Chap 02. Identifiers, Keywords and Types

학습목표

1. Identifier / Keywords
2. Data types
3. Constructing & Initializing Object
4. Pass by value
5. this keyword

```
/*
 * Class : Movie
 */
package chap02.entity;
/**
 *  영화정보를 담는 클래스
 *
 * @author 홍길동
 * @version 1.0, 2013/12/01
 */
public class Movie {
    // 영화 제목
    private String title;
    // 감독
    private String director;
    // 주연
    private String starring;
    // 등급
    private String rating;

 ~~~~~~~~~~~~생 략 ~~~~~~~~~~~~
```

```
    /**
     * 모든 정보를 갖는 Movie 클래스의 생성자
     * @param title 제목
     * @param director 감독
     * @param starring 주연
     * @param rating 등급
     */
    public Movie(String title, String director,
                 String starring, String rating) {
        this.title = title;
        this.director = director;
        this.starring = starring;
        this.rating = rating;
    }

    /**
     * 제목 정보에 대한 getter
     * @return 영화제목
     */
    public String getTitle() {
        return title;
    }
~~~~~~~~~~~~~~~~~~생 략 ~~~~~~~~~~~~~~~~~~
```

Semicolons, Blocks, and White Space

❑ 한 문장(Statement)은 여러 라인에 걸쳐서 기술될 수 있으며, Semicolon(;)으로 그 끝을 표시한다.

```
totals = a + b + c + d + e + f;
```

```
totals = a + b + c +
         d + e + f;
```

❑ Block은 여러 문장을 한 set으로 처리하도록 해주며, Bracee({ })로 묶는다.

```
if ( x > y ) {
    x = x + 1;
    y = y + x;
} else {
    x = x = y;
}
```

Semicolons, Blocks, and White Space

- 클래스, 메소드를 정의할 때 { } 로 묶는다.
- 중첩으로 block 지정이 가능하다.
- 공백 문자(space, tab, enter)는 연산자와 피연산자 사이에 얼마든지 올 수있다.

```
totals = a + b + c + d + e + f;
```

```
totals = a + b + c +
         d + e + f;
```

❑ Identifier 정의

- 클래스, 변수, 메소드에 주는 이름

❑ Identifier Naming Rule

- Unicode 문자, '_', '$' 로 구성한다.
- 숫자로 시작할 수 없다.
- 대/소문자를 구분 한다.

❑ Identifier 정의시 주의점

- Keyword는 Identifier로 사용되어 질 수 없다.
- true, false, null 은 keyword가 아닌 literal(상수) 이지만, 예약된 값이므로 변수명으로 사용할 수 없다. goto, const는 자바 키워드로 정의는 되어있으나 사용되지 않는다.
- sizeof 라는 연산자는 자바에선 안 쓰인다. 키워드도 아님

Java Keywords

abstract	continue	goto	package	synchronized
assert	default	if	private	this
boolean	do	implements	protected	throw
break	double	import	public	throws
byte	else	instanceof	return	transient
case	extends	int	short	try
catch	final	interface	static	void
char	finally	long	strictfp	volatile
class	float	native	super	while
const	for	new	switch	

❑ Java Date Type

- Primitive Data Type
  - 기본 데이터 타입으로 단일 값을 갖는다.
  - 정수형/실수형/문자형/논리형 이 있다.

- Reference Data Type
  - 객체에 대한 주소 값(reference)을 의미한다.
  - Primitive Data Type을 제외한 모든 것이 Reference Date Type이다.

❑ 논리형 : boolean

- 값 범위 : true / false
- default value : false

❑ 문자형 : char

- 값 범위 : 16bit Uni-code 문자
- 표현법 : ‘ ’ 으로 문자를 둘러 싼다. (‘A’, ‘\n’, ‘\uAC00’)
- default value : ‘\u0000’

Primitive Data Type

❑ 정수형 : byte / short / int / long

	byte	short	int	long	
▪ 값 범위 :	1	2	4	8	bytes 내의 정수 값
▪ default value :	0	0	0	0L	

- 표현법 : 10진수(10), 8진수(012), 16진수(0x0A) 로 표현 가능하다.
- 정수형 literal의 data type은 int 형이다.

❑ 실수형 : float / double

	float	double	
▪ 값 범위 :	4	8	bytes 내의 실수 값
▪ default value :	0.0F	0.0D	

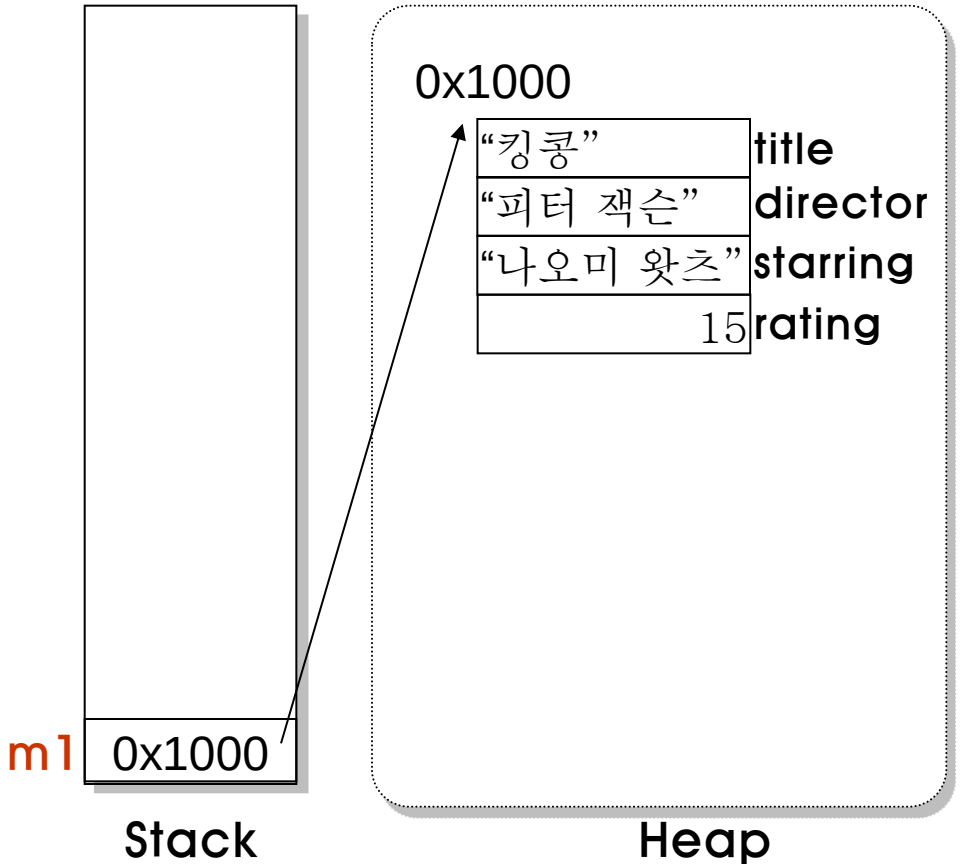
- 실수형 literal의 data type은 double 형이다.

Reference Data Type

- Primitive type을 제외한 모든 Data Type은 Reference type이다.
- Reference type의 변수는 new를 이용해 생성한 객체의 주소 값을 가지고, 그 객체를 제어하는데 쓰인다.
- 객체의 생성은 new 키워드로서 이루어지며, 객체의 데이터 타입은 그 객체의 클래스로서 표현된다.
- default value : null

```
public class Movie  {
    public String title = "킹콩";
    public String director = "피터 잭슨";
    public String starring = "나오미 왓츠";
    public int rating = 15;
}
```

```
Movie m1 = new Movie();
```



❑ 객체의 생성은 new 키워드로서 이루어진다.

ex) Dog d = new Dog();

❑ 객체 생성시, 메모리가 할당되는 순서는 다음과 같다.

1) 메모리 할당 (reference 변수, 객체)

2) 객체의 멤버변수 초기화

i) default value로 초기화

ii) 명시적 초기화

iii) Constructor에 의한 초기화

3) 객체의 주소 값이, reference 변수에 할당

this 키워드

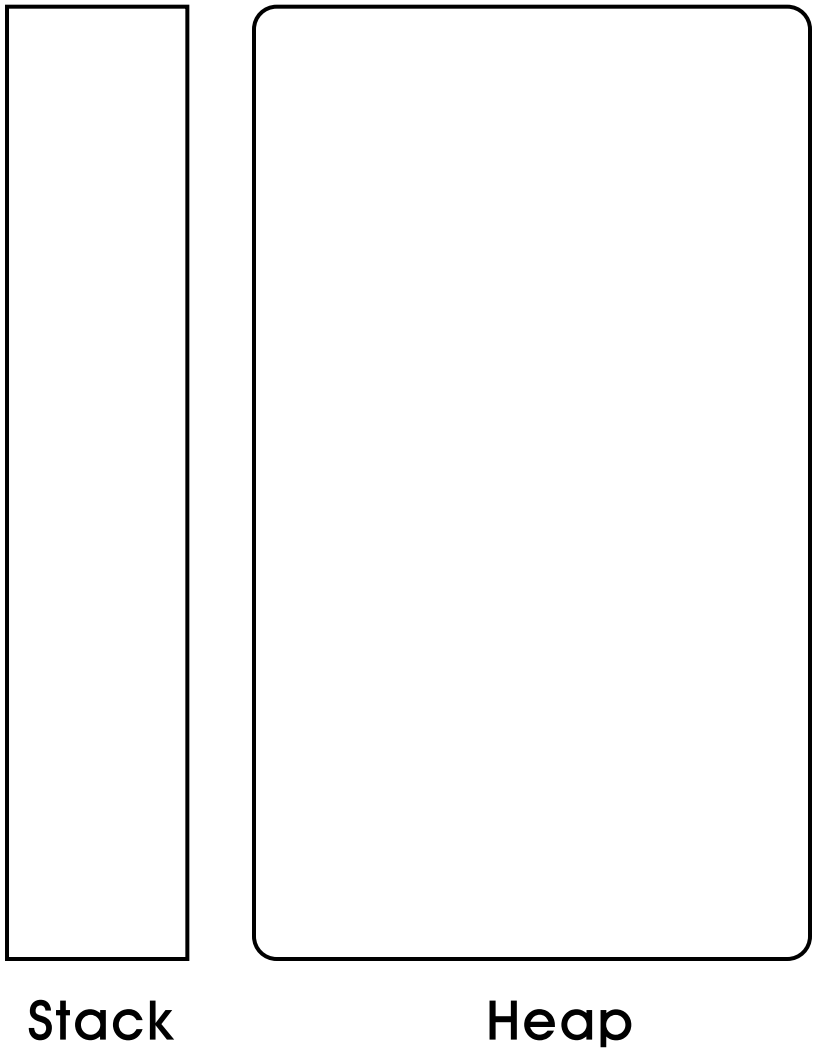
- 자기자신의 객체를 가리킨다.
- 용도 – 자기자신이 멤버변수 참조 : this.*멤버변수_이름*
  - 자기자신의 멤버메소드 참조 : this.*멤버메소드_이름()*
  - 자기 자신의 생성자 호출 : this(*파리미터_리스트*)

```
public class MyDate {

    private int day   = 1;
    private int month = 1;
    private int year  = 2000;


    public MyDate( int day, int month, int year ) {
        this.day       = day;
        this.month     = month;
        this.year      = year;
    }
}
```

Constructing & Initializing Objects


```
public class MyDate {

    private int day   = 1;
    private int month = 1;
    private int year  = 2000;

    public MyDate( int day, int month, int year ) {
        this.day      = day;
        this.month    = month;
        this.year     = year;
    }

    public void print() {
        System.out.println( "The Day is " +
                    day + "-" + month + "-" + year  );
    }
}
```

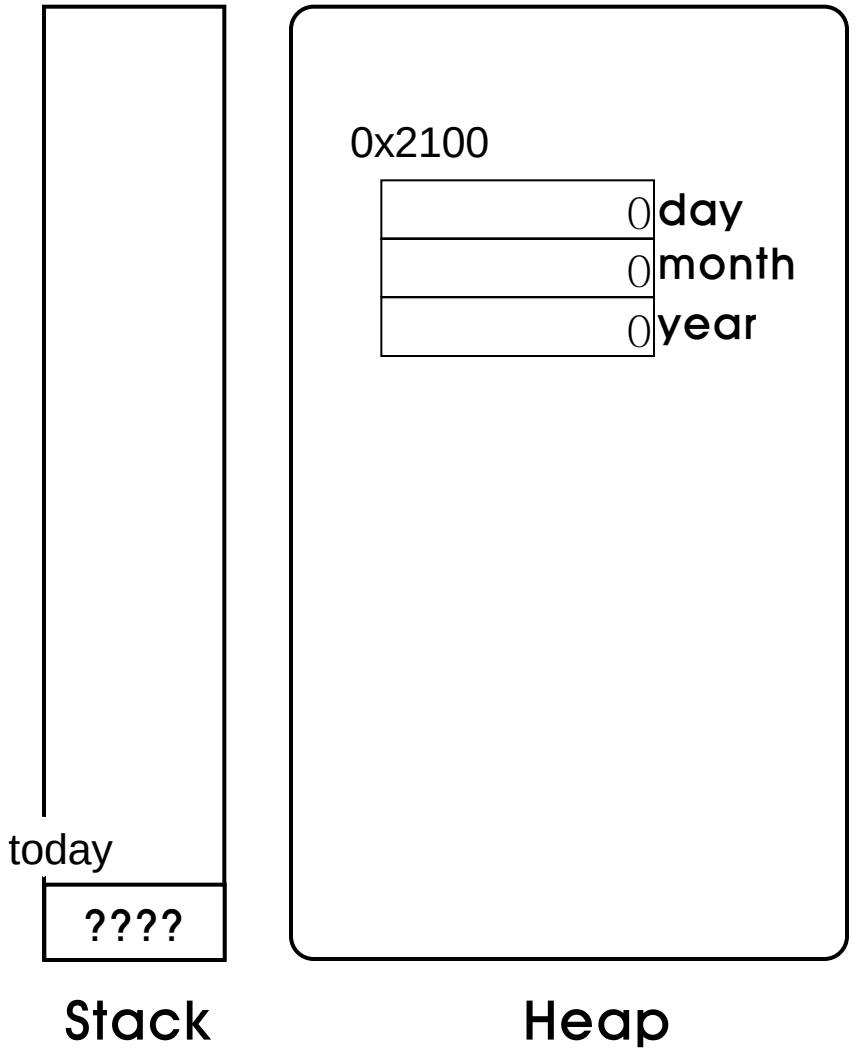
```
public class TestMyDate {
    public static void main( String [ ] args ) {
        MyDate today = new MyDate( 22, 7, 1964 );

        today.print();
    }
}
```



Constructing & Initializing Objects


```
public class MyDate {

   private int day   = 1;
   private int month = 1;
   private int year  = 2000;

   public MyDate( int day, int month, int year ) {
      this.day   = day;
      this.month = month;
      this.year  = year;
   }

   public void print() {
      System.out.println( "The Day is " +
                  day + "-" + month + "-" + year  );
   }
}
```

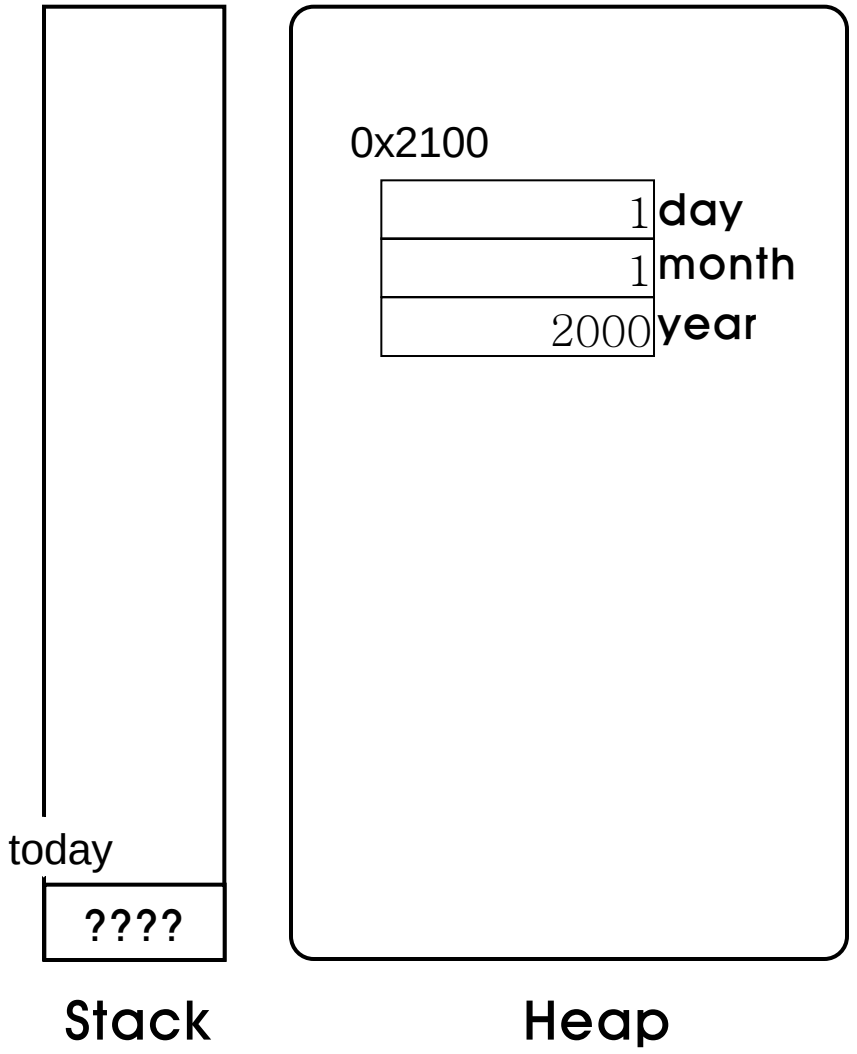
```
public class TestMyDate {
   public static void main( String [ ] args ) {
      MyDate today = new MyDate( 22, 7, 1964 );

      today.print();
   }
}
```



Constructing & Initializing Objects


```
public class MyDate {

    private int day   = 1;
    private int month = 1;
    private int year  = 2000;

    public MyDate( int day, int month, int year ) {
        this.day      = day;
        this.month    = month;
        this.year     = year;
    }

    public void print() {
        System.out.println( "The Day is " +
                    day + "-" + month + "-" + year  );
    }
}
```

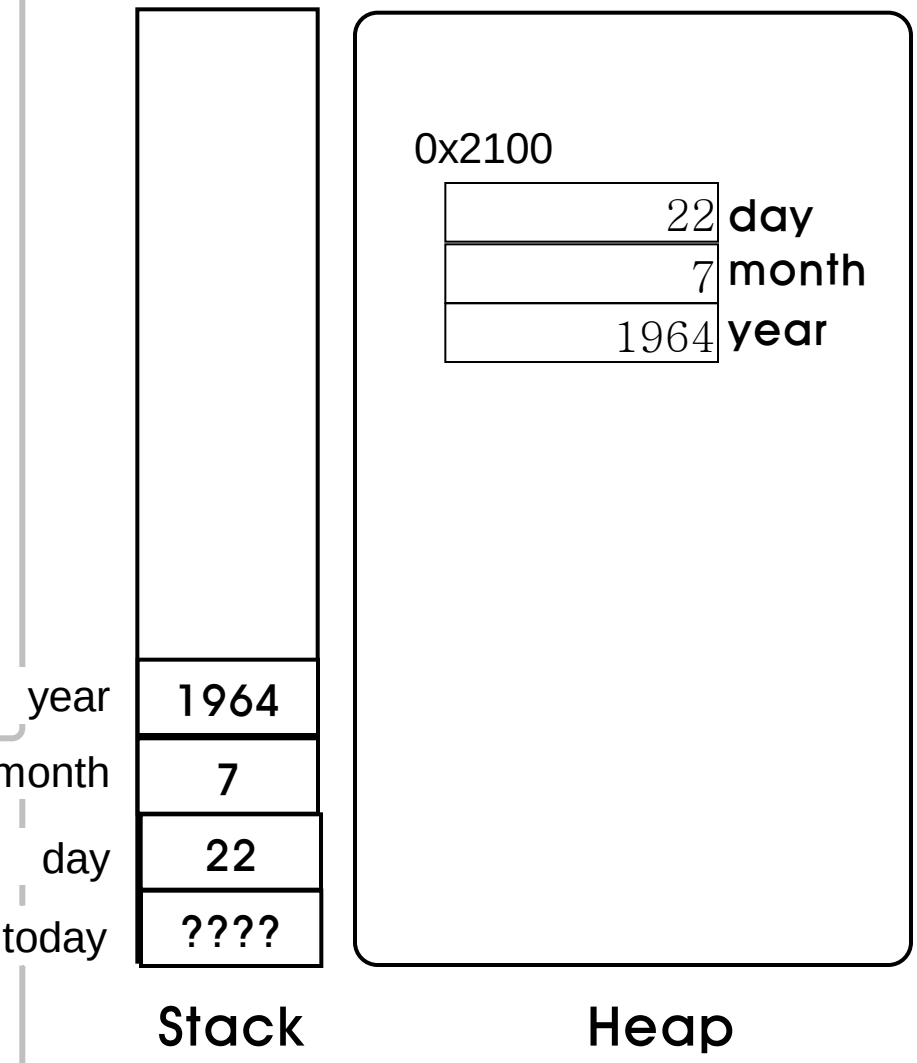
```
public class TestMyDate {
    public static void main( String [ ] args ) {
        MyDate today = new MyDate( 22, 7, 1964 );

        today.print();
    }
}
```



Constructing & Initializing Objects

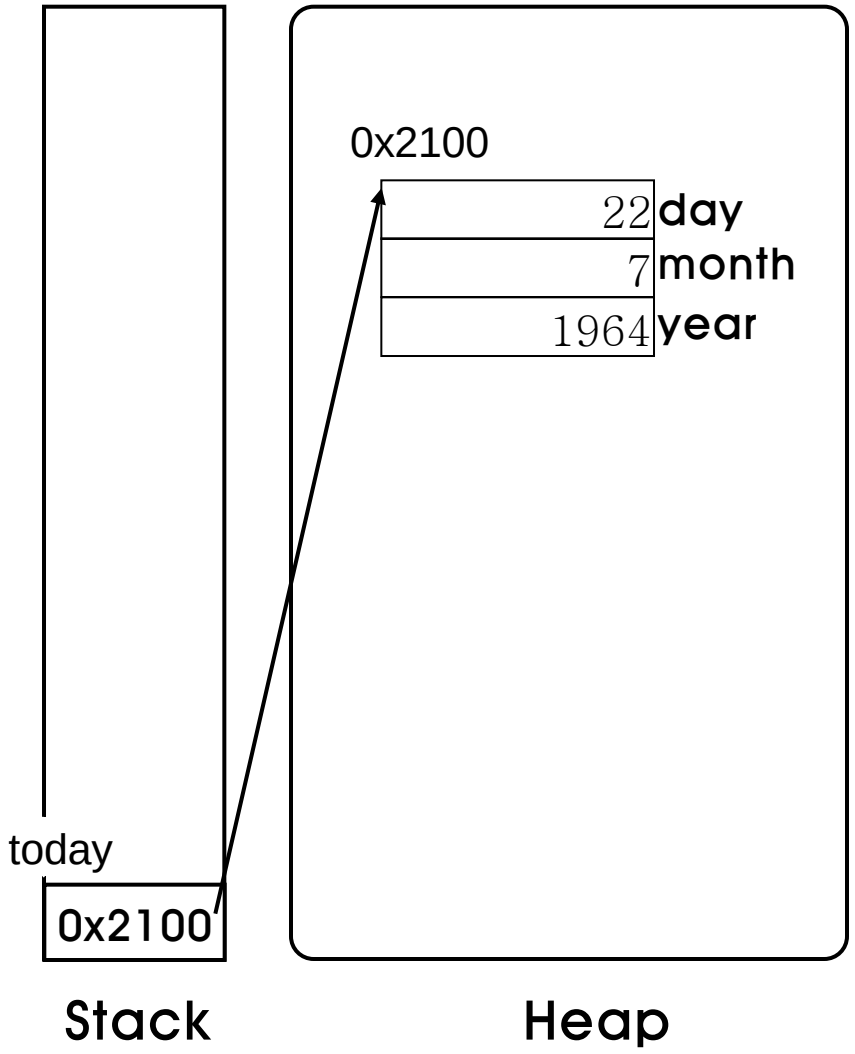

```
public class MyDate {

    private int day   = 1;
    private int month = 1;
    private int year  = 2000;

    public MyDate( int day, int month, int year ) {
        this.day      = day;
        this.month    = month;
        this.year     = year;
    }

    public void print() {
        System.out.println( "The Day is " +
                    day + "-" + month + "-" + year  );
    }
}
```

```
public class TestMyDate {
    public static void main( String [ ] args ) {
        MyDate today = new MyDate( 22, 7, 1964 );

        today.print();
    }
}
```



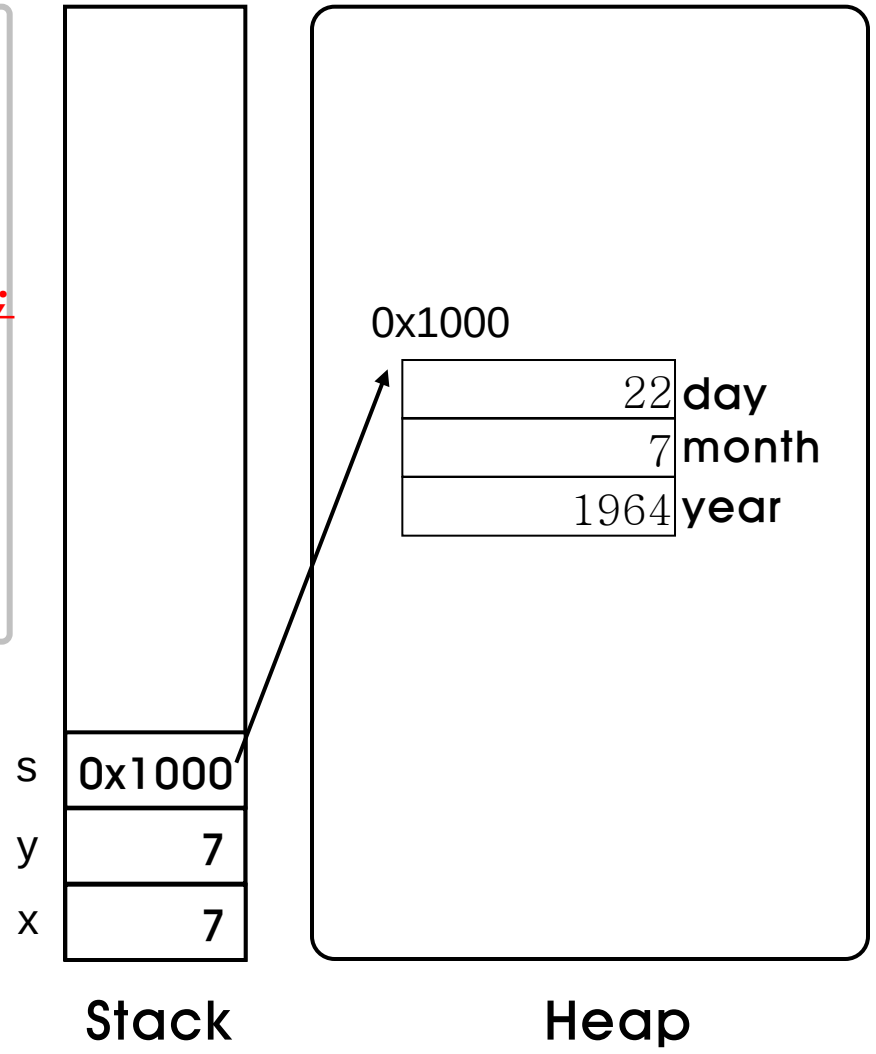
```
public class MyDate {

    private int day   = 1;
    private int month = 1;
    private int year  = 2000;

    public MyDate( int day, int month, int year ) {
        this.day      = day;
        this.month    = month;
        this.year     = year;
    }

    public void print() {
        System.out.println( "The Day is " +
                    day + "-" + month + "-" + year  );
    }
}
```

```
public class TestMyDate {
    public static void main( String [ ] args ) {
        MyDate today = new MyDate( 22, 7, 1964 );

        today.print();
    }
}
```



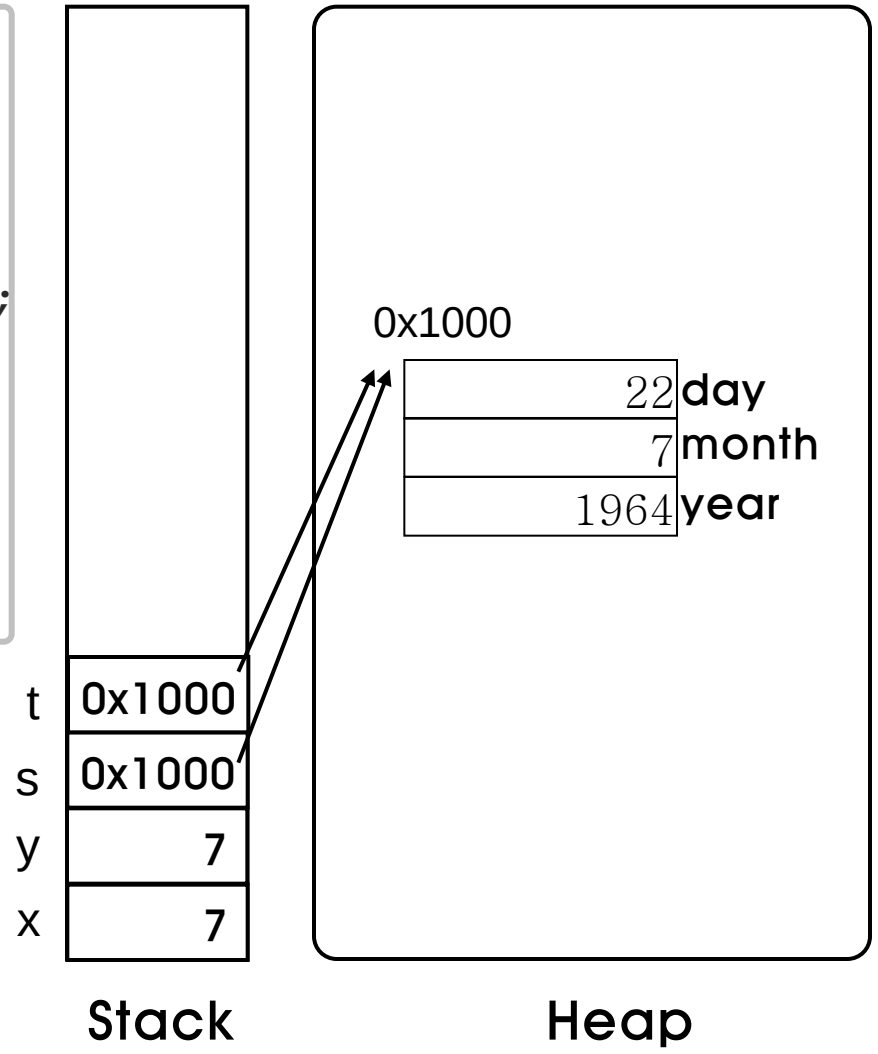
Assigning References


```
int x = 7;

int y = x;

MyDate s = new MyDate( 22, 7, 1964 );

MyDate t = s;

t = new MyDate( 22, 12, 1964 );
```



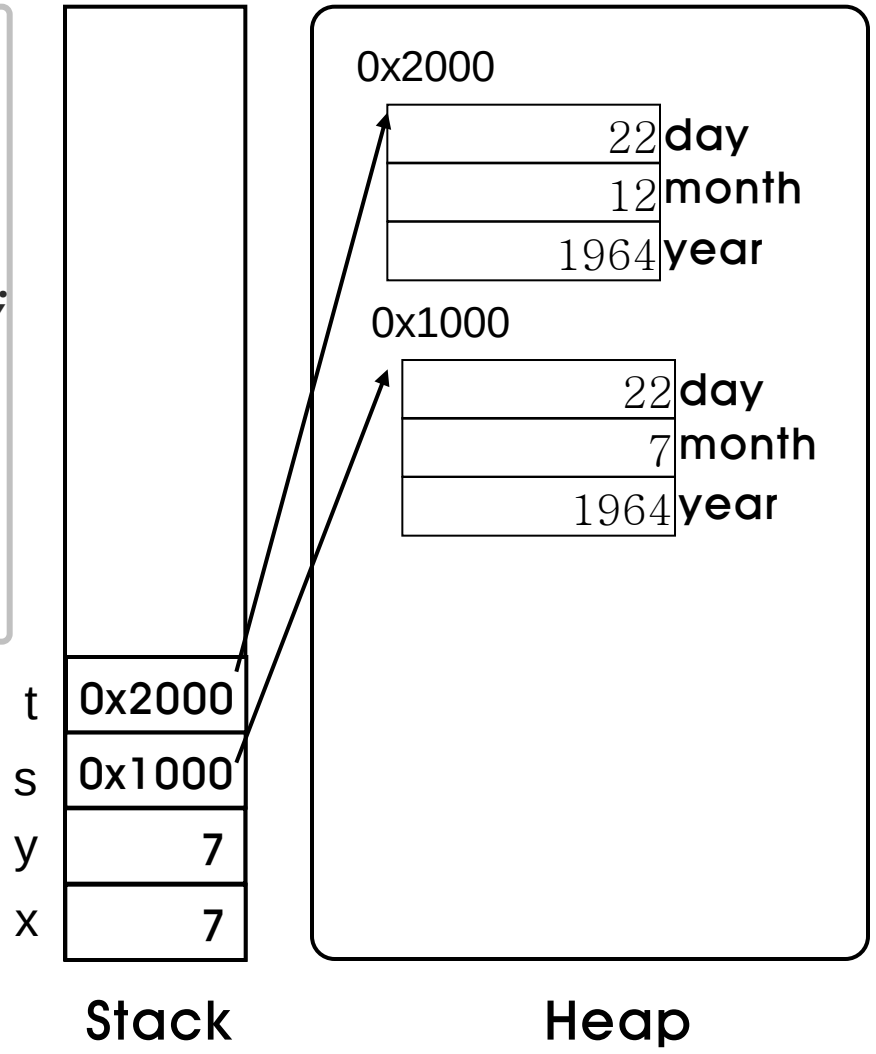
Assigning References


```
int x = 7;

int y = x;

MyDate s = new MyDate( 22, 7, 1964 );

MyDate t = s;

t = new MyDate( 22, 12, 1964 );
```



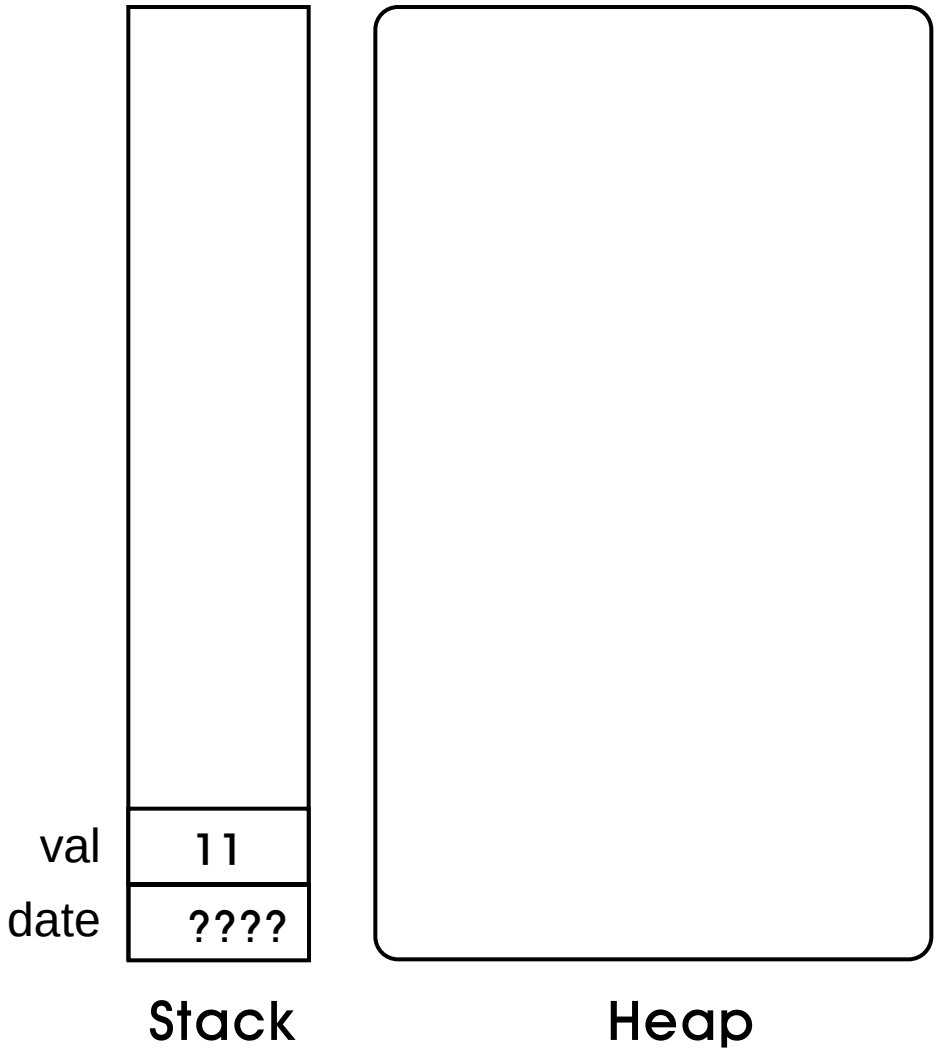
Assigning References


```
int x = 7;

int y = x;

MyDate s = new MyDate( 22, 7, 1964 );

MyDate t = s;

t = new MyDate( 22, 12, 1964 );
```

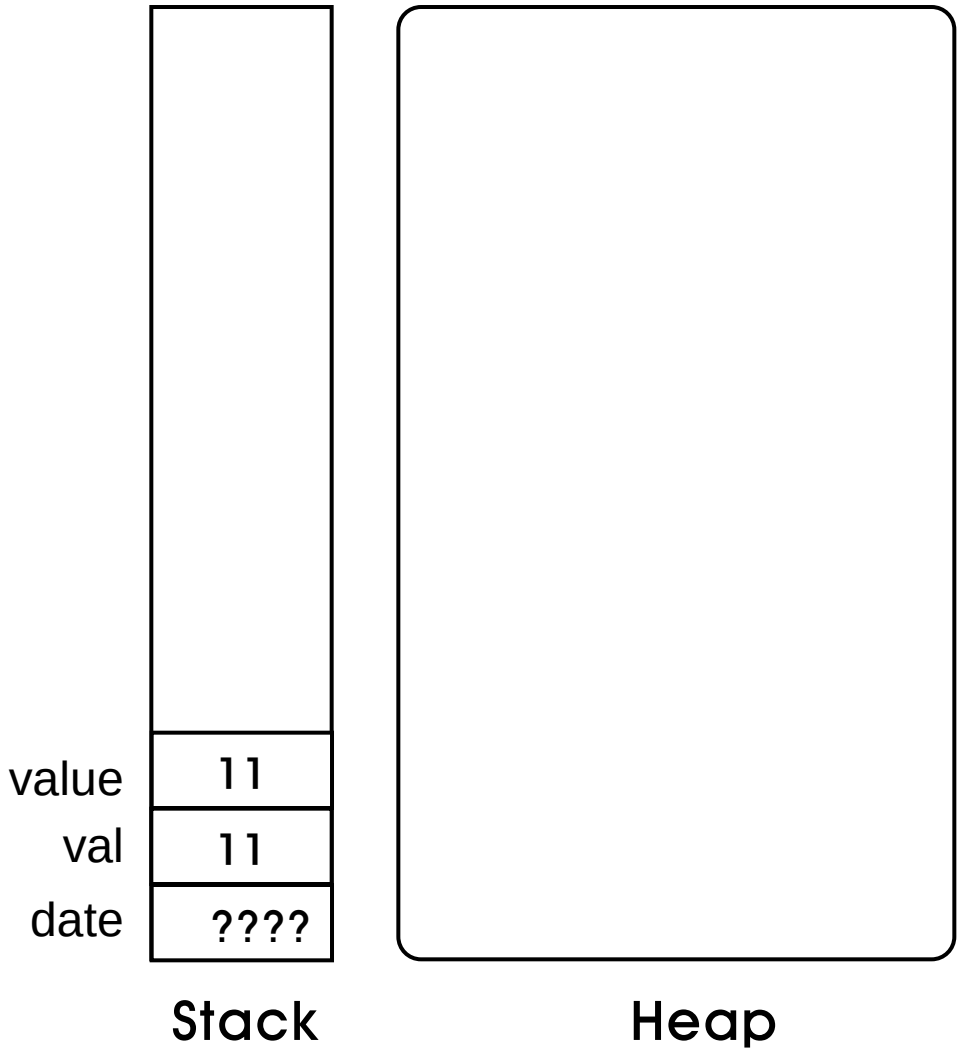


```
public class PassTest {
    public static void changeInt(int value) { // value 공간에 11 전달되어 저장
        value = 55;
    }  // 괄호가 닫히면, value 값은 stack 에서 사라짐

    public static void changeObjectRef(MyDate ref ) { // ref 공간에 주소값 저장
        ref = new MyDate(1,1,2000); // ref에 새로운 주소값이 할당됨
    } // 괄호가 닫히면, ref 값은 stack에서 사라짐

    public static void changeObjectAttr(MyDate ref) { // ref 공간에 주소값 저장
        ref.setDay(4); // ref 가 가리키는 객체의 day값을 4로 변경
    } // 괄호가 닫히면, ref 값은 stack에서 사라짐

    public static void main(String []args) {
        MyDate date;
        int val;
        val = 11;
        changeInt(val);  // val에 저장된 값 11이 전달됨
        System.out.println("Int value is: " + val ); // val 값은 그대로 11 임
        date = new MyDate(22,7,1964);
        changeObjectRef(date); // date 객체의 주소값이 전달됨
        date.print(); // date 객체는 여전히 같은 주소값을 가짐
        changeObjectAttr(date); // date 객체의 주소값이 전달됨
        date.print(); // date 객체의 day 값이 바뀌어져 있음
    }
}
```

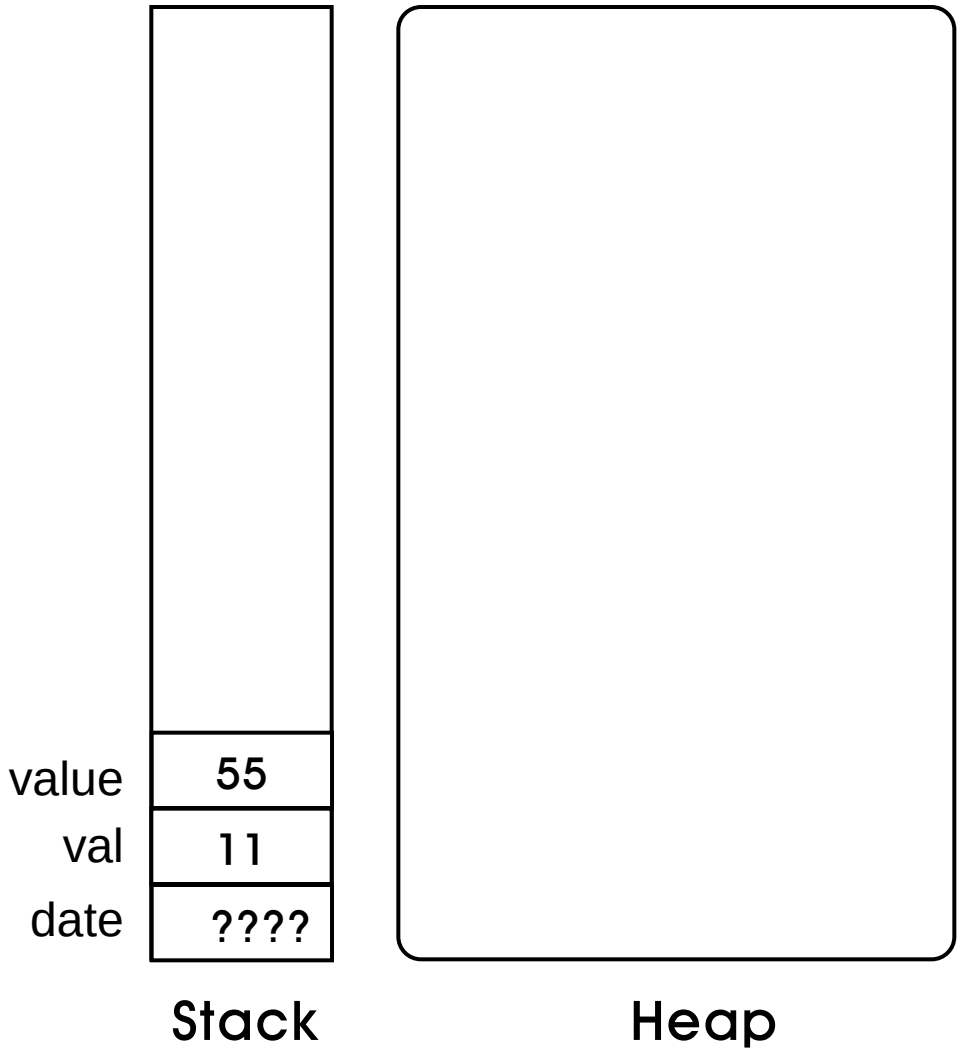
Pass-by-value



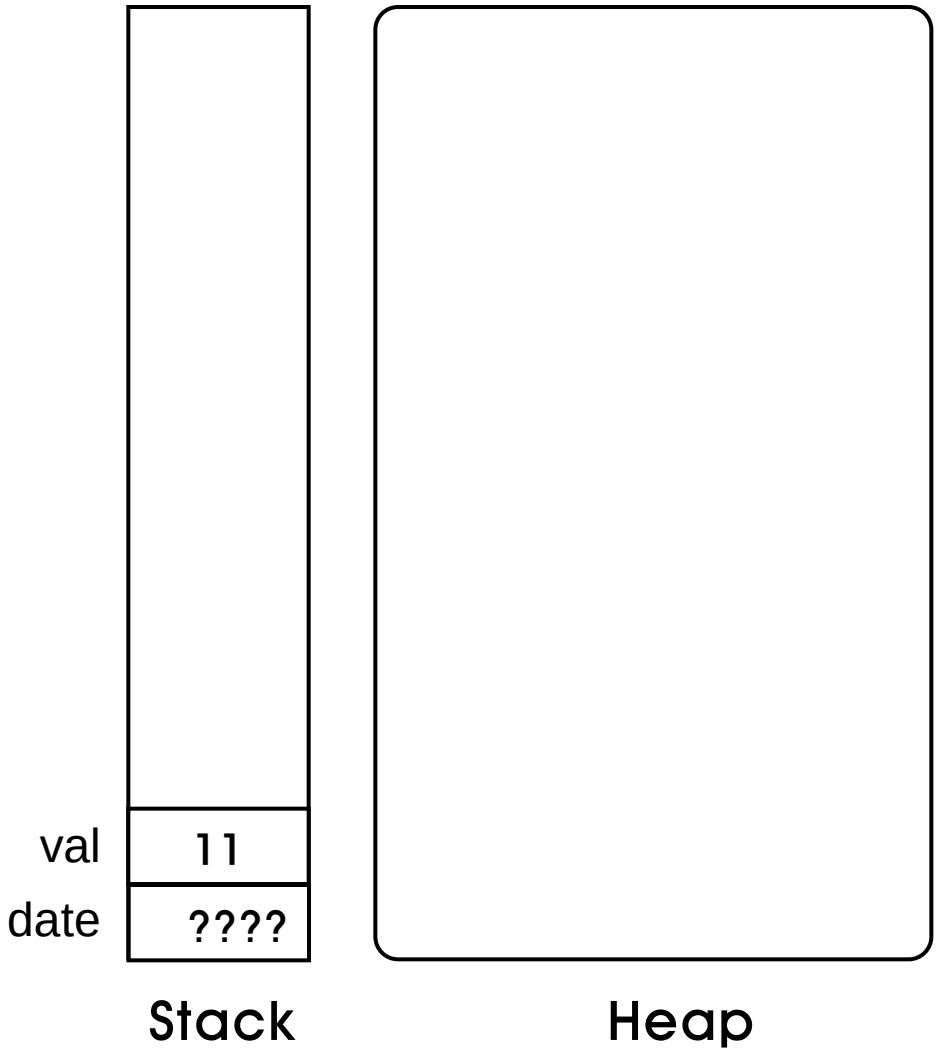
Pass-by-value



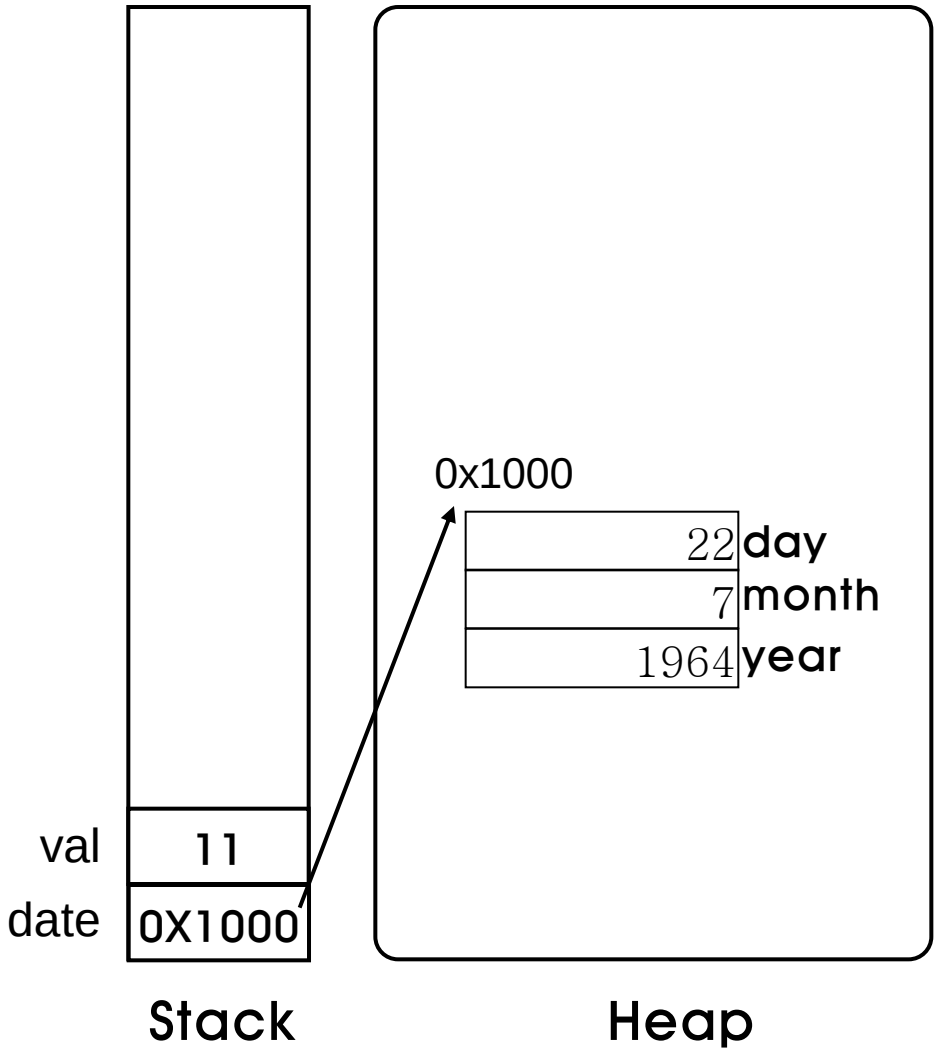
Pass-by-value



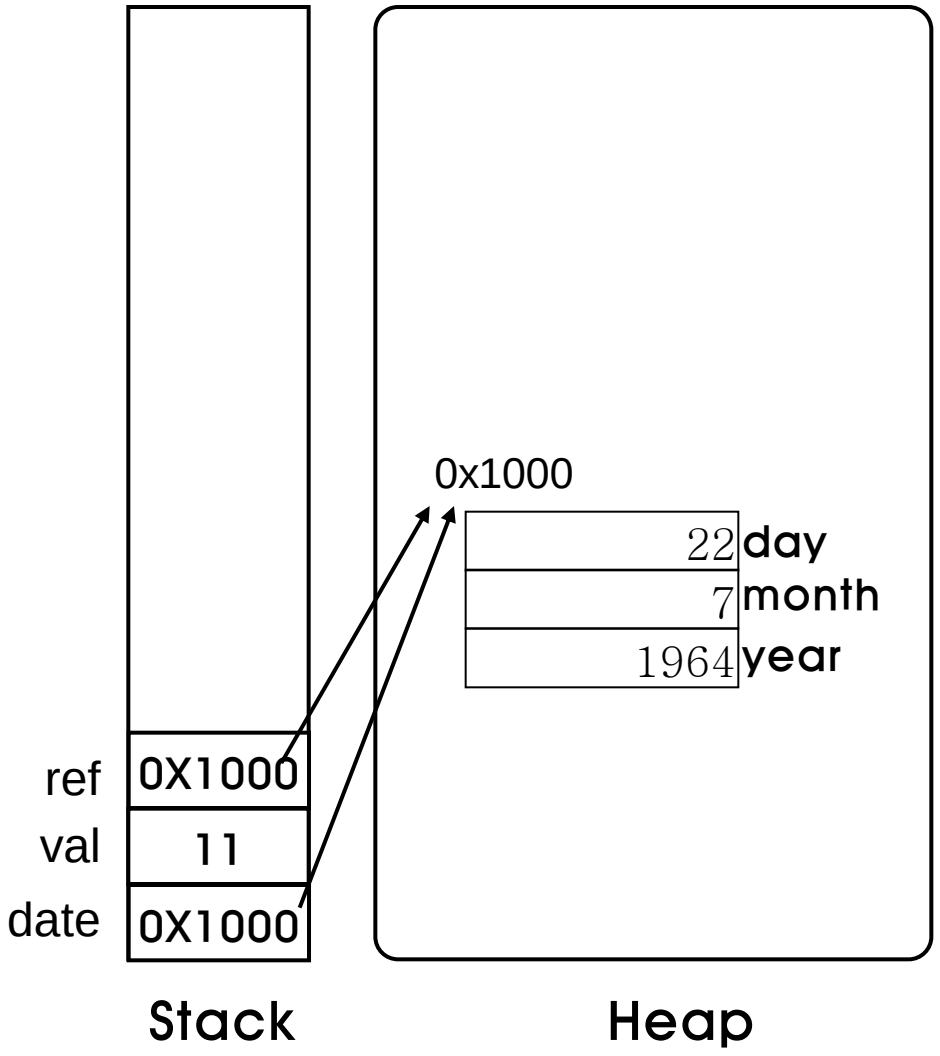
Pass-by-value



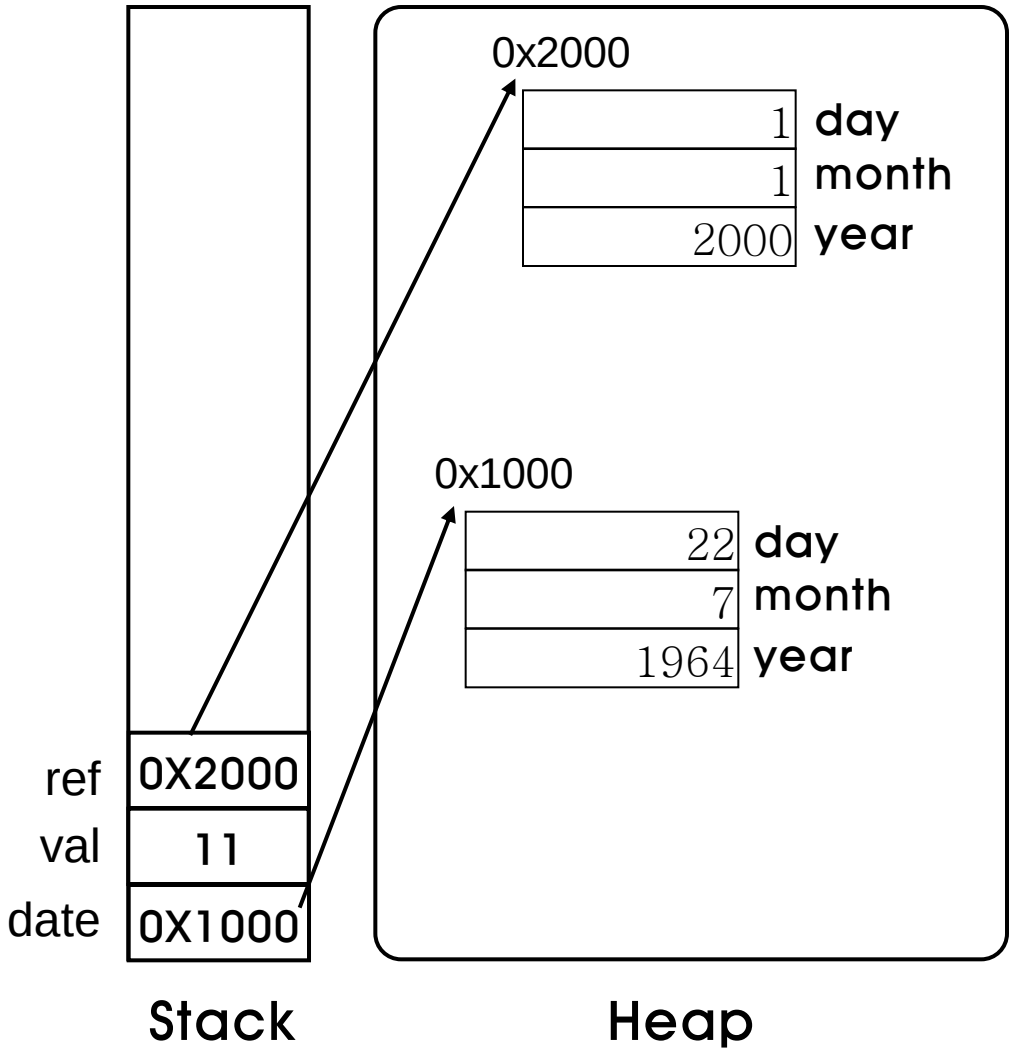
Pass-by-value



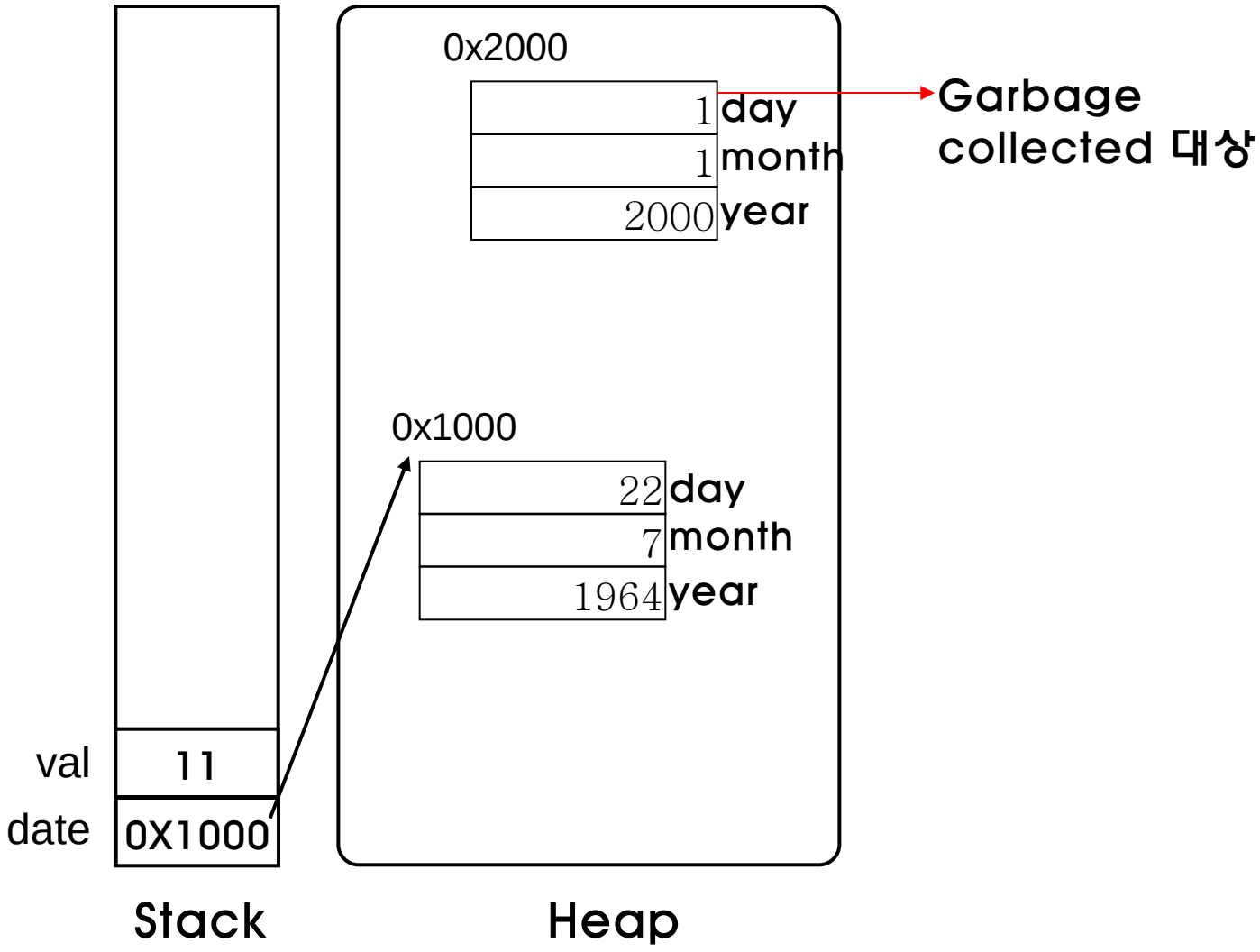
Pass-by-value



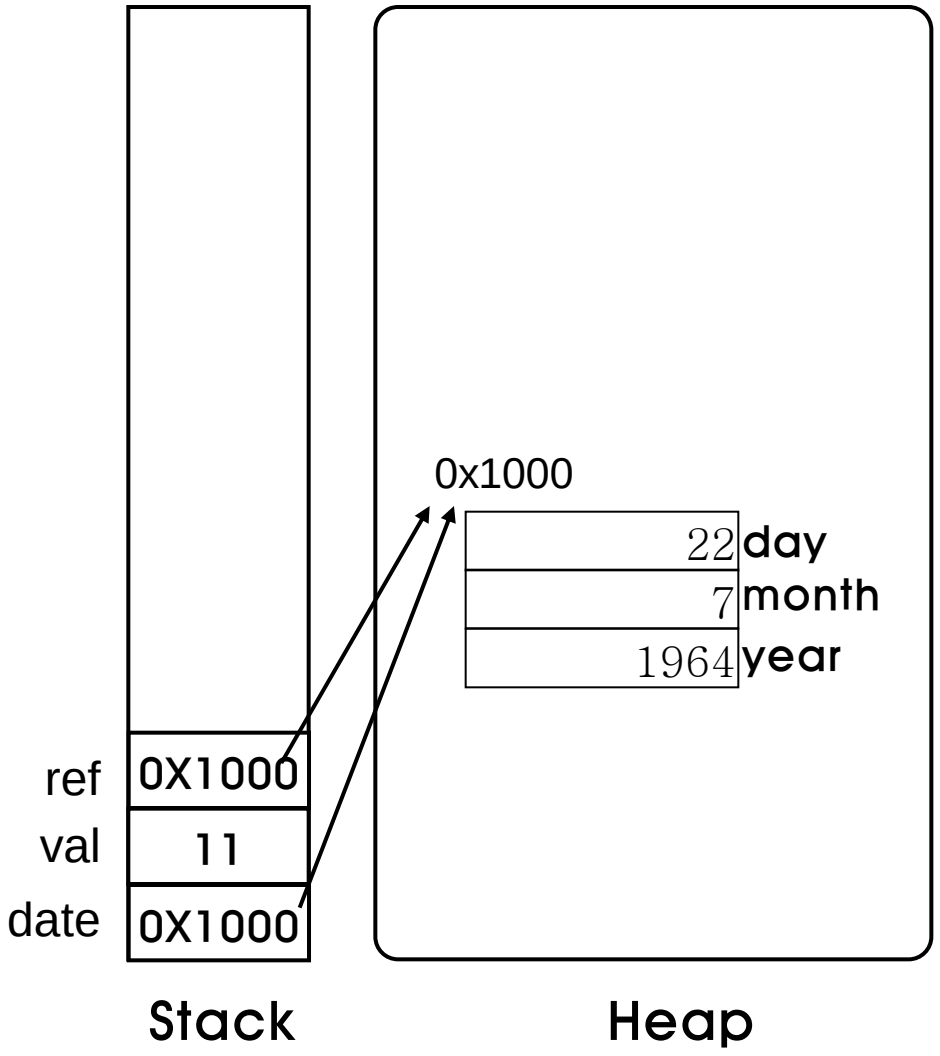
Pass-by-value



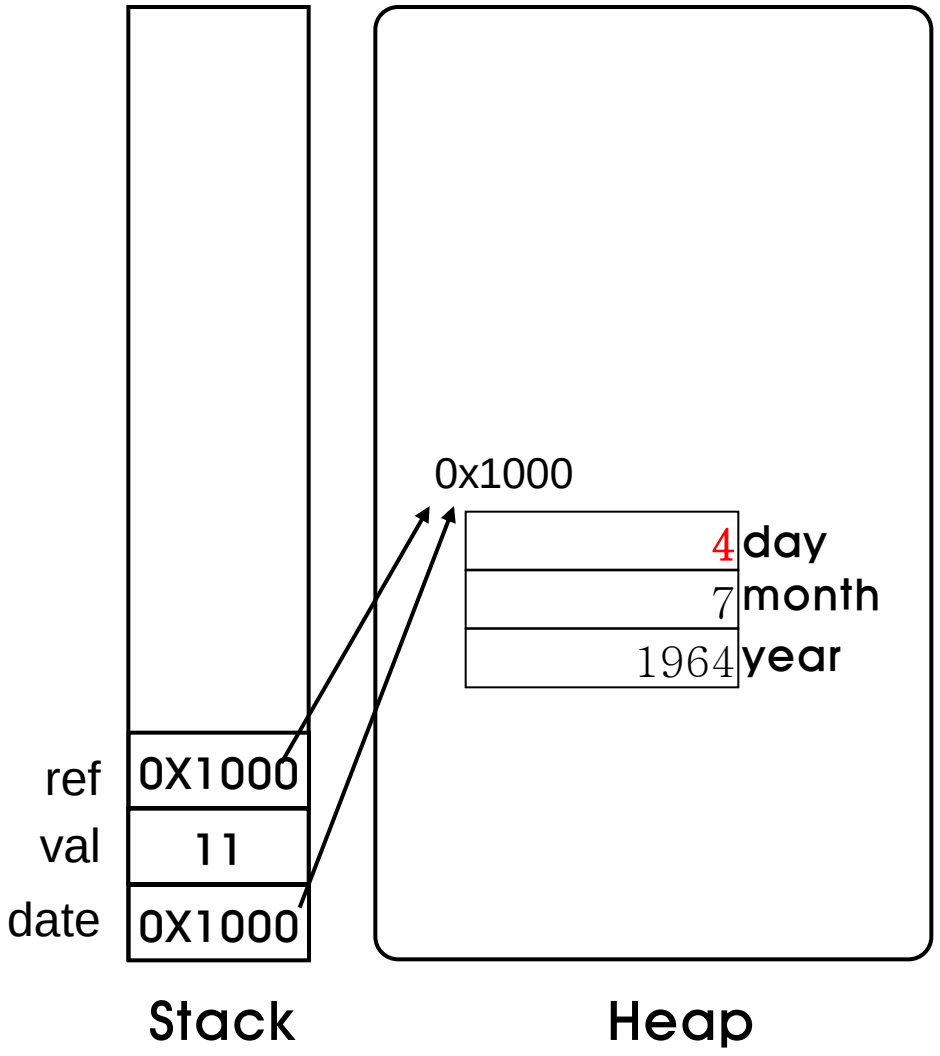
Pass-by-value



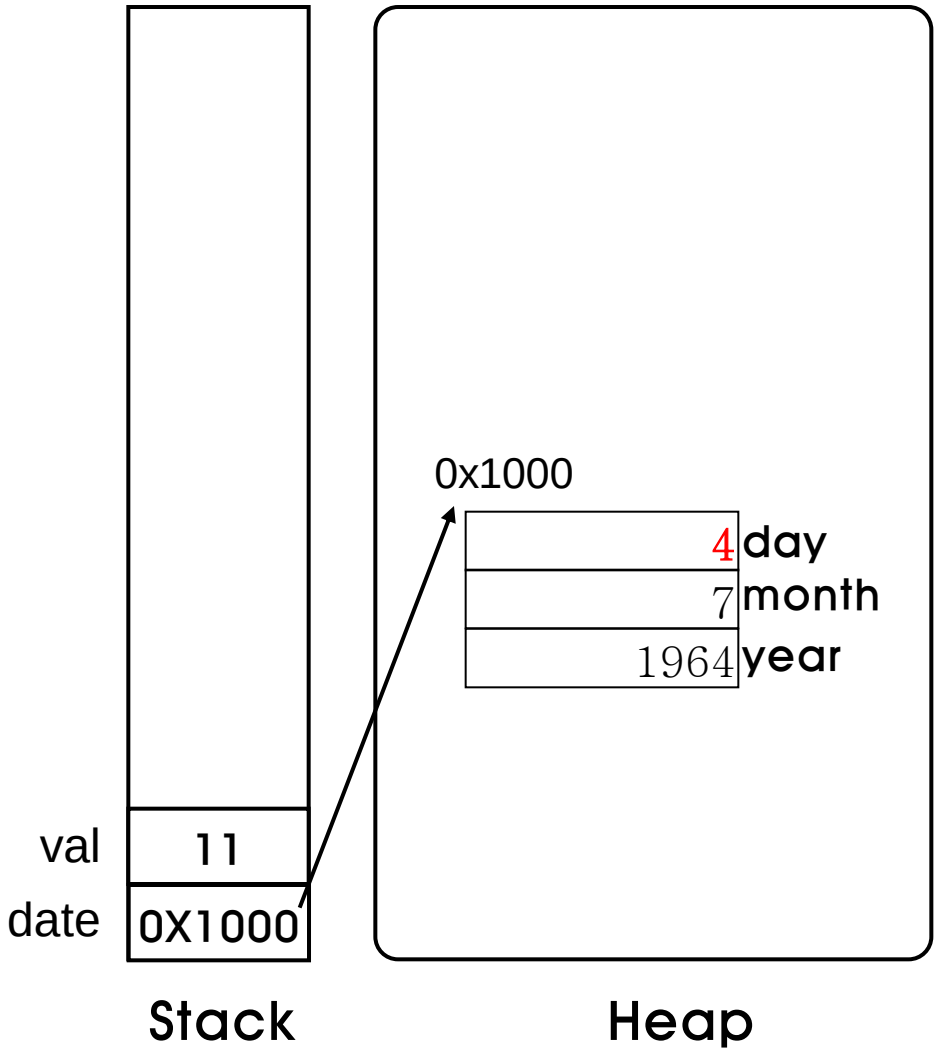
Pass-by-value



Pass-by-value



Pass-by-value



Pass by value– Primitive type

- 메소드 호출 시, parameter로 넘겨 지는 값은

  primitive data type : 실제 값 (변수자체가 아니다.)

  reference data type : 객체의 주소 값(객체 자체가 아니다.)

```
public class PassPrimitive {

    public void passValue() {
        int num = 10;

        change( num );

        System.out.println( num );
    }

    public void change( int num ) {
        num = num + 10;
    }

    public static void main( String[] args ) {
        PassPrimitive pp = new PassPrimitive();

        pp.passValue();
    }
}
```

Pass by value– Reference type


```
public class MyDate {
   private int day;
   private int month;
   private int year;

   public MyDate( int day, int month, int year ) {
      this.day = day;
      this.month = month;
      this.year = year;
   }

   public int getDay() {
      return day;
   }

   public void setDay( int day ) {
      this.day = day;
   }

   public int getMonth() {
      return month;
   }

   public void setMonth( int month )  {
      this.month = month;
   }
   …
```

```
public class TestMyDate {
   public static void main( String [] args ) {
      MyDate today = new MyDate( 22, 7, 1964 );
      TestMyDate test = new TestMyDate();

      today.setDay( 30 );

      test.print( today );
   }
   public void print( MyDate day) {
      System.out.println( "The Day is " +
                          day.getDay() + "-" +
                          day.getMonth() + "-" +
                          day.getYear()
                        );
   }
}
```

Identifier Naming Rule

❑ Class : 대문자로 시작하고, 다음문자부터는 소문자.
여러 단어 조합인 경우, 각 단어의 시작 문자는 대문자.

ex) public class **MemberEntity** { }

❑ Interface : class와 동일

ex) public interface **ConnectionPool** { }

❑ Method : 소문자 시작, 다음 단어부터는 첫 문자만 대문자.

ex) public void **displayMovieInfo( )** { }

❑ Variable (변수) : 소문자 시작, 다음 단어부터는 첫 문자만 대문자.

ex) String **memberName**;

❑ Constant (상수) : 모두 대문자, 단어 사이는 '_' (under bar) 기호로 구분한다.

ex) final int **MAX_NUMBER;**