java.lang package II

- Object 클래스
- wrapper 클래스

Object 클래스

☞ String 클래스와 Object 클래스

모든 클래스가 Object 클래스를 상속하는 것과 관련해서 기억할 것

- Object 클래스에는 toString 메소드가 다음의 형태로 정의되어 있다.

  public String toString( ) { . . . }

- 그리고 우리가 흔히 호출하는 println 메소드는 다음과 같이 정의되어 있다.

  public void println(Object x) { . . . }

- 때문에 모든 인스턴스는 println 메소드의 인자로 전달될 수 있다.
- 인자로 전달되면, toString 메소드가 호출되고, 이때 반환되는 문자열이 출력된다.
- 때문에 toString 메소드는 적절한 문자열 정보를 반환하도록 오버라이딩하는 것이 좋다!

```
class Friend
{
    String myName;
    public Friend(String name)
    {
        myName=name;
    }
    public String toString()
    {
        return "제 이름은 "+myName+"입니다.";
    }
}
```

```
public static void main(String[] args)
{
    Friend fnd1=new Friend("이종수");
    Friend fnd2=new Friend("현주은");

    System.out.println(fnd1);
    System.out.println(fnd2);
```

☞ 인스턴스 비교

```
class IntNumber
{
    int num;
    public IntNumber(int num) { this.num=num; }
    public boolean isEquals(IntNumber numObj)
    {
        if(this.num==numObj.num)
            return true;
        else
            return false;
    }
}
```

이전에 언급했듯이 == 연산자는 참조 값 비교를 한다. 따라서 인스턴스간 내용비교를 위해서는 내용비교 기능의 메소드가 필요하다.

```
public static void main(String[] args)
{
    IntNumber num1=new IntNumber(10);
    IntNumber num2=new IntNumber(12);
    IntNumber num3=new IntNumber(10);

    if(num1.isEquals(num2))
        System.out.println("num1과 num2는 동일한 정수");
    else
        System.out.println("num1과 num2는 다른 정수");

    if(num1.isEquals(num3))
        System.out.println("num1과 num3는 동일한 정수");
    else
        System.out.println("num1과 num3는 다른 정수");
}
```

```
num1과 num2는 다른 정수
num1과 num3는 동일한 정수
```

☞ equals 메소드

```
class IntNumber
{
    int num;
    public IntNumber(int num)
    {
        this.num=num;
    }
    public boolean equals(Object obj)
    {
        if(this.num==((IntNumber)obj).num)
            return true;
        else
            return false;
    }
}
```

Java에서는 인스턴스간의 내용비교를 목적으로 Object 클래스에 equals 메소드를 정의해 놓았다.
따라서 새로 정의되는 클래스의 내용비교가 가능하도록 이 메소드를 오버라이딩하는 것이 좋다!

Object 클래스의 equals 메소드를 인스턴스의 내용비교 메소드로 지정해 놓았기 때문에,처음 접하는 클래스의 인스턴스라 하더라도 equals 메소드의 호출을 통해서 인스턴스간 내용비교를 할 수 있다.

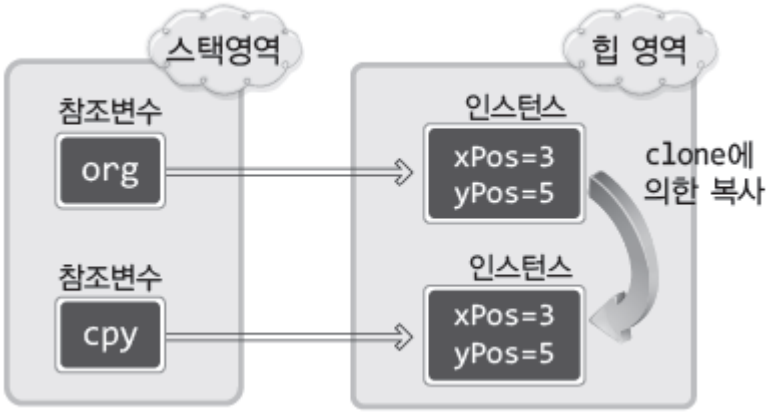
☞ 인스턴스의 복사(복제): clone 메소드

- Object 클래스에는 인스턴스의 복사를 목적으로 clone이라는 이름의 메소드가 정의되어 있다. 단, 이 메소드는 Cloneable 인터페이스를 구현하는 클래스의 인스턴스에서만 호출될 수 있다.
- Cloneable 인터페이스의 구현은 다음의 의미를 지닌다.
  "이 클래스의 인스턴스는 복사를 해도 됩니다."
- 사실 인스턴스의 복사는 매우 민감한 작업이다. 따라서 클래스를 정의할 때 복사의 허용여부를 결정하도록 Cloneable 인터페이스를 통해서 요구하고 있다.

```
class Point implements Cloneable
{
    private int xPos;
    private int yPos;
    . . . . .
    public Object clone() throws CloneNotSupportedException
    {
        return super.clone();
    }
}
```

clone 메소드는 protected로 선언되어 있다.
따라서 외부 호출이 가능하도록 public으로 오버라이딩!

인스턴스를 통째로 복사한다!



☞ String 인스턴스와 배열 인스턴스 대상의 복사

- String 인스턴스에 저장되어 있는 문자열 정보는 변경되지 않는다. 따라서 굳이 String 인스턴스를 깊은 복사의 목록에 포함시킬 필요는 없다.
- 배열 대상의 clone 메소드의 호출 결과는 배열의 복사이다! 즉, 배열과 배열에 저장된 인스턴스의 참조 값은 복사가 되지만, 배열의 참조 값이 참조하는 인스턴스까지 복사가 진행되지는 않는다!

Wrapper 클래스

☞ Wrapper 클래스

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

기본 자료형 데이터를 인스턴스화 해야 하는 상황이 발생할 수 있다. 이러한 상황에 사용할 수 있는 클래스를 가리켜 Wrapper 클래스라 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)
• Double	Double(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");
```

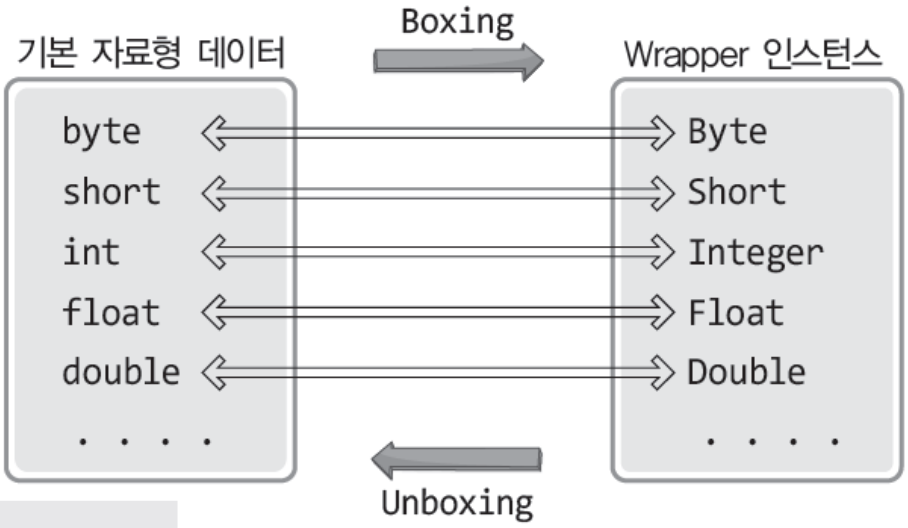
자바제공 Wrapper 클래스 사용의 예!

```
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);
    }
}
```

```
10
3.14
20
4.34
```

본래 Wrapper 클래스는 산술 연산을 고려해서 정의되는 클래스가 아니다. 따라서 Wrapper 인스턴스를 대상으로 산술연산을 할 경우에는 왼쪽과 같이 코드가 복잡해 진다.

Auto Boxing & Auto Unboxing을 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 데이터가 와야 하는데, Wrapper 인스턴스가 있다면, Auto Unboxing!
인스턴스가 와야 하는데, 기본 자료형데이터가 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);
}
```

```
10
3.14
10
3.14
```

☞ Auto Boxing & Auto Unboxing 어떻게?

```
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);
```

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

Auto Boxing, Auto Unboxing 동시 발생

```
11
23
34
-12
```

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

java.util package II

- Calendar 클래스
- Date 클래스
- Random 클래스

☞ 날짜와 시간을 읽어오는 Calendar 클래스

- 컴퓨터에 내장되어 있는 시스템 시계(system clock)로부터 현재의 시간을 읽어오는 클래스.
- 추상 클래스로 선언되어 있음 – 날짜와 시간을 계산하는 방법이 지역과 문화, 나라에 따라 다르기 때문.
- 날짜와 시간을 계산하는데 일반적으로 필요한 메소드만 있음.
- 특정한 역법(시간을 구분하고 날짜에 순서를 매겨 나가는 방법)을 따르는 계산 로직은 Calendar 클래스를 상속하는 서브클래스에서 구현

java.math package

- BigInteger 클래스
- BigDecimal 클래스
- Math 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);
    }
}
```

```
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

```
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));
}
```

실수1.6과 0.1을 숫자로 표현하는순간 이미 오차가 포함되어 버렸다.

```
1.89999999999999966693309261245303 78
0.34000000000000009992007221626408 37
```

```
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 표현을해서, BigDecimal 클래스에 오차 없는 값을 전달해야 한다!

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