스화 한다.
- 문자열은 String 이라는 이름의 클래스로 표현된다.

```
String str1 = "String Instance";
String str2 = "My String";
```

두 개의 String 인스턴스 생성,
그리고 참조변수 str1과 str2로 참조

```
System.out.println("Hello JAVA!");
System.out.println("My Coffee");
```

println 메소드의 매개변수형이 String이
기 때문에 이러한 문장의 구성이 가능하다.

```
class StringInstance
{
    public static void main(String[] args)
    {
        java.lang.String str="My name is Sunny";

        int strLen1=str.length();
        System.out.println("길이 1 : "+strLen1);

        int strLen2="한글의 길이는 어떻게?".length();
        System.out.println("길이 2 : "+strLen2);
    }
}
```

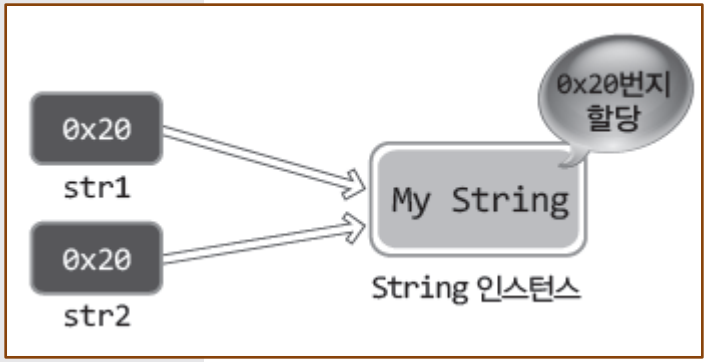
문자열의 선언은 인스턴스의 생성으로
이어짐을 보이는 문장

String 인스턴스는 상수 형태의 인스턴스이다.

- String 인스턴스에 저장된 문자열의 내용은 변경이 불가능하다.
- 이는 동일한 문자열의 인스턴스를 하나만 생성해서 공유하기 위함이다.

```
public static void main(String[] args)
{
    String str1="My String";
    String str2="My String";
    String str3="Your String";

    if(str1==str2)
        System.out.println("동일 인스턴스 참조");
    else
        System.out.println("다른 인스턴스 참조");
```

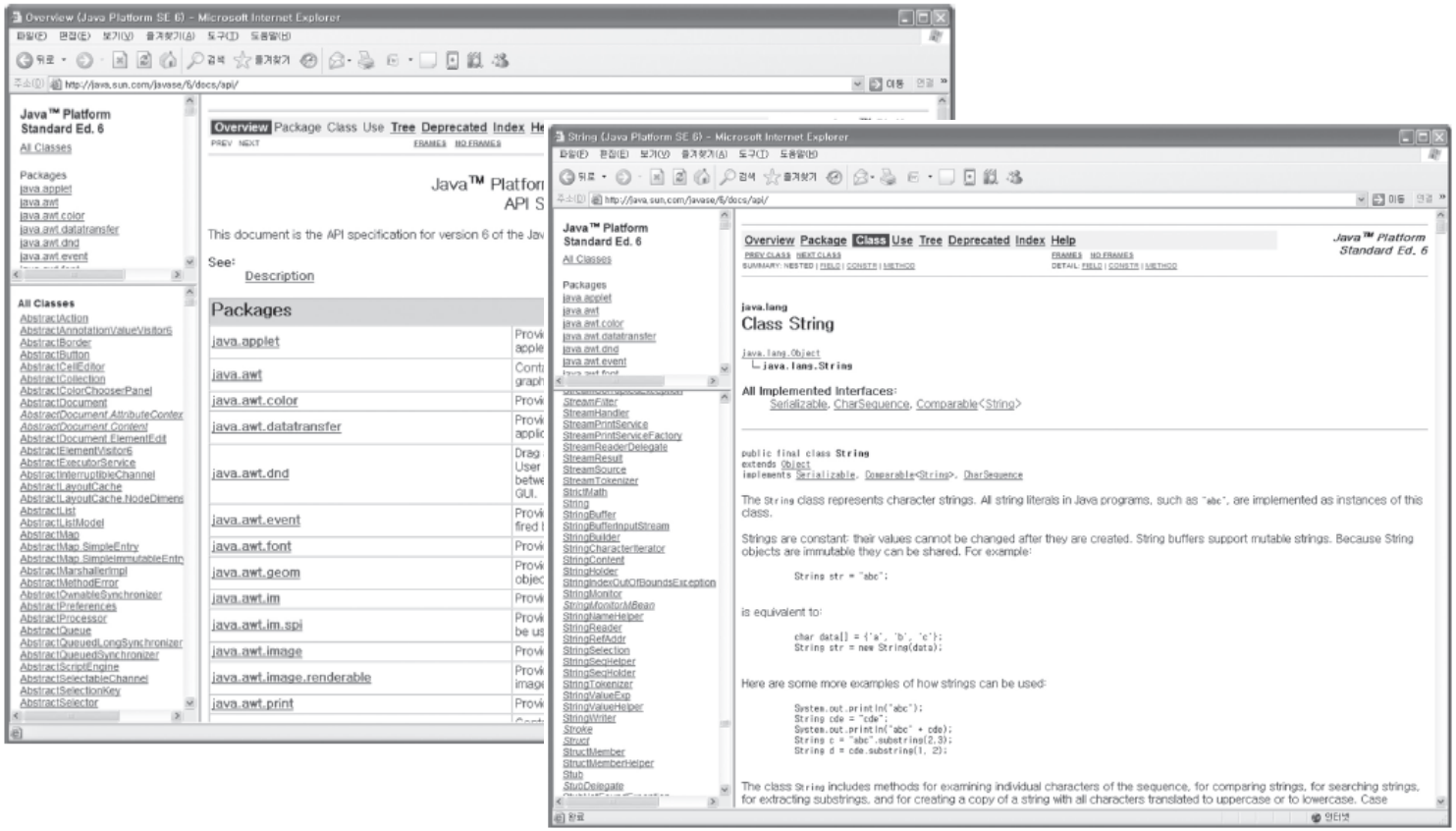


String 인스턴스의 문자열 변경이 불가능하기 때문에 둘 이상의 참조변수가 동시에 참조를 해도 문제가 발생하지 않는다!

11-3. API Document의 참조를 통한 String 클래스의 인스턴스 메소드 관찰

■ 생성자를 뺀 메소드의 수만 50개가 넘습니다.

- 책에서 메소드의 기능을 찾는 습관은 조금씩 버려야 한다.
- API 문서를 볼 줄 모르는 자바 개발자는 있을 수 없다.
- API 문서를 참조하지 않고 개발하는 자바 개발자도 있을 수 없다.



String 클래스가 제공하는 유용한 메소드들

- 문자열의 길이 반환 — public int length()
- 두 문자열의 결합 — public String concat(String str)
- 두 문자열의 비교 — public int compareTo(String anotherString)

위 메소드의 사용방법만이라도 API 문서에서 확인해 보자!

```
class StringMethod
{
    public static void main(String[] args)
    {
        String str1="Smart";
        String str2=" and ";
        String str3="Simple";
        String str4=str1.concat(str2).concat(str3);

        System.out.println(str4);
        System.out.println("문자열 길이 : "+str4.length());

        if(str1.compareTo(str3)<0)
            System.out.println("str1이 앞선다");
        else
            System.out.println("str3이 앞선다");
    }
}
```

실행결과

```
Smart and Simple
문자열 길이 : 16
str3이 앞선다
```

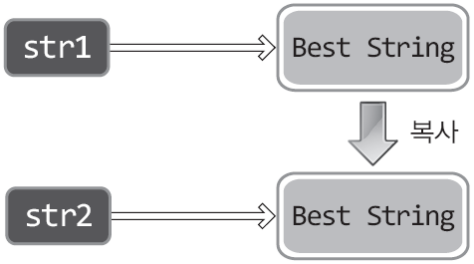
자바에서의 문자열 복사!

나는 정말 JAVA를 그만할 때 됐다고

- 자바에서는 문자열을 상수의 형태로 관리하고, 또 동일한 유형의 문자열을 둘 이상 유지하지 않으므로 문자열의 복사라는 표현이 흔하지 않다.
- 무엇보다도 자바에서는 문자열을 복사가 필요 없다.

그러나 원하는 것이 인스턴스를 새로 생성해서 문자열의 내용을 그대로 복사하는 것이라면 다음과 같이 코드를 구성하면 된다.

```
String str1="Best String";
String str2=new String(str1);
```




```
public String(String original)
```

새로운 문자열 인스턴스의 생성에 사용되는 생성자

```
class StringCopy
{
    public static void main(String[] args)
    {
        String str1="Lemon";
        String str2="Lemon";
        String str3=new String(str2);

        if(str1==str2)
            System.out.println("  실행  ");
        else
            System.out.println("  .  .  .  ");

        if(str2==str3)
            System.out.println("  .  .  .  ");
        else
            System.out.println("  실행  ");
    }
}
```

비교 연산자는 참조 값 비교!

■ + 연산과 += 연산의 진실


```
public static void main(String[] args)
{
    String str1="Lemon"+"ade";
    String str2="Lemon"+'A';
    String str3="Lemon"+3;
    String str4=1+"Lemon"+2;
    Str4+='!';

    System.out.println(str1);
    System.out.println(str2);
    System.out.println(str3);
    System.out.println(str4);
}
```

String str1="Lemon".concat("ade");
String str2="Lemon".concat(String.valueOf('A'));
String str3="Lemon".concat(String.valueOf(3));

위 예제의 str4의 선언이 다음과 같이 처리된다면?

String str4=Strimg.valueOf(1).concat("Lemon").concat(String.valueOf(2));

아무리 많은 + 연산을 취하더라도 딱 두 개의 인스턴스만 생성된다. StringBuilder 클래스의 도움으로...

11-4. StringBuilder & StringBuffer 클래스

StringBuilder

- StringBuilder는 문자열의 저장 및 변경을 위한 메모리 공간을 지니는 클래스
- 문자열 데이터의 추가를 위한 append와 삽입을 위한 insert 메소드 제공

```
class BuilderString
{
    public static void main(String[] args)
    {
        StringBuilder strBuf=new StringBuilder("AB");  buf: AB
        strBuf.append(25);  buf: AB25
        strBuf.append('Y').append(true);  buf: AB25Ytrue
        System.out.println(strBuf);

        strBuf.insert(2, false); buf: ABfalse25Ytrue
        strBuf.insert(strBuf.length(), 'Z'); buf: ABfalse25YtrueZ
        System.out.println(strBuf);
    }
}
```

실행결과

```
AB25Ytrue
ABfalse25YtrueZ
```

연속해서 함수호출이 가능한 이유는
append 메소드가 strBuf의 참조 값을 반환하기 때문이다.

참조를 반환하는 메소드

- this의 반환은 인스턴스 자신의 참조 값 반환을 의미한다.
- 그리고 이렇게 반환되는 참조 값을 대상으로 연이은 함수호출이 가능하다.

```
class SimpleAdder
{
    private int num;
    public SimpleAdder() {num=0;}
    public SimpleAdder add(int num)
    {
        this.num+=num;
        return this;
    }
    public void showResult()
    {
        System.out.println("add result : "+num);
    }
}
```

```
public static void main(String[] args)
{
    SimpleAdder adder=new SimpleAdder();
    adder.add(1).add(3).add(5).showResult();
}
```

실행결과

```
add result : 9
```

add 함수는 adder의 참조 값을 반환한다.

StringBuilder의 버퍼와 문자열 조합

- 추가되는 데이터 크기에 따라서 버퍼의 크기가 자동으로 확장된다.
- 생성자를 통해서 초기 버퍼의 크기를 지정할 수 있다.

- `public StringBuilder()` 기본 16개의 문자저장 버퍼 생성
- `public StringBuilder(int capacity)` capacity개의 문자저장 버퍼 생성
- `public StringBuilder(String str)` str.length()+16 크기의 버퍼 생성

문자열의 복잡한 조합의 과정에서는 StringBuilder의 인스턴스가 활용된다.
때문에 추가로 생성되는 인스턴스의 수는 최대 두 개이다!

```
String str4=1+"Lemon"+2;
```

```
new StringBuilder().append(1).append("Lemon").append(2).toString();
```

StringBuilder 인스턴스의 생성에서 한 개

toString 메소드의 호출에 의해서 한 개

StringBuffer 클래스

- StringBuffer 클래스와 StringBuilder 클래스의 공통점 세 가지
  - 메소드의 수(생성자 포함)
  - 메소드의 기능
  - 메소드의 이름과 매개변수형
- StringBuffer 클래스와 StringBuilder 클래스의 차이점
  - StringBuffer는 쓰레드에 안전, StringBuilder는 쓰레드에 불안전

예제 StringBuilder.java에서 StringBuilder 클래스를 StringBuffer 클래스로 바꿔도 컴파일 및 실행이 된다.

■ 단계별 프로젝트의 소개

• 하나의 주제를 단계별로 완성해 나가는 형태이다.

• 자바의 실력향상에 매우 직접적인 도움이 된다.

• 본인의 힘으로 직접 진행하는 것이 가장 좋으나, 부담스러운 경우에는 결과물을 참조해가면서 진행하는 것도 괜찮다.