Java chapter 7

Object 클래스
String 클래스
String 클래스의 메서드들

7-1. Object 클래스

아무 클래스를 만들고, 그 클래스의 객체 생성 후 .을 찍으면 객체로부터 사용 가능한 변수와 메서드가 뜬다.

그런데, 잘 보면 우리가 클래스에 만들지 않은 메서드가 함께 뜨는 것을 확인할 수 있다.

```
class Tv {
  public void turnOn(){
    System.out.println("Tv 전원을 켜다 " );
  }
}
```

```
public static void main(String[] args){
  Tv tv = new Tv();
  //객체명 뒤에 .을 작성하면 작성하지 않은 내용도 나온다..
  tv.
}
```

만들지 않은 메서드가 뜬다는 것은 그 메서드를 다른 클래스에서 상속받았기 때문이다.

즉, 우리가 만든 클래스는 어떤 클래스를 상속하고 있다. 그 클래스가 바로 Object클래스다.

- 클래스가 어떠한 클래스도 상속하고 있지 않으면 자동으로 Object 클래스를 상속한다.
- Object 클래스는 자바에서 모든 클래스의 최상위 부모 클래스다(단군 할아버지 같은 느낌)
- 어떠한 클래스도 상속 받지 않는 클래스가 Object클래스를 상속받는다고 했지만, 생각해보면 모든 클래스가 Object 클래스를 상속하고 있다.
- 결국 모든 클래스는 Obejct 클래스를 상속하고 있기 때문에, 모든 클래스에서 Object 클래스에서 정의한 메서드를 사용할 수 있다.
- Object 클래스가 모든 클래스의 부모 클래스이기 때문에, 모든 클래스 자료형의 객체는 Object 클래스로 받을 수 있다.

Object클래스의 toString() 메서드

toString() 메서드의 리턴 타입은 String이고, 매개변수는 없다.

toString() 메서드는 클래스 자료형을 문자열로 표현할 때 사용하는 메서드다.

클래스 자료형은 개발자가 직접 만들 수 있다. 우리가 만드는 클래스 자료형도 toString()메서드를 사용하면 알아서 문자열로 표현해줄까?

결론은 Object 클래스의 toString() 메서드는 우리가 오버라이딩을 통해 원하는 내용으로 문자열을 만들어 사용하라는 의도에서 만들어졌다.

클래스 자료형의 정보를 문자열로 표현한다는 것은 현재 객체가 가진 모든 맴버변수의 값을 문자열로 표현하는 것이 일반적이다.

@override

자바에는 @ 기호가 붙은 문법이 존재한다. @는 annotation(어노테이션)이라 부른다.

어노테이션은 사전적 의미로 '주석 ' 이라는 의미이며, 개발자의 실수로 일어나는 논리적 오류를 막거나, 어노테이션 별로 특정한 기능을 수행한다.

override 어노테이션을 사용한 메서드는 메서드가 오버라이딩 문법과 일치하지 않으면 개발자에서 알려주는 역할을 한다.

오버라이드 문법이 맞는지만 체크해주기 때문에, 다른 이유에서 발생하는 오류는 알려주지 않는다.

Object클래스의 equals() 메서드

equals() 메서드는 두 객체가 같은 객체인지 판별하는 기능을 한다.

리턴 타입은 boolean이며, 매개변수로 Object 자료형을 갖는다.

매개변수로 Object 자료형을 갖는다는 것은 모든 자료형을 매개변수의 인자로 받을 수 있다는 말이다.

모든 자료형을 인자로 전달받을 수 있게 메서드를 설계한 이유는, 어떤 데이터가 들어와도 비교가 가능하게 하기 위함이다.

같다는 의미는 무엇일까? 숫자와 문자가 같은지 다른지 판단하는 기준은 상식적으로 알고 있다.

그럼 우리가 만드는 클래스의 객체가 같다는 것의 기준은 무엇인가?(두 Person 객체가 같다는 기준은? 두 Student 객체가 같다는 것은?)

우리가 만드는 클래스의 객체가 같은지 다른지 판단하는 기준은 매번 달라진다.

즉, equals() 메서드는 개발자가 메서드를 오버라이드해서 같다는 정의를 재정의해서 사용하기 위한 용도로 만들어졌다.

* Object 클래스에 선언된 equals() 메서드를 오버라이드하지 않고 사용하면 두 객체의 주소가 같은지 비교하여 같다/다르다를 판단한다.

다음은 우리가 만드는 클래스에서 equals() 메서드를 오버라이딩한 예시이다.

```
class Student{
    int stuNum; //학번
    String name;

    public Student(int stuNum, String name){
        this.stuNum = stuNum;
        this.name = name;
    }

    @override
    public boolean equalse(Object obj){
        return stuNum == (Student)obj.stuNum;
    }
}
```

```
public static void main(String[] args){
    Student s1 = new Student(1, "김자바");
    Student s2 = new Student(2, "김자바");
    Student s3 = new Student(1, "박자바");

    if(s1.equals(s2))
        System.out.println("두 객체는 같음");
    else
        System.out.println("두 객체는 다름"); //실행

    if(s1.equals(s3))
        System.out.println("두 객체는 같음"); //실행
    else
        System.out.println("두 객체는 다름");
}
```

* String 클래스의 equals() 메서드는 Object 클래스의 equals() 메서드를 오버라이딩하여 문자열을 비교하도록 재정의 한 것이다.

7-2. String 클래스

String은 기본자료형이 아니다. String은 클래스이기에 참조자료형이다.

하지만 의아한 부분이 있다. 우리는 지금까지 String 변수를 만들 때 기본자료형의 변수 생성 문법을 사용했다.

클래스는 사용하려면 기본적으로 객체 생성이 원칙 아닌가? String도 클래스이기에 객체를 생성해서 사용해야 하지 않을까.

결론적으로, String 자료형의 변수를 만드는 문법은 두 가지가 존재한다.

```
String str1 = new String("java");  //클래스의 객체 생성 문법
String str2 = "java"; //기본자료형 변수 생성 문법
```

String도 클래스기 때문에 원칙적으로는 객체를 생성해서 사용해야 한다. 하지만 자주 사용하다보니 편하게 사용할 수 있도록 기본자료형처럼 사용하는 문법을 허용한 것이다. 하지만 String 자료형의 객체를 만드는 두 문법은 차이점이 존재한다.

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

    String str3 = new String("Simple String");
    String str4 = new String("Simple String");

    if(str1 == str2)
      System.out.println("str1과 str2는 동일 주소 참조 " ); //실행
    else
      System.out.println( " str1과 str2는 다른 주소 참조 " );

    if(str3 == str4)
      System.out.println( " str3과 str4는 동일 주소 참조 " );
    else
      System.out.println( " str3과 str4는 다른 다른 참조"); //실행
  }
```

'==' 연산은 주로 숫자 비교에 사용해왔다.

'==' 연산의 정확한 기능은 사실 숫자 비교 연산이 아니다.

원칙적으로는 두 데이터가 같은지 비교하는 것이다.

여기서의 두 데이터가 같다는 기준은 주소값이다.

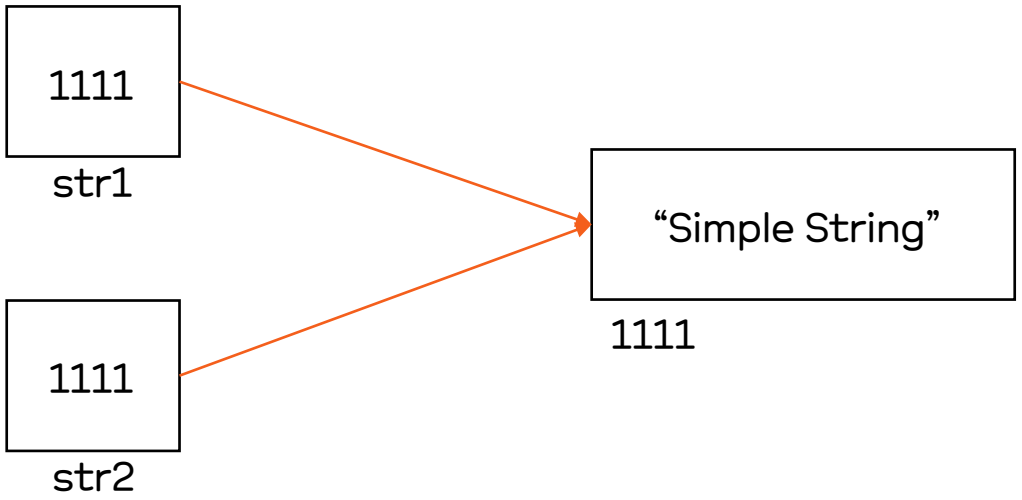
즉, '==' 연산의 정확한 해석은 비교하는 두 데이터가 같은 주소값을 참조하는지 판단하는 것이다. 그렇기에 문자열 비교에 사용하지 않는다.

이 기능에 추가적으로 숫자 비교 기능이 있는 것이다.

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

    String str3 = new String("Simple String");
    String str4 = new String("Simple String");

    if(str1 == str2)
      System.out.println("str1과 str2는 동일 주소 참조 " ); //실행
    else
      System.out.println( " str1과 str2는 다른 주소 참조 " );

    if(str3 == str4)
      System.out.println( " str3과 str4는 동일 주소 참조 " );
    else
      System.out.println( " str3과 str4는 다른 다른 참조"); //실행
}
```

String 변수는 생성 시 메모리 공간 낭비를 최소화하는 방향으로 설계되었다. 즉, 이미 만들어진 문자열과 같은 문자열을 객체로 생성하면 새로운 객체를 다시 만들지 않고, 기존에 존재한 데이터를 참조한다.

str1과 str2 객체의 참조값은 다음과 같다.

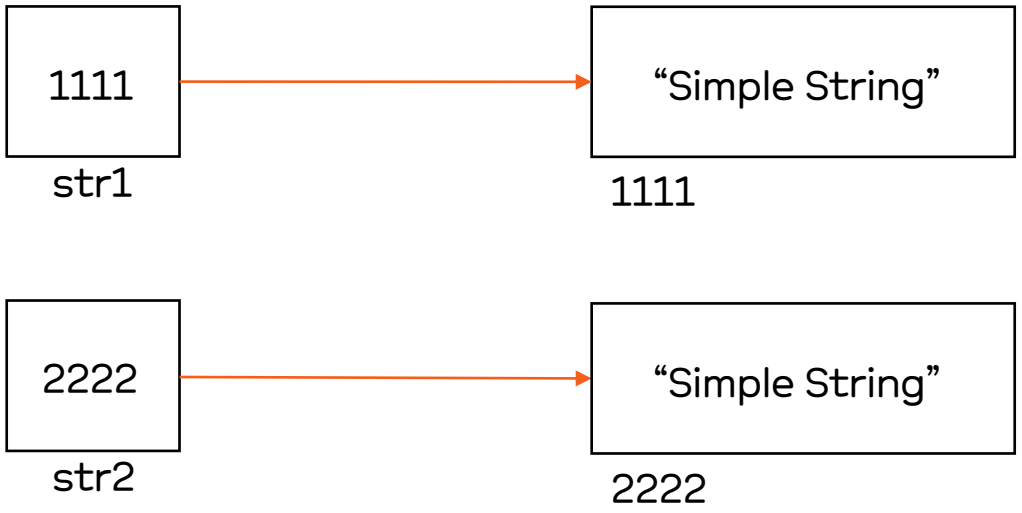


```
public static void main(String[] args) {
    String str1 = "Simple String";
    String str2 = "Simple String";

    String str3 = new String("Simple String");
    String str4 = new String("Simple String");

    if(str1 == str2)
      System.out.println("str1과 str2는 동일 주소 참조 " ); //실행
    else
      System.out.println( " str1과 str2는 다른 주소 참조 " );

    if(str3 == str4)
      System.out.println( " str3과 str4는 동일 주소 참조 " );
    else
      System.out.println( " str3과 str4는 다른 다른 참조"); //실행
}
```

new 키워드는 '새로운 객체를 만들어달라'는 키워드다.

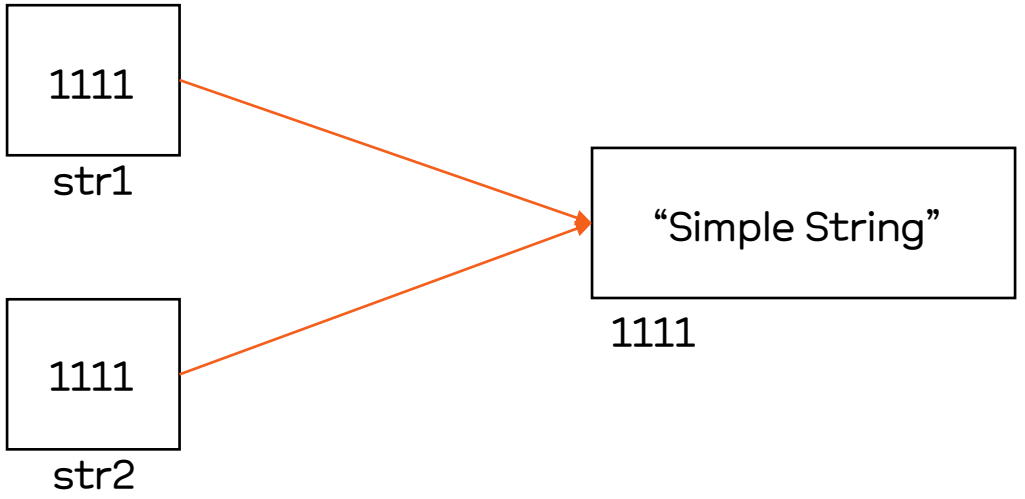
그렇기 때문에 new 키워드를 사용하여 객체를 생성하면 메모리 소모를 생각하지 않고, 명령대로 새 객체를 만들어준다.

두 객체 str3와 str4 객체의 참고값은 다음과 같다.



앞선 설명에서 의문점이 생기지 않았는가? 다시 한번 내용을 보겠다.

```
String str1 = “Simple String”;
String str2 = “Simple String”;
```



이 코드의 결과를 우측 그림과 같이 표현했다.

그럼 이 상태에서 다음 코드를 추가하면 str1과 str2의 값은 어떻게 되는가

```
str2 = “new String”;
```

또 다른 생각을 해보자. String은 참조변수이기에 기본자료형과 다르게 변수에는 실제 데이터가 저장된 메모리의 주소값이 저장되어있다.

이 점을 고려하여 다음 코드의 실행 결과 str1과 str2 변수의 값을 예측해보라.

```
String str1 = “java”;
String str2 = str1;
str2 = “python”;
```

여러분들이 기본자료형 변수의 생성과 참조자료형 객체의 생성에 대한 차이점을 잘 알고 있다면 도무지 이해가지 않는 현상이 발생하고 있을 것이다.

이런 특이한 결과가 발생하는 것은 String 객체는 immutable(이뮤터블) 객체이기 때문이다.

immutable 객체란 한 번 값이 할당되면 더 이상 그 값을 바꿀 수 없는 성질을 지닌 객체를 말한다.

즉, String 객체에 한 번 값이 저장되면 immutable 하기 때문에 그 값이 변경되지 않는다. 하지만.. 이런 설명을 들으면 더 혼란스러워질 것이다.

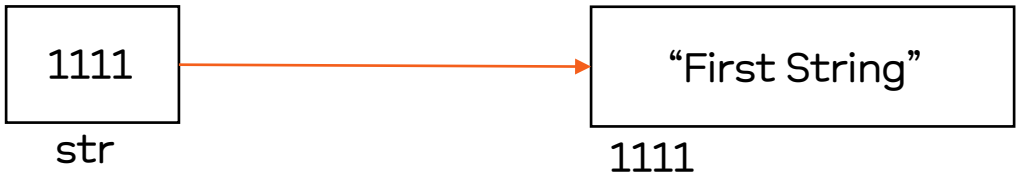
아래 코드를 해석해보자.

```
String str = “First String”;
str = “Second String”;
```

위 코드 실행 결과 str 객체에 저장된 문자열은 무엇인가? String은 값이 변하지 않는 immutable 객체이기 때문에 변하지 않는가?

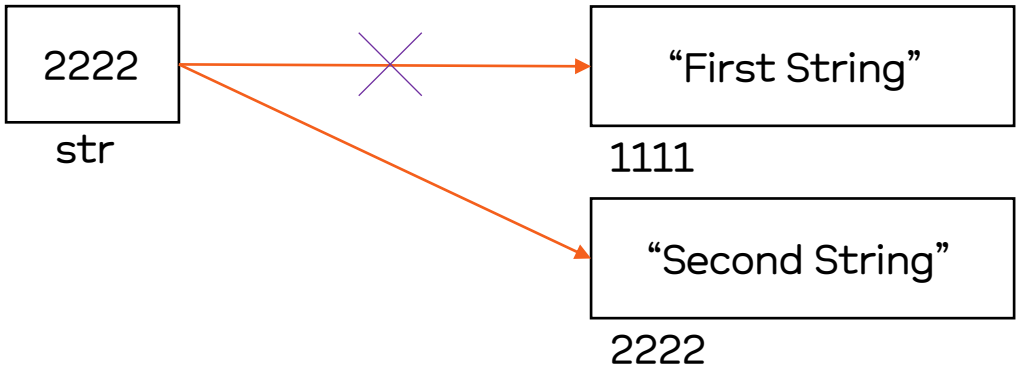
이전 슬라이드의 코드 실행 결과 결론적으로 값이 변경된다. 실행 흐름을 그림으로 그리면 다음과 같다.

String str = “First String”;



String은 참조자료형이기 때문에 위 그림과 같이 str변수는 실제 데이터인 “First String”이라는 문자열이 저장된 메모리의 주소값이 저장된다.

str = “Second String”;



String은 한 번 값이 저장되면 그 값을 변경할 수 없기 때문에 값 변경 명령이 주어지면 기존의 데이터 참조를 버리고 새로운 공간에 데이터를 생성 후 참조값을 재할당한다. 그렇기 때문에 Stiring 변수의 값을 지속적으로 변경하면 메모리가 낭비된다. (StringBilder 클래스를 알아보세요)

위 흐름을 파악했다면 앞선 슬라이드의 2가지 의문점이 해결될 것이다.

7-3. String 클래스의 메서드들

String 클래스에 정의된 메서드 중 많이 사용하는 메서드를 소개하겠다.

```
public static void main(String[] args) {
    String s1 = "Simple";
    String s2 = "String";

    //concat 메서드는 두 문자열의 나열 결과를 리턴한다.
    String s3 = s1.concat(s2);
    System.out.println(s3); // “SimpleString”

    //length() 메서드는 문자열의 길이를 정수 데이터로 리턴한다.
    int len = s3.length();
    System.out.println(len); // 12

    //valueOf() static 메서드는 매개변수를 전달된 데이터를 문자열로 리턴한다.
    //매개변수로는 int, double, char, boolean등 많이 자료형이 전달된다.
    String result = String.valueOf(12.345);
    System.out.println(result); // “12.345”
}
```

```
public static void main(String[] args) {
   String s = “hi java";

   //substring(시작 인덱스, 마지막 인덱스) 메서드는 일부분의 문자열을 추출하여 리턴한다. 두번째 매개변수를 생략하면 문자열의 끝까지 추출한다.
   String s1 = s.substring(3);
   System.out.println(s1); // “java”

   String s2 = s.substring(1, 4); //4번 index 전까지!
   System.out.println(s2); // “i  j”

   //split() 메서드는 매개변수로 전달된 문자열을 기준으로 문자열을 잘라 리턴한다.
   String s3 = “a,b,c”;
   String[] s4 = s3.split(“,”);
   System.out.println(Arrays.toString(s4)); // [a,b,c]

    //replace(찾을 문자열, 바꿀 문자열) 메서드는 특정 문자열을 원하는 문자열로 치환하여 리턴한다.
    String s5 = “010-1111-2222”;
    System.out.println(s5.replace(“-”, “.”)); // “010.1111.2222”
}
```

h	i		j	a	v	a
0	1	2	3	4	5	6