Chapter 20. 자바의 다양한 기본 클래스

20-1. Wrapper 클래스

■ 기본 자료형 데이터를 감싸는 Wrapper 클래스


```
public static void showData(Object obj)
{
    System.out.println(obj);
}
```

왼쪽의 메소드에서 보이듯이 기본 자료형 데이터를 인스턴스화 해야 하는 상황이 발생할 수 있다. 이러한 상황에 사용할 수 있는 클래스를 가리켜 **Wrapper 클래스**라 한다.

```
class IntWrapper
{
    private int num;
    public IntWrapper(int data)
    {
        num=data;
    }
    public String toString()
    {
        return ""+num;
    }
}
```

프로그래머가 정의한 int형 기본 자료형에 대한 Wrapper 클래스!

이렇듯 Wrapper 클래스는 기본 자료형 데이터를 저장 및 참조할 수 있는 구조로 정의된다.

■ 자바에서 제공하는 Wrapper 클래스

- Boolean — Boolean(boolean value)
- Character — Character(char value)
- Byte — Byte(byte value)
- Short — Short(short value)
- Integer — Integer(int value)
- Long — Long(long value)
- Float — Float(float value), Float(double value)

순전히 기본 자료형 데이터의 표현이 목적이라면, 별도의 클래스 정의 없이 제공되는 Wrapper 클래스를 사용하면 된다!

위의 클래스 이외에도 문자열 기반으로 정의된 Wrapper 클래스도 존재하기 때문에 다음과 같이 인스턴스 생성도 가능. 단, Character 클래스 제외!

```
Integer num1=new Integer("240")
Double num2=new Double("12.257");
```

자바제공 Wrapper 클래 사용의 예!

```
public static void main(String[] args)
{
    Integer intInst=new Integer(3);
    showData(intInst);
    showData(new Integer(7));
}
```

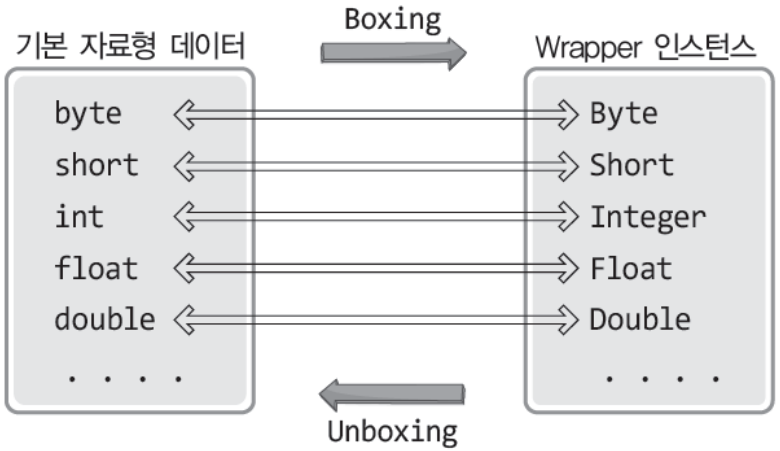
Wrapper 클래스의 두 가지 기능

Boxing

✈ 기본 자료형 데이터를 Wrapper 인스턴스로 감싸는 것!

UnBoxing

✈ Wrapper 인스턴스에 저장된 데이터를 꺼내는 것!




```
class BoxingUnboxing
{
    public static void main(String[] args)
    {
        Integer iValue=new Integer(10);
        Double dValue=new Double(3.14);
        System.out.println(iValue);
        System.out.println(dValue);
        iValue=new Integer(iValue.intValue()+10);
        dValue=new Double(dValue.doubleValue()+1.2);
        System.out.println(iValue);
        System.out.println(dValue);
    }
}
```

실행결과

```
10
3.14
20
4.34
```

본래 Wrapper 클래스는 산술연산을 고려해서 정의되는 클래스가 아니다. 따라서 Wrapper 인스턴스를 대상으로 산술연산을 할 경우에는 왼쪽과 같이 코드가 복잡해진다.

Auto Boxing & Auto Unboxing 을 이용하면 다소 편해진다!

Auto Boxing & Auto Unboxing

나는 정말 JAVA를 공부한 적이 없다구요

자바제공 Wrapper 클래스를 사용하는 것이 좋은 이유!

Auto Boxing

✈ 기본 자료형 데이터가 자동으로 Wrapper 인스턴스로 감싸지는 것!

Auto UnBoxing

✈ Wrapper 인스턴스에 저장된 데이터가 자동으로 꺼내지는 것!

기본 자료형 데이터와 와야 하는데, Wrapper 인스턴스가 있다면, Auto Unboxing!

인스턴스가 와야 하는데, 기본 자료형 데이터가 있다면, Auto Boxing!

```
public static void main(String[] args)
{
    Integer iValue=10;      // auto boxing
    Double dValue=3.14;     // auto boxing

    System.out.println(iValue);
    System.out.println(dValue);

    int num1=iValue;        // auto unboxing
    double num2=dValue;     // auto unboxing
    System.out.println(num1);
    System.out.println(num2);
}
```

실행결과

```
10
3.14
10
3.14
```

■ Auto Boxing & Auto Unboxing 어떻게?

나는 정말 JAVA를 공부한 적이 없다구요

```
public static void main(String[] args)
{
    Integer num1=10;
    Integer num2=20;

    num1++;
    System.out.println(num1);

    num2+=3;
    System.out.println(num2);

    int addResult=num1+num2;
    System.out.println(addResult);

    int minResult=num1-num2;
    System.out.println(minResult);
}
```

Auto Boxing, Auto Unboxing 동시 발생

```
num1=new Integer(num1.intValue()+1);
```

Auto Boxing, Auto Unboxing 동시 발생

```
num2=new Integer(num2.intValue()+3);
```

```
11
23
34
-12
```

실행결과

예제에서 보이듯이 Auto Boxing과 Unboxing은 다양한 형태로 진행된다. 특히 산술연산의 과정에서도 발생을 한다는 사실에 주목할 필요가 있다.

20-2. BigInteger 클래스와 BigDecimal 클래스

■ 매우 큰 정수의 표현을 위한 BigInteger 클래스


```
class SoBigInteger
{
    public static void main(String[] args)
    {
        System.out.println("최대 정수 : " + Long.MAX_VALUE);
        System.out.println("최소 정수 : " + Long.MIN_VALUE);

        BigInteger bigValue1=new BigInteger("10000000000000000000");
        BigInteger bigValue2=new BigInteger("-99999999999999999999");

        BigInteger addResult=bigValue1.add(bigValue2);
        BigInteger mulResult=bigValue1.multiply(bigValue2);

        System.out.println("큰 수의 덧셈결과 : "+addResult);
        System.out.println("큰 수의 곱셈결과 : "+mulResult);
    }
}
```

실행결과

```
최대 정수 : 9223372036854775807
최소 정수 : -9223372036854775808
큰 수의 덧셈결과 : 1
큰 수의 곱셈결과 : -999999999999999999990000000000000000000
```

큰 정수를 문자열로 표현한 이유는 숫자로 표현이 불가능하기 때문이다! 기본 자료형의 범위를 넘어서는 크기의 정수는 숫자로 표현 불가능하다!

■ 오차 없는 실수의 표현을 위한 BigDecimal 클래스

나는 정말 JAVA를 공부한 적이 없다고

```
public static void main(String[] args)
{
    BigDecimal e1=new BigDecimal(1.6);
    BigDecimal e2=new BigDecimal(0.1);

    System.out.println("두 실수의 덧셈결과 : "+ e1.add(e2));
    System.out.println("두 실수의 곱셈결과 : "+ e1.multiply(e2));
}
```

실수 1.6과 0.1을 숫자로 표현하는 순간 이미 오차가 포함되어 버렸다.

```
1.899999999999999966693309261245303 78
0.340000000000000099200722162640837
```

실행결과

```
public static void main(String[] args)
{
    BigDecimal e1=new BigDecimal("1.6");
    BigDecimal e2=new BigDecimal("0.1");

    System.out.println("두 실수의 덧셈결과 : "+ e1.add(e2));
    System.out.println("두 실수의 곱셈결과 : "+ e1.multiply(e2));
}
```

실수도 문자열로 표현을 해서, BigDecimal 클래스에 오차 없는 값을 전달해야 한다!

```
두 실수의 덧셈결과 : 1.7
두 실수의 곱셈결과 : 0.16
```

실행결과

20-3. Math 클래스와 난수의 생성, 그리고 문자열 토큰의 구분

■ 수학관련 기능을 제공하는 Math 클래스


```
public static void main(String[] args)
{
    System.out.println("원주율 : " + Math.PI);
    System.out.println("2의 제곱근 : " + Math.sqrt(2));

    System.out.println(
        "파이에 대한 Degree : " + Math.toDegrees(Math.PI));
    System.out.println(
        "2파이에 대한 Degree : " + Math.toDegrees(2.0*Math.PI));

    double radian45=Math.toRadians(45);    // 라디안으로의 변환!
    System.out.println("싸인 45 : " + Math.sin(radian45));
    System.out.println("코싸인 45 : " + Math.cos(radian45));
    System.out.println("탄젠트 45 : " + Math.tan(radian45));

    System.out.println("로그 25 : " + Math.log(25));
    System.out.println("2의 4승 : "+ Math.pow(2, 4));
}
```

실행결과

```
원주율 : 3.141592653589793
2의 제곱근 : 1.4142135623730951
파이에 대한 Degree : 180.0
2파이에 대한 Degree : 360.0
싸인 45 : 0.7071067811865475
코싸인 45 : 0.7071067811865476
탄젠트 45 : 0.9999999999999999
로그 25 : 3.2188758248682006
2의 4승 : 16.0
```

Math 클래스에는 수학관련 메소드가 static으로 정의되어 있다는 사실을 기억하는 것이 중요하다! 필요한 메소드는 API 문자를 통해 참조하면 된다. 단 대부분의 메소드가 라디안 단위로 정의되어 있음은 기억하기 바란다!

■ 난수(Random Number)의 생성


```
Random rand=new Random();
```

- boolean nextBoolean() — boolean형 난수 반환
- int nextInt() — int형 난수 반환
- long nextLong() — long형 난수 반환
- int nextInt(int n) — 0이상 n미만의 범위 내에 있는 int형 난수 반환
- float nextFloat() — 0.0 이상 1.0 미만의 float형 난수 반환
- double nextDouble() — 0.0 이상 1.0 미만의 double형 난수 반환

```
public static void main(String[] args)
{
    Random rand=new Random();

    for(int i=0; i<100; i++)
        System.out.println(rand.nextInt(1000));
}
```

0 이상 1,000 미만의 난수가 총 100개 생성되어 출력!

■ 씨드(Seed) 기반의 난수 생성

나는 정말 JAVA를 공부한 적이 없다구요

```
public static void main(String[] args)
{
    Random rand=new Random(12);
    for(int i=0; i<100; i++)
        System.out.println(rand.nextInt(1000));
}
```

12라는 씨앗이 전달되었으니, 12를 심어서 만들어지는 난수가 총 100개 생성된다.

씨앗! 이 동일하면 생성되는 난수의 패턴은 100% 동일!

이렇듯 컴퓨터의 난수는 씨앗을 기반으로 생성되기 때문에 가짜 난수(Pseudo-random number)라 불린다.

```
public static void main(String[] args)
{
    Random rand=new Random(12);
    rand.setSeed(System.currentTimeMillis());
    for(int i=0; i<100; i++)
        System.out.println(rand.nextInt(1000));
}
```

현재 시간을 밀리 초 단위로 반환

Random 클래스의 디폴트 생성자 내에서는 왼편의 코드와 같이 씨드를 설정한다.

매 실행 시마다 다른 유형의 난수를 발생시키는 방법

■ 문자열 토큰(Token)의 구분

"08 : 45"

"11 : 24"

콜론 : 을 기준으로 문자열을 나눈다고 할 때,

✈ 08, 45, 11, 24는 토큰(token)

✈ 콜론 : 는 구분자(delimiter)

```
public static void main(String[] args)
{
    String strData="11:22:33:44:55";
    StringTokenizer st=new StringTokenizer(strData, ":");

    while(st.hasMoreTokens())
        System.out.println(st.nextToken());
}
```

토큰을 나눌 대상 문자열 정보

구분자 정보

반환할 토큰이 있는가?

다음 번 토큰 반환!

실행결과

```
11
22
33
44
55
```

구분자 정보는 둘 이상 담을 수 있다. 하나의 문자열 안에 둘 이상을 담을 수 있다.