■ 기본 자료형을 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.

■ Wrapper 클래스

```
public class WrapperFloat {
    private float number;

    public WrapperFloat(float number) {
        this.number = number;
    }

    public float floatValue() {
        return number;
    }

    @Override
    public String toString() {
        return number + "";
    }
}
```

```
public class Wrapper1 {
    public static void main(String[] args) {
        WrapperFloat wf =
            new WrapperFloat(10.0f);
        float f = wf.floatValue();
        System.out.println(wf);
    }
}
```

■ 자바에서 제공하는 Wrapper 클래스

기본 타입	포장 클래스
byte	Byte
char	Character
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean

■ 자바에서 제공하는 Wrapper 클래스

```
public class Wrapper2 {
    public static void main(String[] args) {
        Integer in = new Integer(10000);
        System.out.println(in.intValue());

        Float fl = new Float("100.0f");
        System.out.println(fl.floatValue());

        Double db = new Double("123.123");
        System.out.println(db);
    }
}
```

■ 박싱(Boxing)과 언박싱(Unboxing)

- 박싱(Boxing): 기본 타입의 값을 포장 객체로 만드는 과정
- 언박싱(Unboxing): 포장 객체에서 기본 타입의 값을 얻어내는 과정
- 박싱 하는 방법

  - 생성자 이용

기본 타입의 값을 줄 경우	문자열을 줄 경우
Byte obj = new Byte(10);	Byte obj = new Byte("10");
Character obj = new Character('가');	
Short obj = new Short(100);	Short obj = new Short("100");
Integer obj = new Integer(1000);	Integer obj = new Integer("1000");
Long obj = new Long(10000);	Long obj = new Long("10000");
Float obj = new Float(2.5F);	Float obj = new Float("2.5F");
Double obj = new Double(3.5);	Double obj = new Double("3.5");
Boolean obj = new Boolean(true);	Boolean obj = new Boolean("true");

  - 메소드 이용

```
Integer obj = Integer.valueOf(1000);
Integer obj = Integer.valueOf("1000");
```

■ 박싱(Boxing)과 언박싱(Unboxing)

● 언박싱 하는 방법

- 각 포장 클래스마다 가지고 있는 클래스 호출
- 기본 타입명 + Value()

기본 타입의 값을 이용		
byte	num	= obj.byteValue();
char	ch	= obj.charValue();
short	num	= obj.shortValue();
int	num	= obj.intValue();
long	num	= obj.longValue();
float	num	= obj.floatValue();
double	num	= obj.doubleValue();
boolean	bool	= obj.booleanValue();

■ 박싱(Boxing)과 언박싱(Unboxing)

● Auto Boxing - 포장 클래스 타입에 기본값이 대입될 경우 발생

```
Integer obj = 100;     //자동 박싱
```

```
List<Integer> list = new ArrayList<Integer>();
list.add(200);    //자동 박싱
```

● Auto Unboxing - 기본 타입에 포장 객체가 대입될 경우 발생

```
Integer obj = new Integer(200);
int value1 = obj;              //자동 언박싱
int value2 = obj + 100;     //자동 언박싱
```

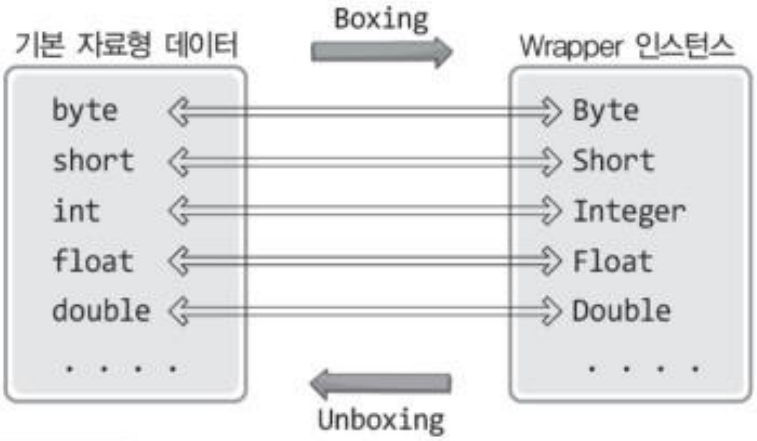
■ Wrapper 클래스 사용

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

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
```

■ BigInteger 클래스 : 매우 큰 정수 표현

```
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);
    }
}
```

실행 결과

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

■ BigInteger 클래스

```
public class BigIntegerExam {
    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("10000000000000000000000000000");
        BigInteger bigValue2 = new BigInteger("10000000000000000000000000000");

        BigInteger addResult = bigValue1.add(bigValue2);
        BigInteger subResult = bigValue1.subtract(bigValue2);
        BigInteger mulResult = bigValue1.multiply(bigValue2);
        BigInteger divResult = bigValue1.divide(bigValue2);

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

■ BigDecimal 클래스 : 오차 없는 실수 표현

```
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.

```
1.89999999999999996669330926124530378
0.34000000000000009992007221626408837
```

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

■ BigDecimal 클래스

```
public class BigDecimalExam {
    public static void main(String[] args) {
        float f1 = 1.6f;
        float f2 = 0.1f;
        System.out.println((float)(f1 * f2));

        double d1 = 1.6;
        double d2 = 0.1;
        System.out.println((double)(d1 * d2));

        BigDecimal e1 = new BigDecimal("1.6");
        BigDecimal e2 = new BigDecimal("0.1");

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