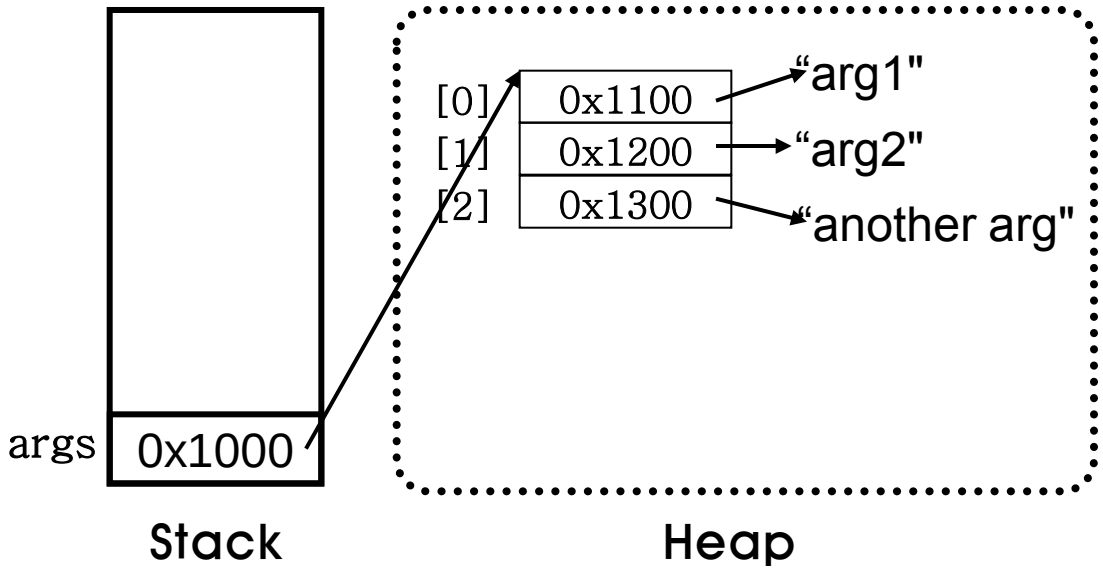
Chap 10. Text-Base Application

학습목표

1. Command-Line Arguments
2. System Properties & Properties Class
3. File Class
4. File Stream I/O → Chap 14
5. String/StringBuffer Class
6. Collection API

```
public class TestArgs {
    public static void main( String[] args ) {
        for ( int inx = 0 ; inx < args.length ; inx++ ) {
            System.out.println( "args[" + inx + "] is '" +
                                args[inx] + "'" );
        }
    }
}
```

```
java TestArgs arg1 arg2 "another arg"
```



System Properties & Property Class

```
import java.util.*;
public class TestProperties {
    public static void main( String[] args ) {
        Properties props = System.getProperties();
        Enumeration prop_names = props.propertyNames();

        while ( prop_names.hasMoreElements() ) {
            String prop_name = (String)prop_names.nextElement();
            String property = props.getProperty( prop_name );
            System.out.println( "property '" + prop_name + "' is '" +
                                property + "'" );
        }
    }

}
```

새로운 Property 추가

```
java -Dname=value  Class_Name
```

❑ 파일 또는 디렉토리에 대한 정보를 제공

```
File f = new File( path );
```

- path : 파일의 경로는 말하는 것으로 파일 또는 디렉토리
- 주의 : File 객체를 생성했다고 해서, 물리적인 파일이 생성되는 것은 아니다.

```
import java.io.*;

public class FileTest {

    public static void main( String[] args ) {

        File f = new File( "d:/error.log" );

        System.out.println( "파일여부: " + f.isFile() );
        System.out.println( "디렉토리여부: " + f.isDirectory() );
        System.out.println( "파일이름: " + f.getName() );
        System.out.println( "파일이 있는 디렉토리: " + f.getParent() );
        System.out.println( "디렉토리 포함 파일이름: " + f.getPath() );
        System.out.println( "파일크기: " + f.length() + " bytes" );
    }
}
```

```
import java.io.*;

public class FileTest {
    public static void main( String[] args ) {
        // File 생성.
        File f2 = new File( "d:/create.txt" );
        try {
            f2.createNewFile();
        } catch ( IOException e ) {
            System.out.println( "에러!!!" );
        }

        // 디렉토리 만들기.
        File f3 = new File( "d:/temp_dir" );
        if ( !f3.exists() ) {
            f3.mkdir();
        } else {
            System.out.println( "이미 존재 " );
        }

        // 디렉토리 리스트 .
        File list = new File( "d:/workshop" );

        String[] str = list.list();
        System.out.println( "다음 디렉토리 내의 파일 리스트입니다: " +
                            list.getPath() );
        for ( int inx = 0 ; inx < str.length ; inx++ ) {
            System.out.println( str[inx] );
        }
    }
}
```

"c:₩data₩text.txt"에서 문자열을 읽기 위한 스트림 작성

File 지정

```
String fileName ="c:/data/text.txt";
혹은 File fileName =  new File( "c:/data/text.txt" );
```

Stream생성

```
BufferedReader br
            = new BufferedReader( new FileReader(fileName) );
```

READ

```
s = br.readLine();  //결과의 null여부로 파일을 다 읽었는지 확인
```

Stream닫기

```
br.close();
```

```
import java.io.*;

public class ReadFile {
    public static void main ( String args[] ) {

        try {
            BufferedReader in
                 = new BufferedReader(new FileReader("c:/data/test.txt"));
            String s;

            while ( ( s = in.readLine() ) != null ) {
                System.out.println( s );
            }

            in.close();
        } catch ( FileNotFoundException e1 ) {
            System.err.println( "File not found: " + "c:/data/test.txt" );
        } catch ( IOException e2 ){
            e2.printStackTrace();
        }
    }
}
```

문자열 관련 클래스

❑ String

- String 객체는 immutable(변경할 수 없는) 하다.
- String 변경 메서드를 호출한다는 것은 새로운 객체를 생성한다는 뜻

❑ StringBuffer

- StringBuffer 객체는 mutable(변경할 수 있는) 하다.
- Buffer를 두어 문자열 연산을 하여, 빠르다.

String 클래스 – Example

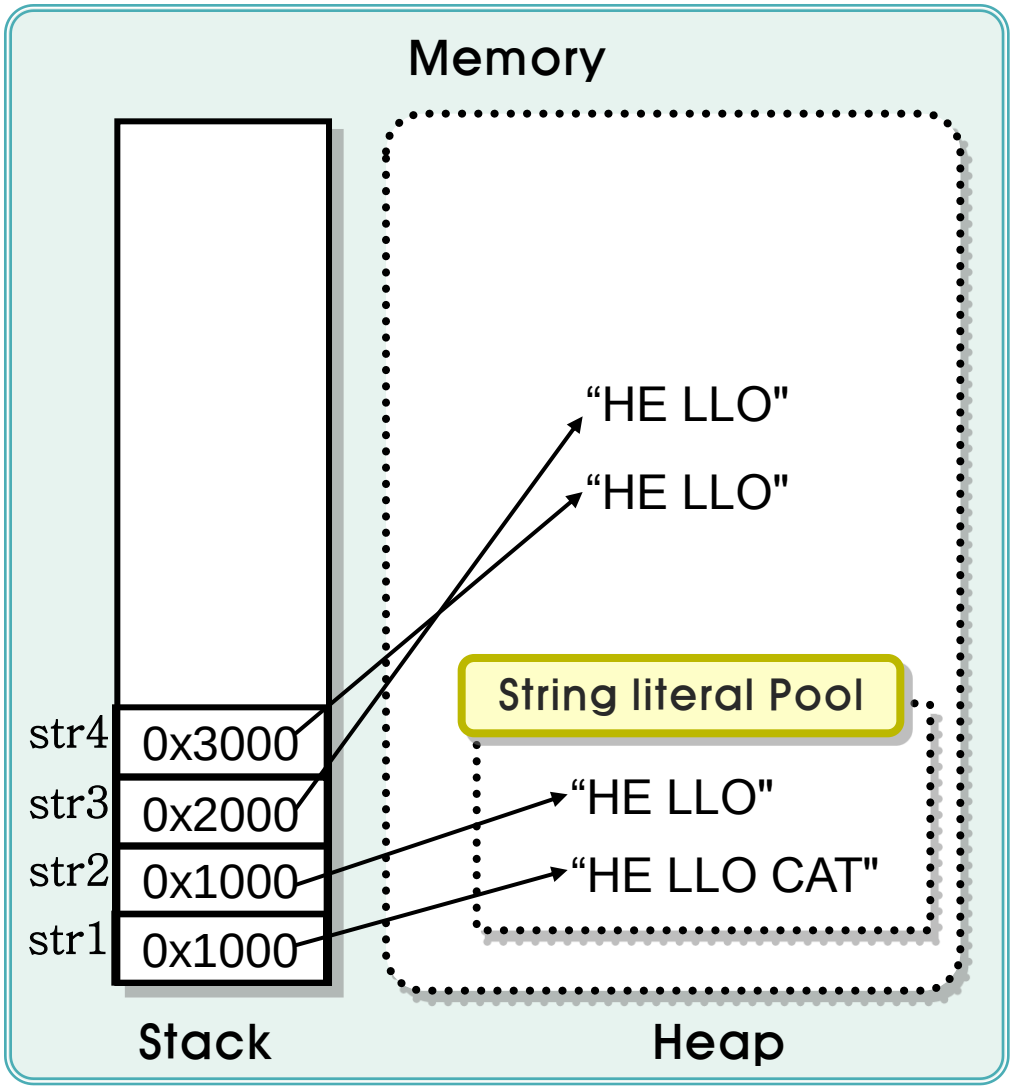
```
public class TestString {
    public static void main( String[] args ) {
        //           01234567890123456789
        String s = "HE LLO JAVA World";

        System.out.println( s.substring( 3, 6 ) );
        System.out.println( s.indexOf( "J", 4 ) );
        System.out.println( s.charAt(14) );
        s.concat( " Student" );
        System.out.println( s );
    }
}
```

StringBuffer 클래스 – Example


```
public class TestStringBuffer {
      public static void main( String[] args ) {
            StringBuffer sb = new StringBuffer( "ROM" );
            System.out.println( sb );
            System.out.println( sb.append( "A" ) );
            System.out.println( sb.insert( 3, "R" ) );
            System.out.println( sb.reverse() );
      }
}
```

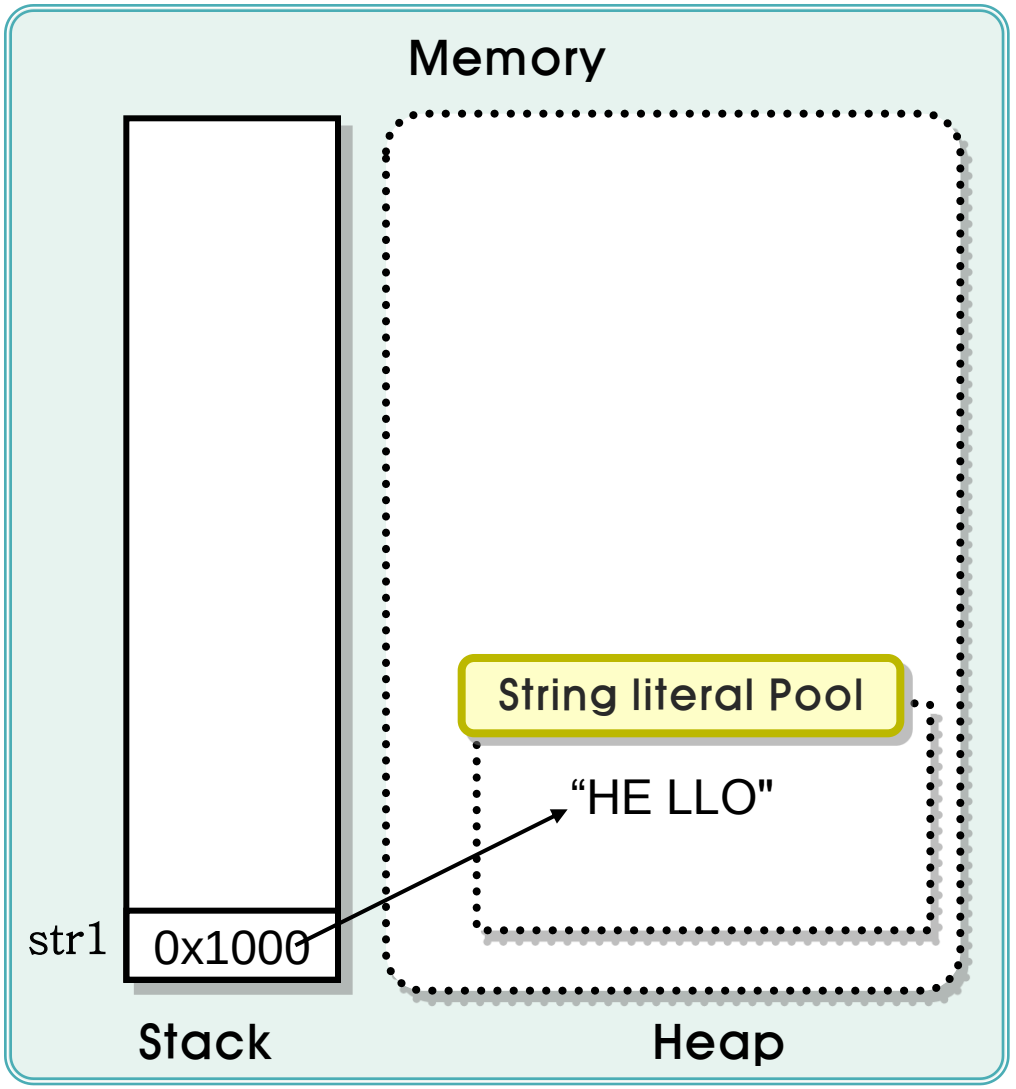
String 객체 메모리 할당 영역

```
public class StringStringBuffer {
    public static void main( String[] args ) {
        String str1 = “HE LLO";
        String str2 = “HE LLO";
        String str3 = new String( “HE LLO" );
        String str4 = new String( “HE LLO" );
        String str1 = str1 + “ CAT”;
    }
}
```



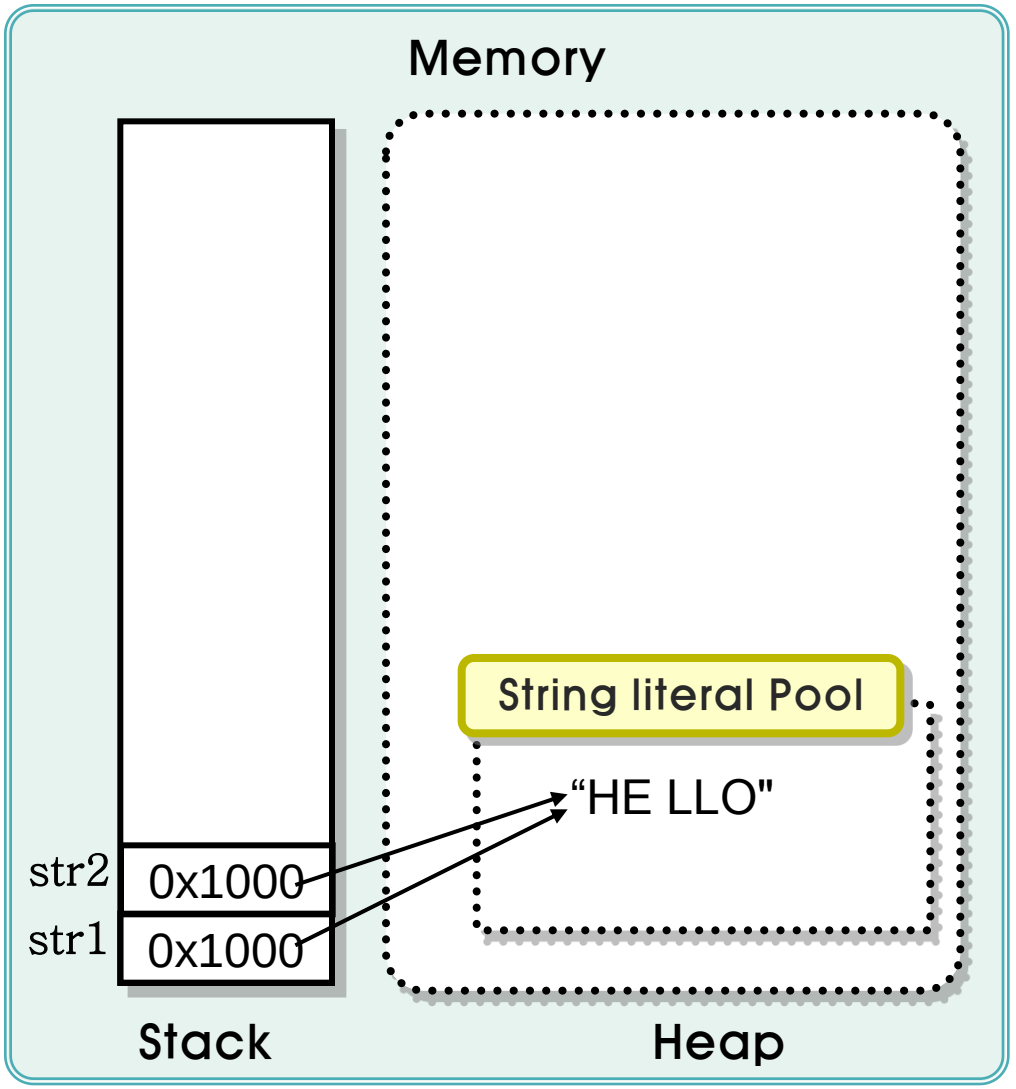
String 객체 메모리 할당 영역

```
public class StringStringBuffer {
    public static void main( String[] args ) {
        String str1 = “HE LLO";
        String str2 = “HE LLO";
        String str3 = new String( “HE LLO" );
        String str4 = new String( “HE LLO" );
        String str1 = str1 + “ CAT”;
    }
}
```



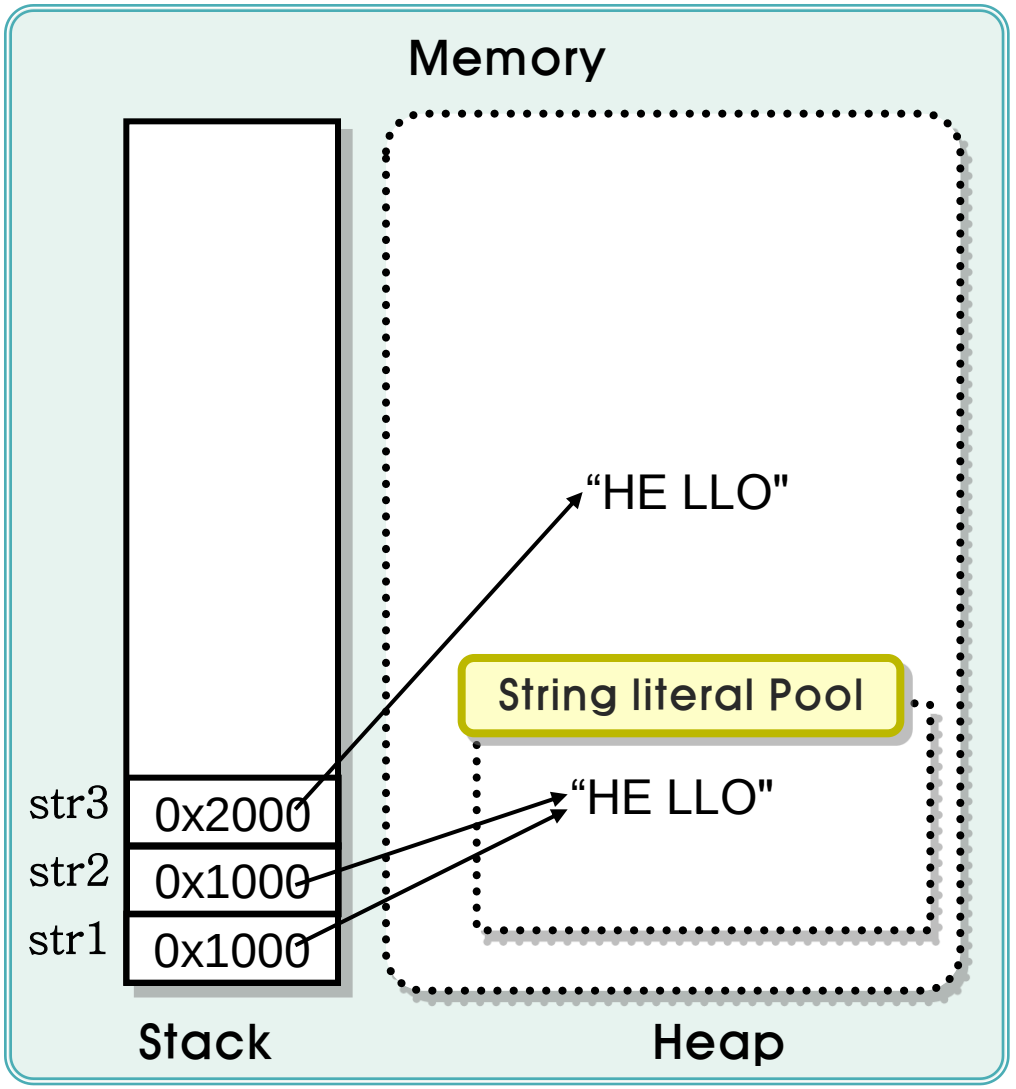
String 객체 메모리 할당 영역

```
public class StringStringBuffer {
    public static void main( String[] args ) {
        String str1 = “HE LLO";
        String str2 = “HE LLO";
        String str3 = new String( “HE LLO" );
        String str4 = new String( “HE LLO" );
        String str1 = str1 + “ CAT”;
    }
}
```



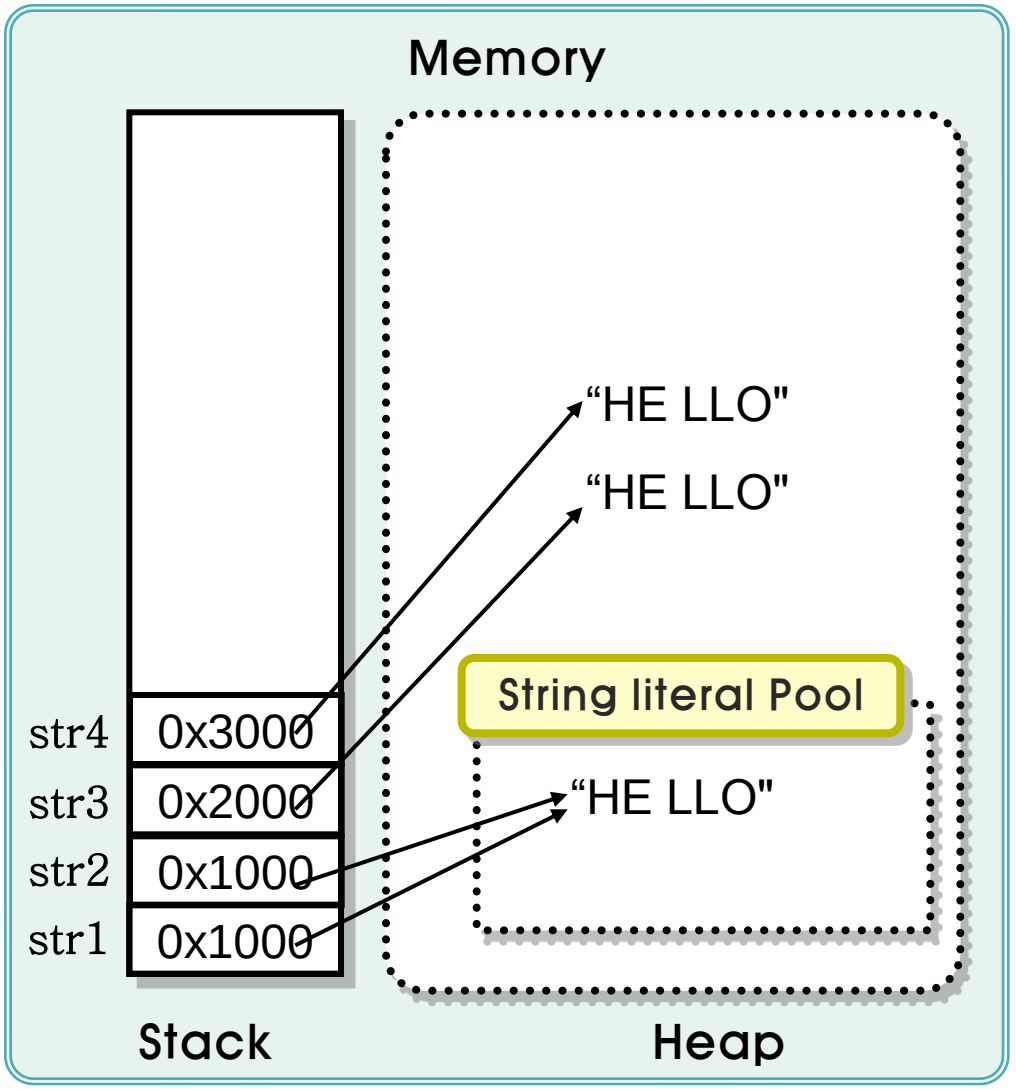
String 객체 메모리 할당 영역

```
public class StringStringBuffer {
    public static void main( String[] args ) {
        String str1 = “HE LLO";
        String str2 = “HE LLO";
        String str3 = new String( “HE LLO" );
        String str4 = new String( “HE LLO" );
        String str1 = str1 + “ VCC”;
    }
}
```



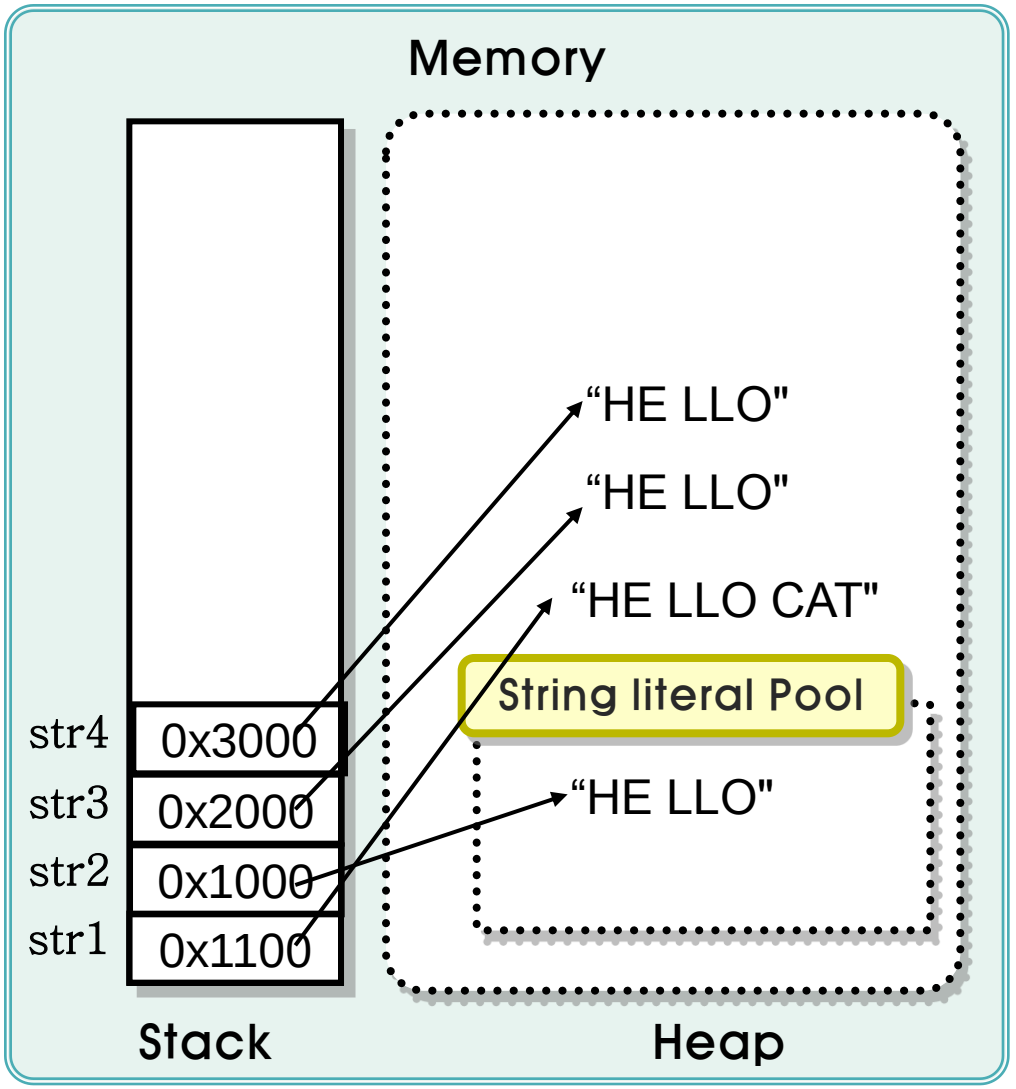
String 객체 메모리 할당 영역

```
public class StringStringBuffer {
    public static void main( String[] args ) {
        String str1 = “HE LLO";
        String str2 = “HE LLO";
        String str3 = new String( “HE LLO" );
        String str4 = new String( “HE LLO" );
        String str1 = str1 + “ CAT”;
    }
}
```



String 객체 메모리 할당 영역

```
public class StringStringBuffer {
    public static void main( String[] args ) {
        String str1 = “HE LLO";
        String str2 = “HE LLO";
        String str3 = new String( “HE LLO" );
        String str4 = new String( “HE LLO" );
        String str1 = str1 + “ CAT”;
    }
}
```



String 문자열 비교

```
String str1 = "HELLO";
String str2 = "HELLO";
String str3 = new String("HELLO" );
String str4 = new String( "HELLO" );

if ( str1 == str2 ) {
    System.out.println( "str1 과 str2는 같은 객체이다" );
} else {
    System.out.println( "str1 과 str2는 다른 객체이다" );
}

if ( str2 == str3 ) {
    System.out.println( "str2 과 str3는 같은 객체이다" );
} else {
    System.out.println( "str2 과 str3는 다른 객체이다" );
}

if ( str3 == str4 ) {
    System.out.println( "str3 과 str4는 같은 객체이다" );
} else {
    System.out.println( "str3 과 str4는 다른 객체이다" );
}

if ( str1.equals( str4 ) ) {
    System.out.println( "str1 과 str4는 같은 문자열이다" );
} else {
    System.out.println( "str1 과 str4는 다른 문자열이다" );
}
```

문자열 비교시 주의점

- 문자열 비교는 반드시 equals() 메소드를 사용할 것
- == 연산자 비교의 사용은 하지 말 것!!!
- 문자열 비교는 String ref변수가 null 이 아닌 경우에 equals 메소드를 호출 해야 하기 때문에, null 체크를 해 주는 습관을 들이자.

```
if ( str1 != null && str1.equals( str2 ) ) {
    …
}
```

StringBuffer 문자열 비교

```
StringBuffer sb1 = new StringBuffer("HELLO" );
StringBuffer sb2 = new StringBuffer( "HELLO" );

if ( sb1.equals( sb2 ) ) {
    System.out.println( "sb1 와 sb2은 같은 문자열이다" );
} else {
    System.out.println( "sb1 와 sb2은 다른 문자열이다" );
}

if ( sb1.toString().equals( sb2.toString() ) ) {
    System.out.println( "sb1 와 sb2은 같은 문자열이다" );
} else {
    System.out.println( "sb1 와 sb2은 다른 문자열이다" );
}
```

StringBuffer와 String 문자열 비교

```
StringBuffer sb1 = new StringBuffer( “HELLO" );
StringBuffer sb2 = new StringBuffer( “HELLO" );
String str = “HELLO";

if ( str.equals( sb1 ) ) {
    System.out.println( "sb 와 str은 같은 문자열이다" );
} else {
    System.out.println( "sb 와 str은 다른 문자열이다" );
}

if ( str.equals( sb1.toString() ) ) {
    System.out.println( "sb 와 str은 같은 문자열이다" );
} else {
    System.out.println( "sb 와 str은 다른 문자열이다" );
}

if ( str.contentEquals( sb1 ) ) {
    System.out.println( "sb 와 str은 같은 문자열이다" );
} else {
    System.out.println( "sb 와 str은 다른 문자열이다" );
}
```

StringBuffer 문자열 비교시 주의점

- 문자열 비교는 반드시 toSring()를 이용하여, String으로 변환 후, equals() 사용한다.

```
if ( sb1.toString().equals(sb2.toString()) ) {

}
```

```
import java.util.*;

public class StringTokenizerTest {

    public static void main( String[] args ) {

        String str = "하하|호호|후후";

        StringTokenizer st = new StringTokenizer( str, "|" );

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

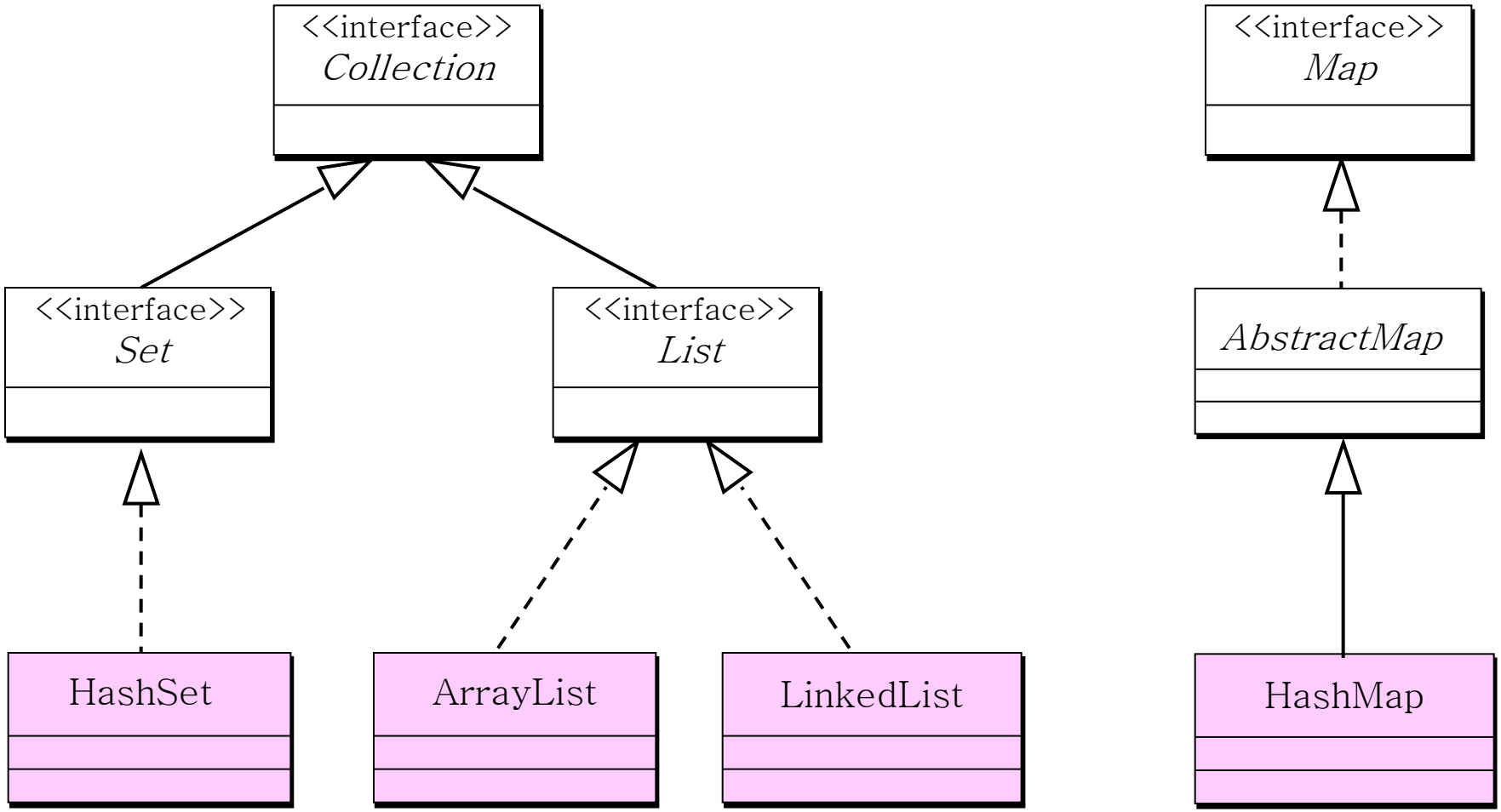
```
import java.util.Date;
import java.text.SimpleDateFormat;

public class ForamttingDate {

    public static void main( String[] args ) {
        Date today = new Date();
        System.out.println( today );

        SimpleDateFormat ft = new SimpleDateFormat( "yyyy-MM-dd‟ );
        String formattedDate = ft.format(new Date());
        System.out.println( formattedDate );
    }

}
```

❑ 배열

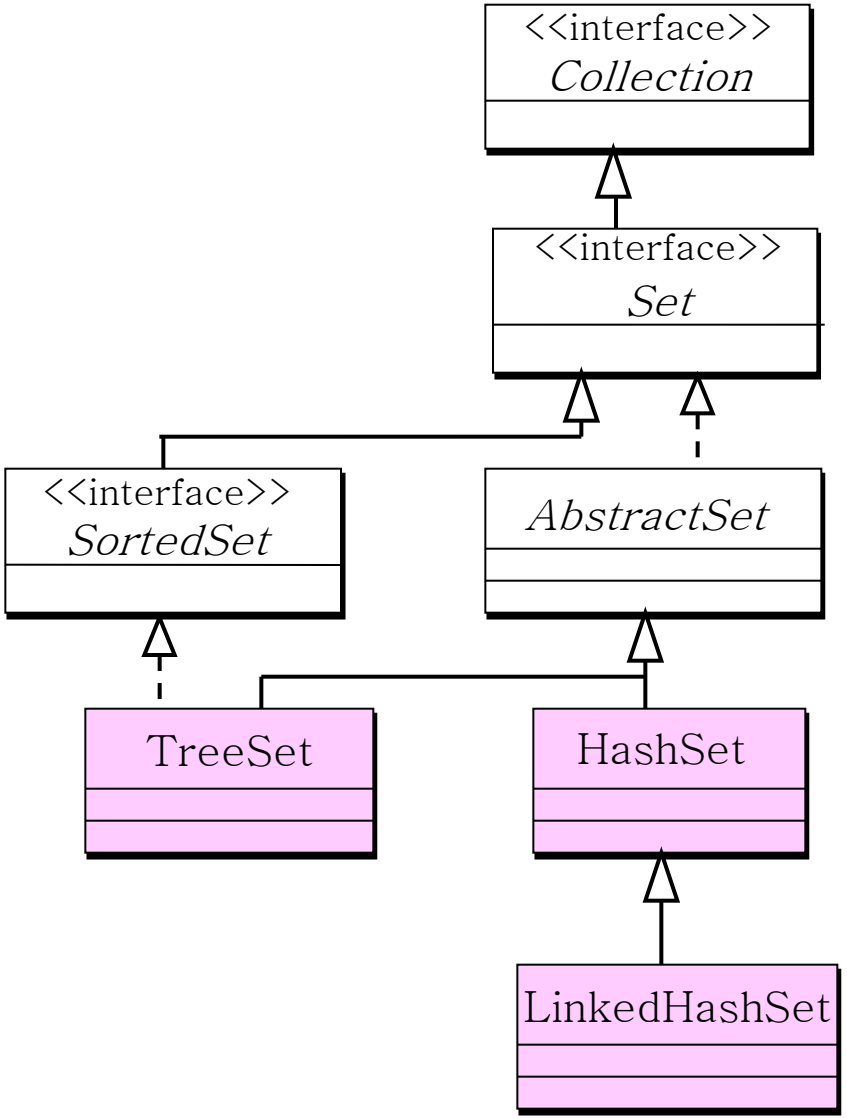
- 장점 : 객체를 저장하고, 검색하는데 가장 효율적이다, 사용하기 편하다.
- 단점 : 사이즈 변경이 불가하다.

❑ Collection

- 객체를 저장할 때 마다, 크기를 자동으로 늘려 준다.
- Set 계열 : 중복을 허용하지 않고, 추가되는 순서를 유지하지 않는다.
- List 계열 : 중복을 허용하고, 추가되는 순서를 유지한다.
- Map 계열 : 키와 값의 쌍으로 저장 된다. (키와 값 모두 객체여야 한다.)
- java.util package에 있다.


<<interface>>
Collection
<<interface>>
Set
<<interface>>
List
HashSet
ArrayList
LinkedList
<<interface>>
Map
AbstractMap
HashMap

Collections – Set 계열



❑ HashSet

- Set에 객체를 저장하는데 Hash를 사용하여 처리 속도가 빠르다.
- hashCode() 및 equals()를 재정의 해야 한다.
- 오직 하나의 Null객체를 가질 수 있다.

❑ LinkedHashSet

- HashSet과 거의 같다. 차이점은 Set에 추가되는 순서를 유지한다는 점.
- JDK1.4에서 추가

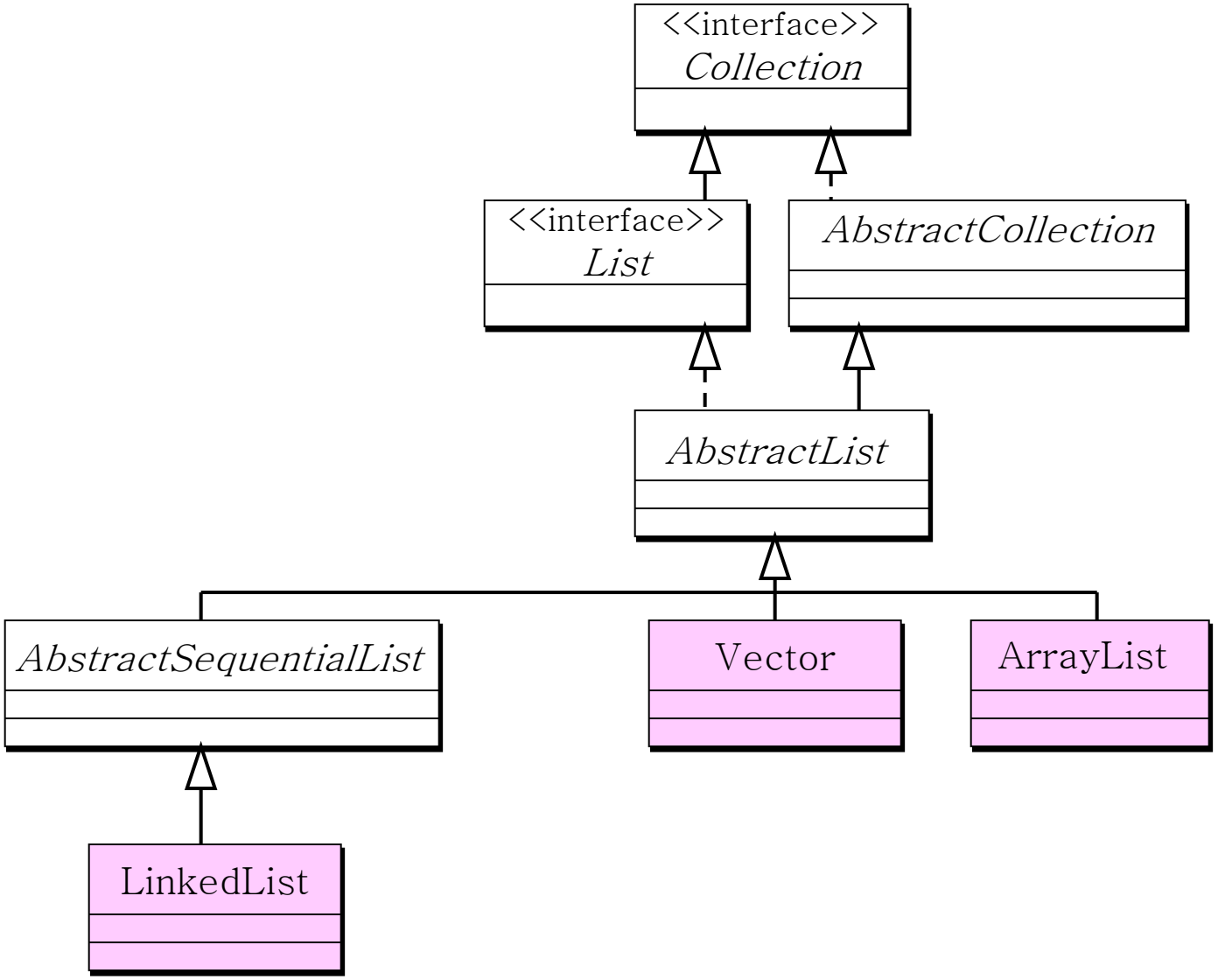
❑ TreeSet

- 객체의 Hash값에 의한 오름차순의 정렬 유지

Collections – Set Example


```
import java.util.*;
public class SetExample {
    public static void main( String[] args ) {
        Set set = new HashSet();
        set.add( "add" );
        set.add( "second" );
        set.add( "3rd" );
        set.add( new Integer(4) );
        set.add( new Float(5.0F) );
        set.add( "second" );
        set.add( new Integer(4) );
        System.out.println( set );

        set.remove( "second" );
        set.remove( new Float(5.0F) );
        System.out.println( set );
    }
}
```

Collections – List 계열



❑ ArrayList

- List에서 객체를 얻어내는데 효율적

❑ LinkedList

- List에서 데이터를 삽입하거나 삭제하는데 효율적

❑ Vector

- List객체들 중에서, 가장 성능이 좋지 않다.
- 동기화(Synchronization)를 제공한다.

Collections – List Example

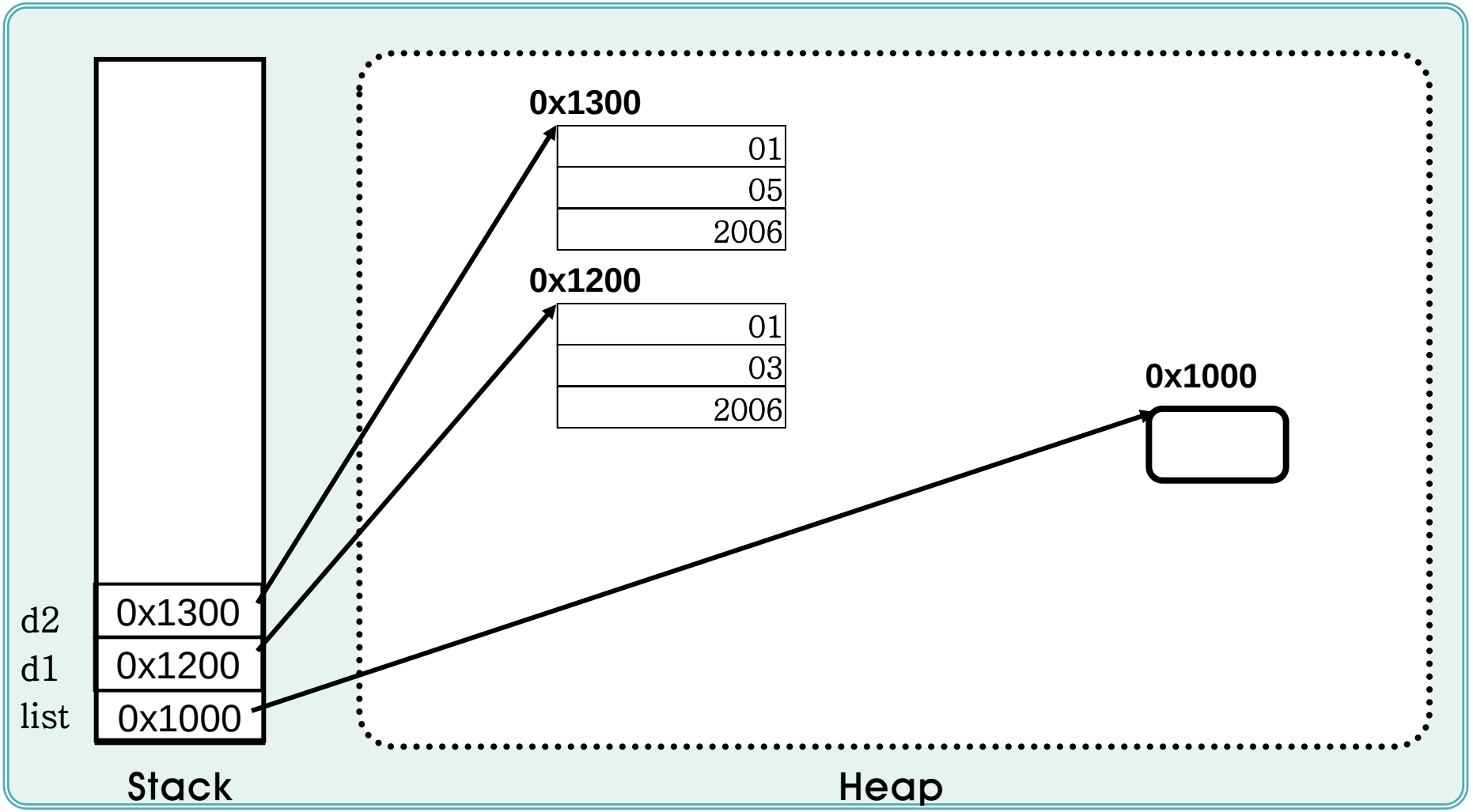

```
import java.util.*;
public class ListExample {
    public static void main( String[] args ) {
        List list = new ArrayList();
        list.add( "add" );
        list.add( "second" );
        list.add( "3rd" );
        list.add( new Integer(4) );
        list.add( new Float(5.0F) );
        list.add( "second" );
        list.add( new Integer(4) );
        System.out.println( list );

        list.remove( "second" );
        list.remove( new Float(5.0F) );
        System.out.println( list );
    }
}
```

Collections – List Example


```
import java.util.ArrayList;

public class ListTest {
    public static void main( String[] args ) {
        List list = new ArrayList();
        MyDate d1 = new MyDate(2006,03,01 );
        MyDate d2 = new MyDate(2006,05,01 );

        System.out.println( "d1:"+ d1 );
        System.out.println( "d2:"+ d2 );

        list.add(d1);
        list.add(d1);
        list.add(d2);

        d1.setDay(30);

        MyDate d11 = (MyDate)list.get(0);
        MyDate d12 = (MyDate)list.get(1);

        System.out.println( "d11.getDay():"+d11.getDay() );
        System.out.println( "d12.getDay():"+d12.getDay() );
        System.out.println( "d2:" + (MyDate)list.get(2));
    }
}
```

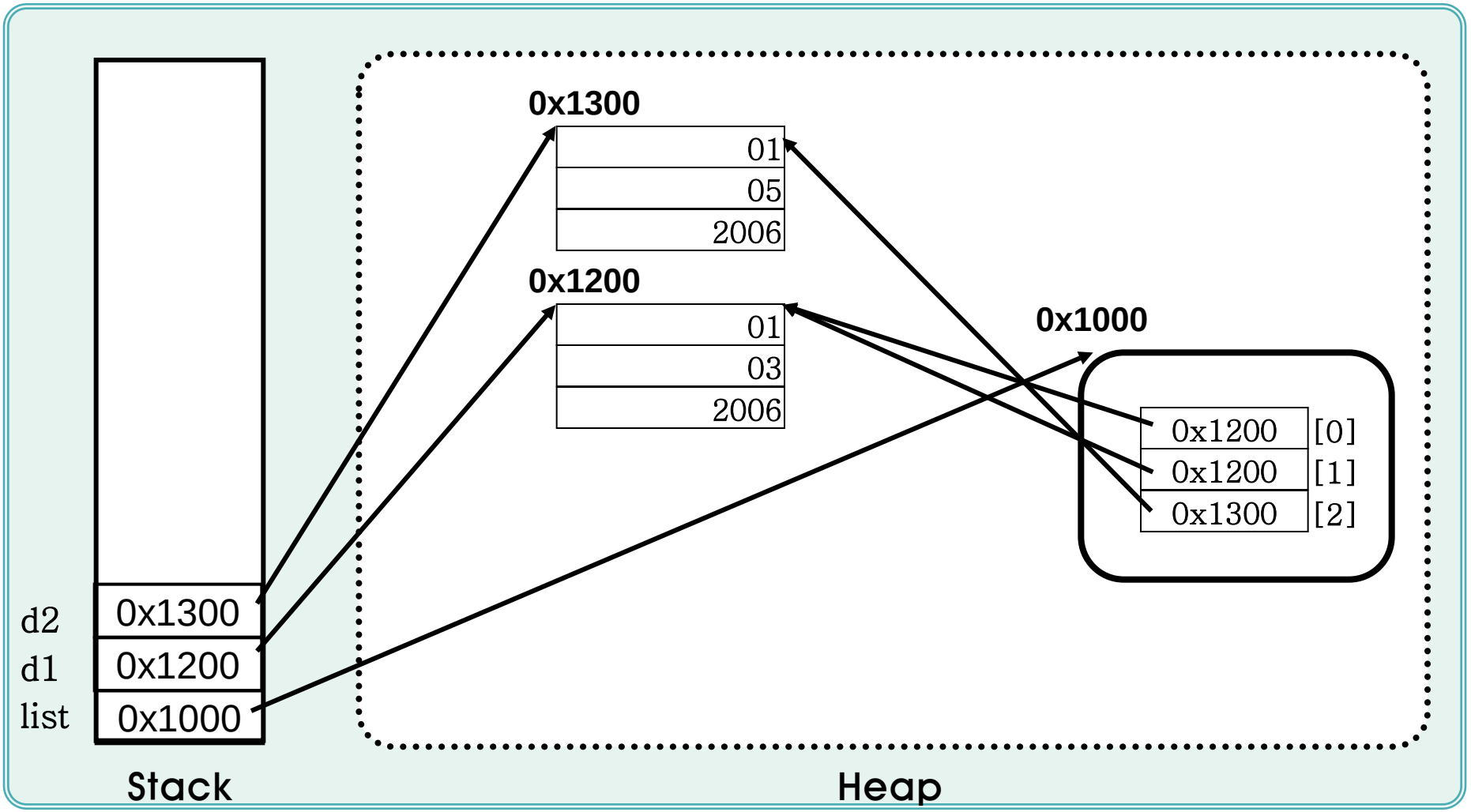
Collections – Vector

```
List list = new ArrayList();
MyDate d1 = new MyDate(2006,03,01 );
MyDate d2 = new MyDate(2006,05,01 );
```



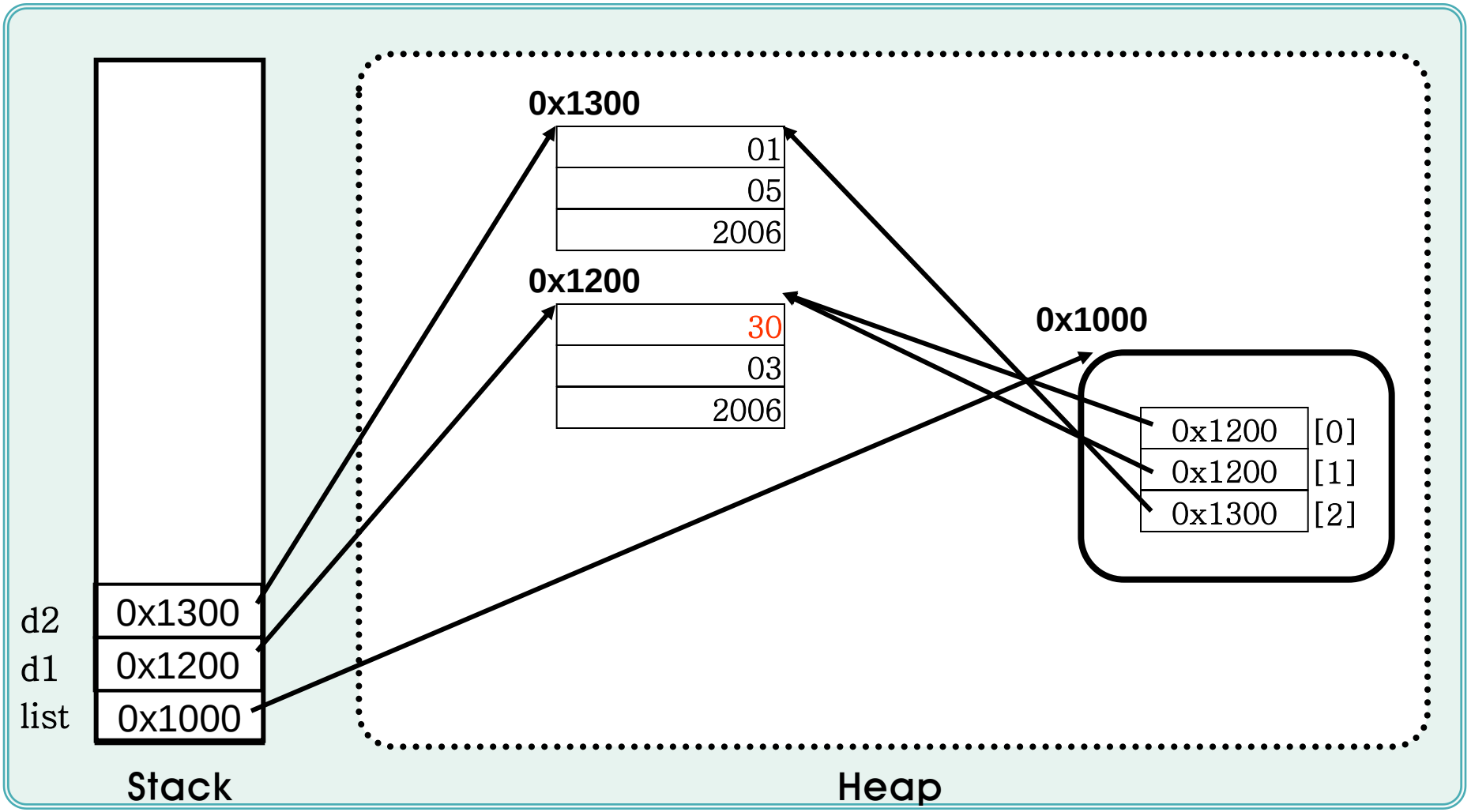
Collections – Vector

```
list.add(d1);
list.add(d1);
list.add(d2);
```



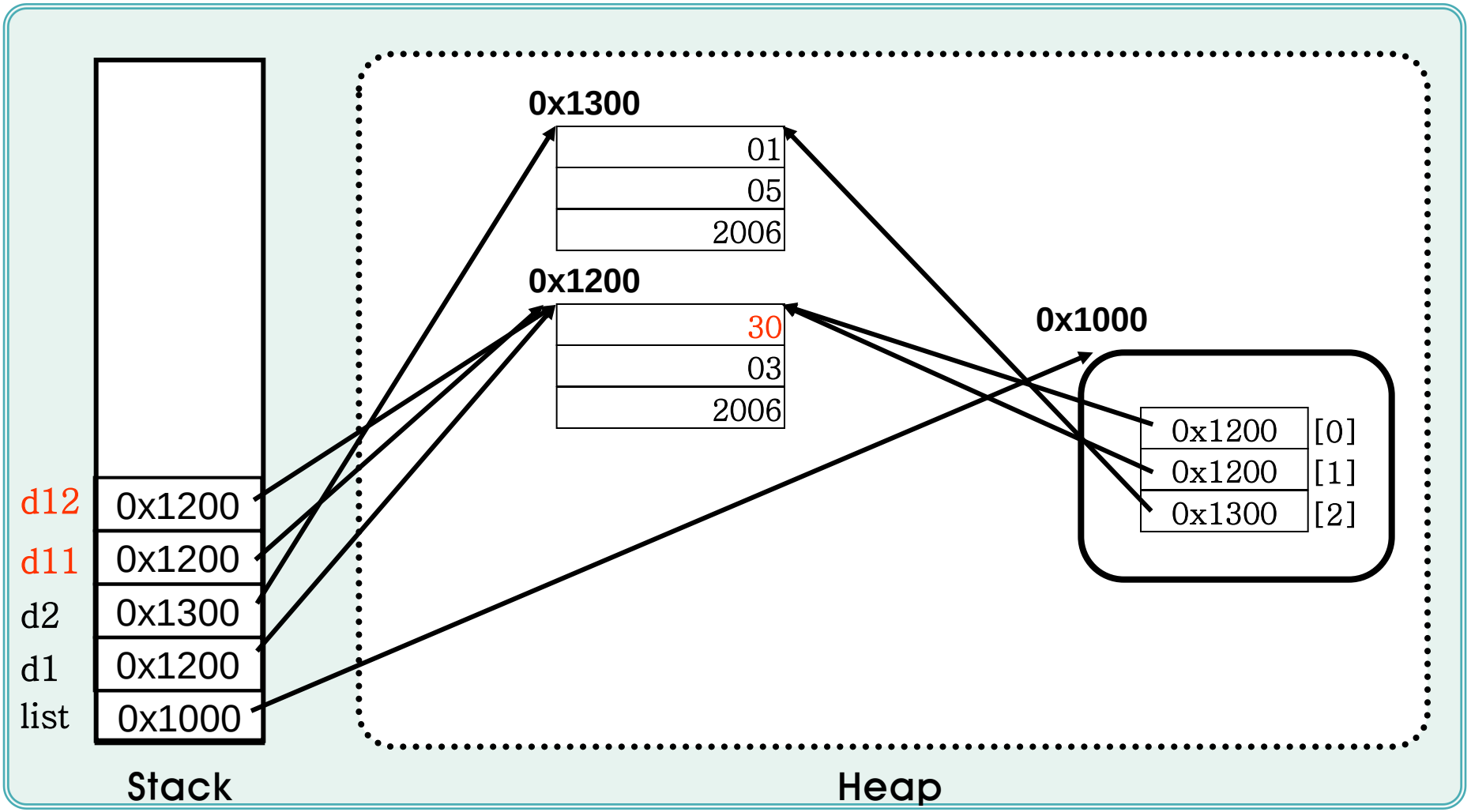
Collections – Vector

```
d1.setDay(30);
```

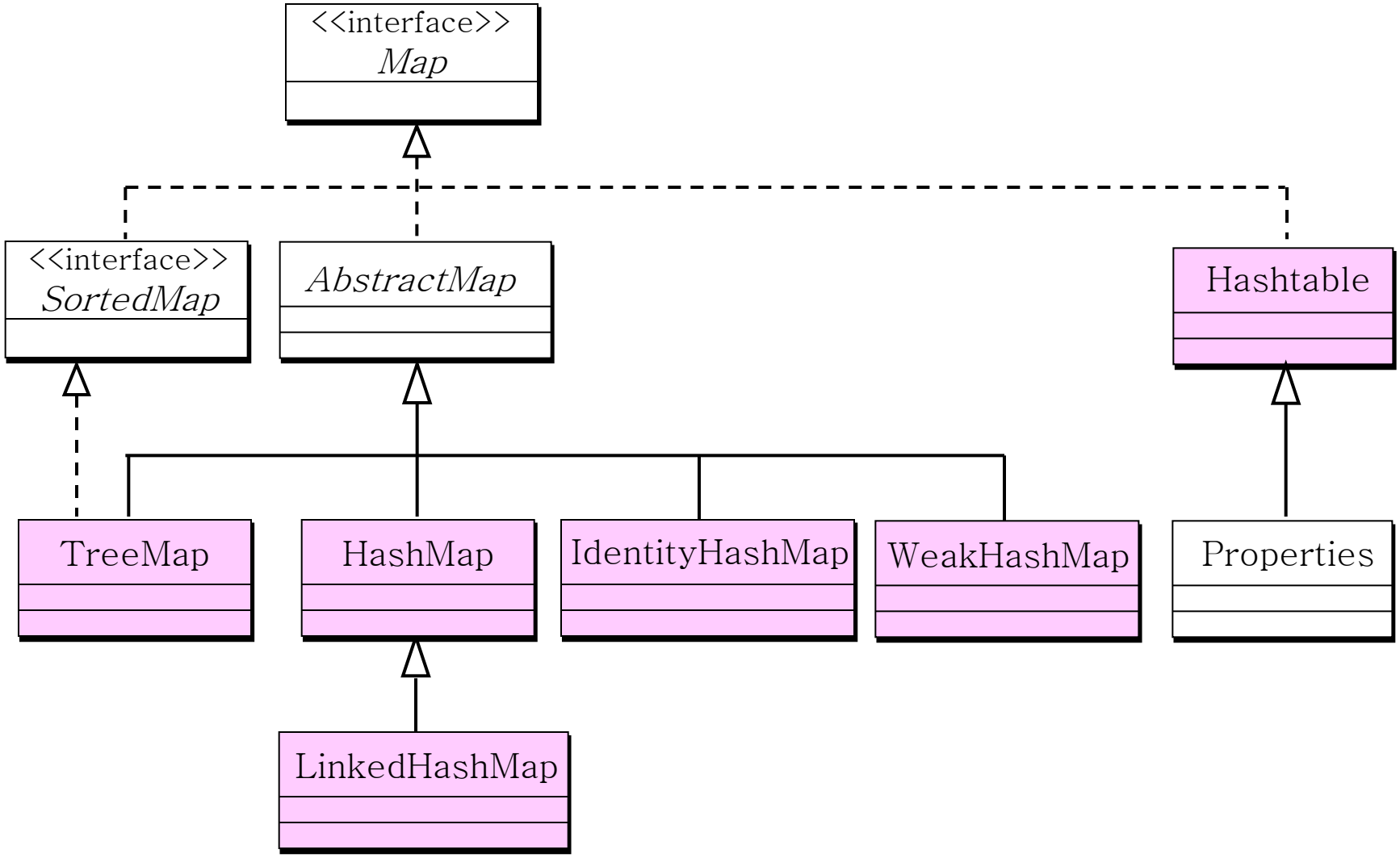


Collections – Vector

```
MyDate d11 = (MyDate)vec.get(0);
MyDate d12 = (MyDate)vec.get(1);
```



Collections – Map 계열



Collections – Map 계열

❑ HashMap

- Map에 키를 저장하는데 hash를 사용하여 성능이 좋다.
- 저장되는 순서가 유지 되지 않는다.
- 오직 하나의 Null 키를가질 수 있다.

❑ LinkedHashMap

- HashMap과 거의 같다. 차이점은 Map에 추가되는 순서를 유지한다는 점.
- JDK1.4에서 추가

❑ HashTable

- 동기화(Synchronization)를 제공한다.
- null키와 null 값을 저장할 수 없다.

Collections – Map 계열

```
import java.util.*;
public class MapExample {
    public static void main( String[] args ) {
        HashMap hMap = new HashMap();

        String key1 = "1";
        String key2 = "2";

        String value1 = "Hello Java";
        String value2 = "Java World";

        hMap.put( key1, value1 );
        hMap.put( key2, value2 );

        System.out.println( hMap.get( key1) );
        System.out.println( hMap.get( "2" ) );

        hMap.put( key1, value2 );
        System.out.println( hMap.get( key1) );
    }
}
```

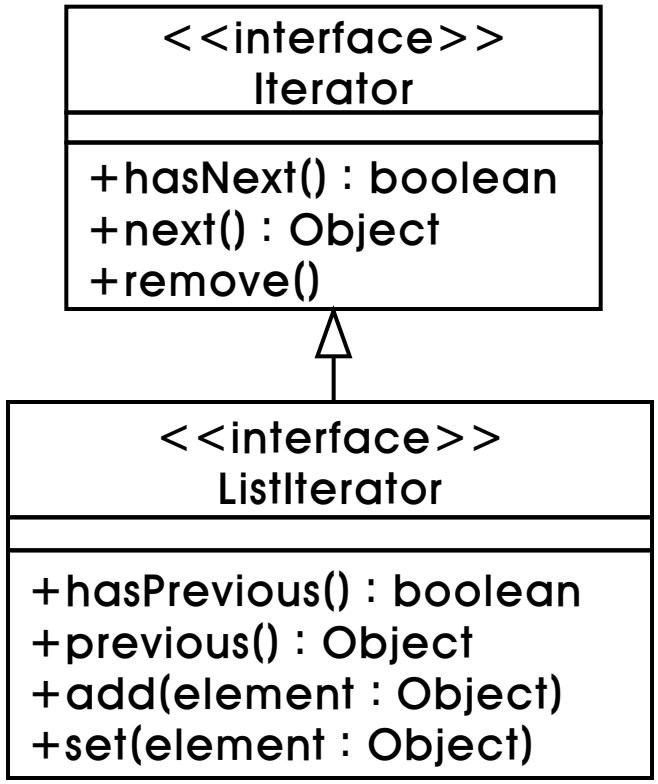
Iterator / ListIterator

❑ Iterator

- Collection 객체들의 내용을 순차적으로 접근 하는 기능 제공
- Set, List 계열에서 사용가능

❑ ListIterator

- Collection 객체들의 내용을 순차적 및 역방향 으로도 접근 하는 기능 제공
- List 계열에 사용 가능

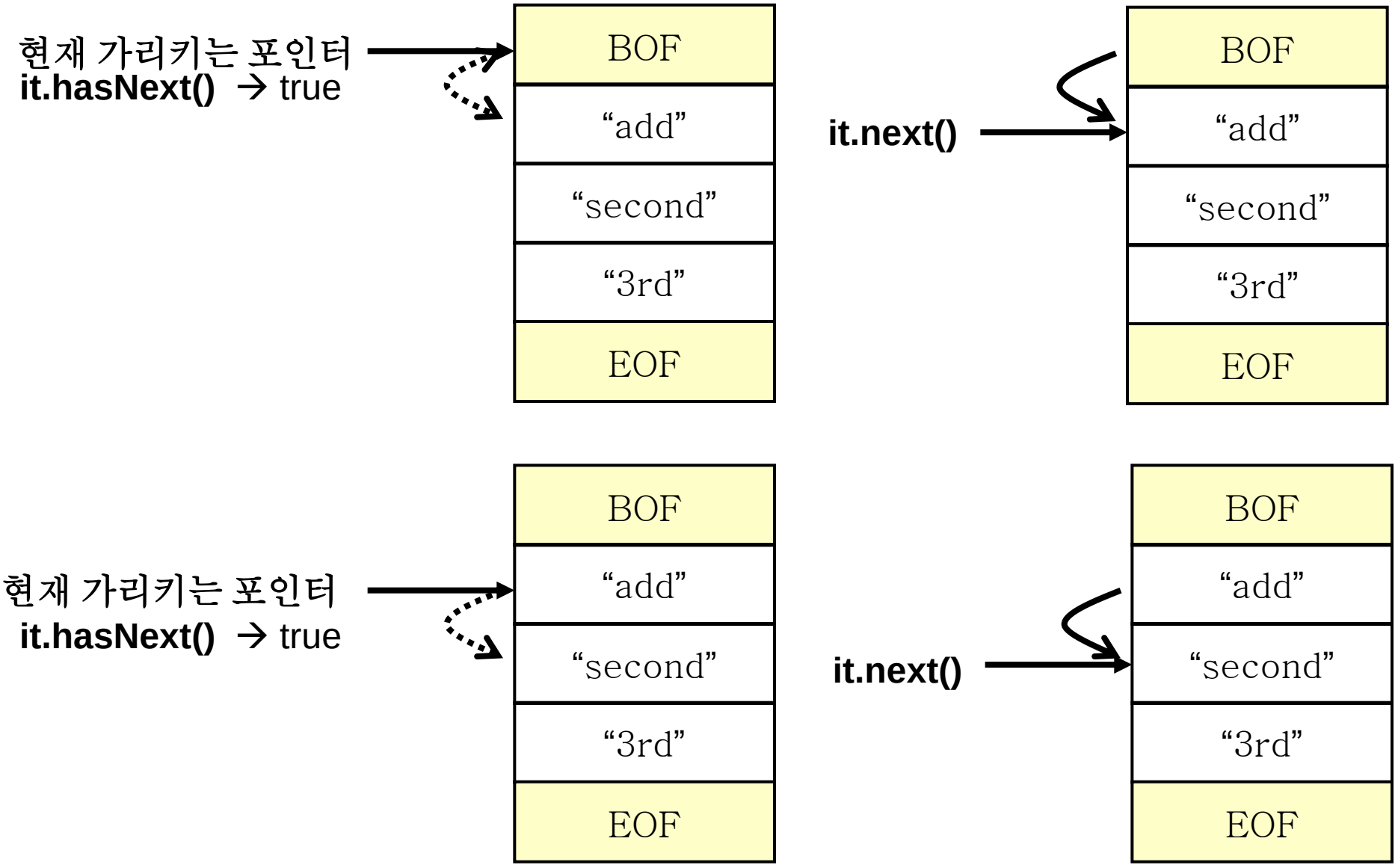


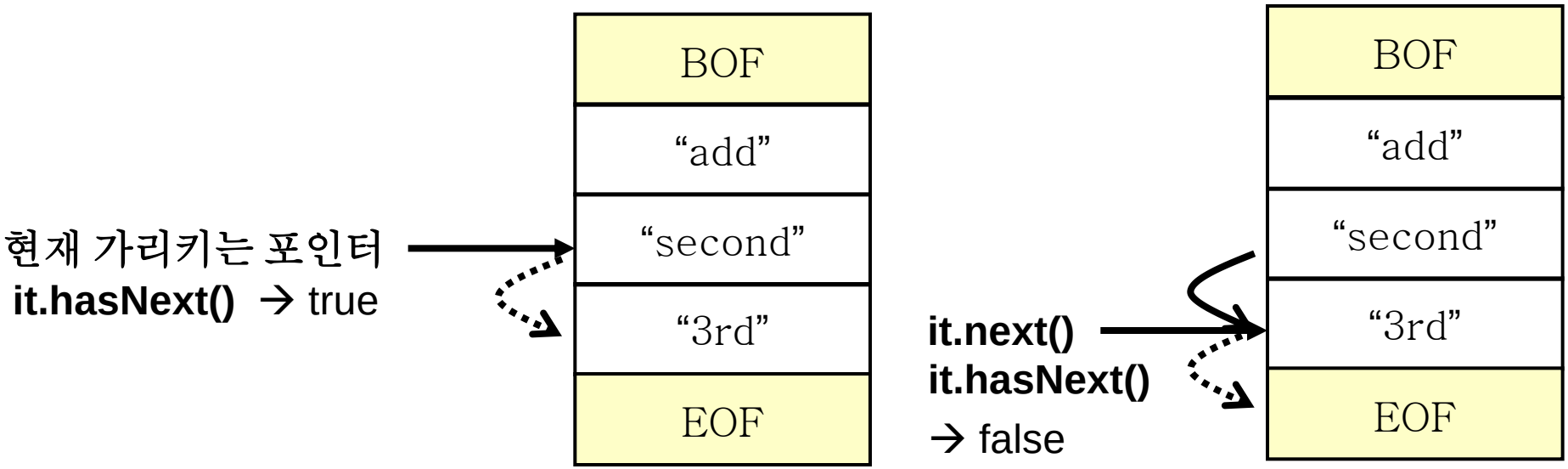
Iterator / ListIterator – Example


```
import java.util.*;

public class IteratorTest {
    public static void main( String[] args ) {
        List list = new ArrayList();
        list.add( "add" );
        list.add( "second" );
        list.add( "3rd" );

        Iterator it = list.iterator();

        System.out.println( "=>Iterator 사용 " );

        while ( it.hasNext() ) {
            System.out.println( it.next() );
        }

        ListIterator listIt = list.listIterator();

        System.out.println( "=>ListIterator 사용 " );

        if ( listIt.hasNext() ) {
            System.out.println( listIt.next() );
        }

        if ( listIt.hasPrevious() ) {
            System.out.println( listIt.previous() );
        }
    }
}
```

Iterator Interface

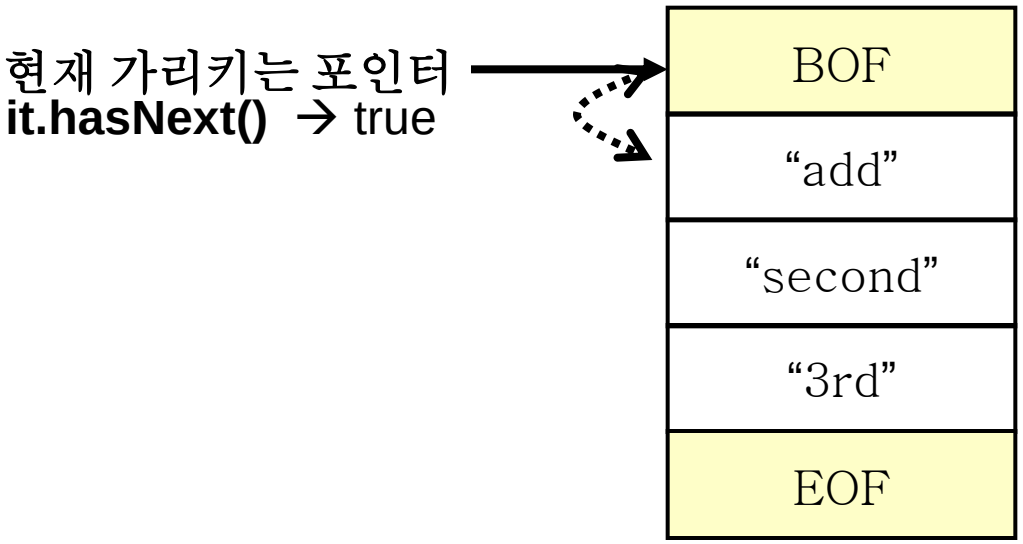
Example




현재 가리키는 포인터
it.hasNext() → true
BOF
“add”
“second”
“3rd”
EOF
it.next()
it.hasNext()
→ false
BOF
“add”
“second”
“3rd”
EOF

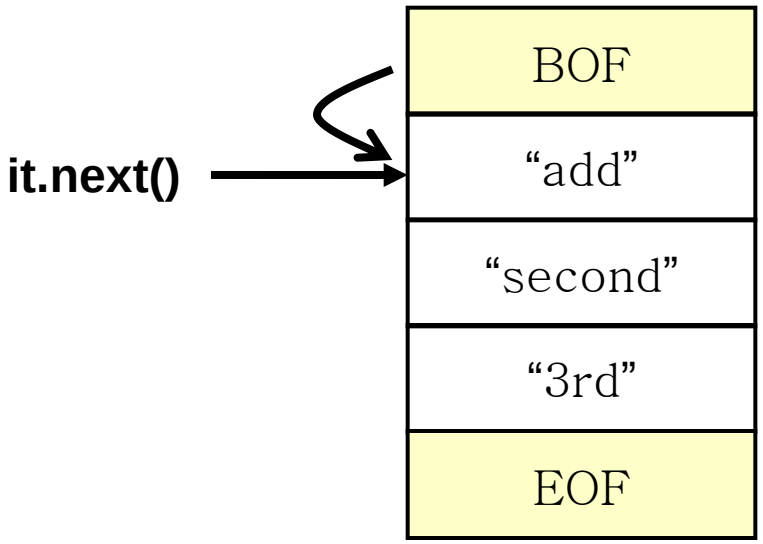
Iterator Interface

Example



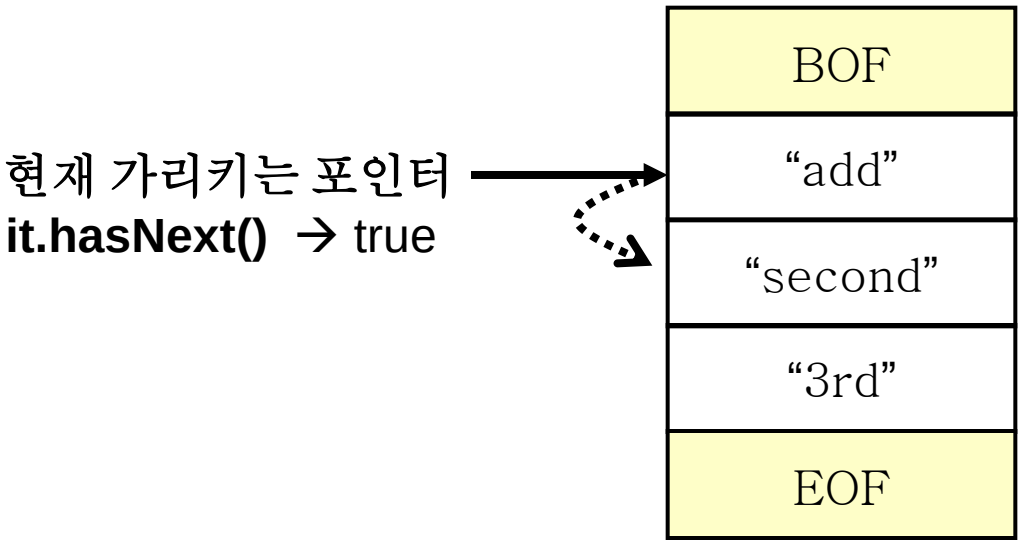
Iterator Interface

Example



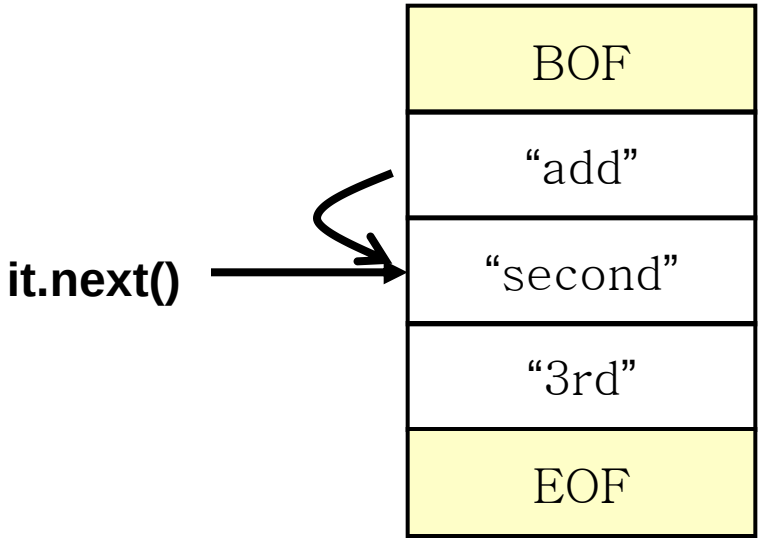
Iterator Interface

Example



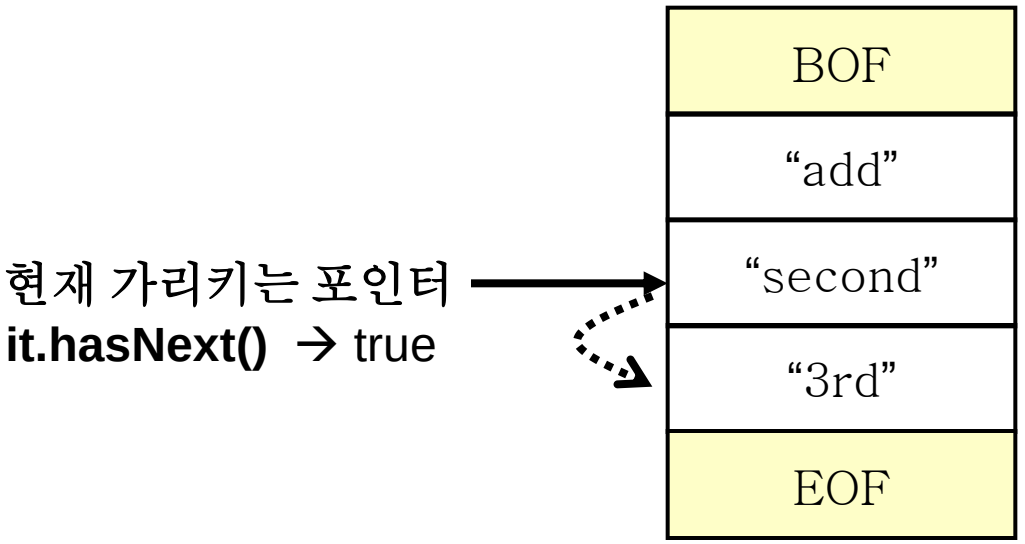
Iterator Interface

Example



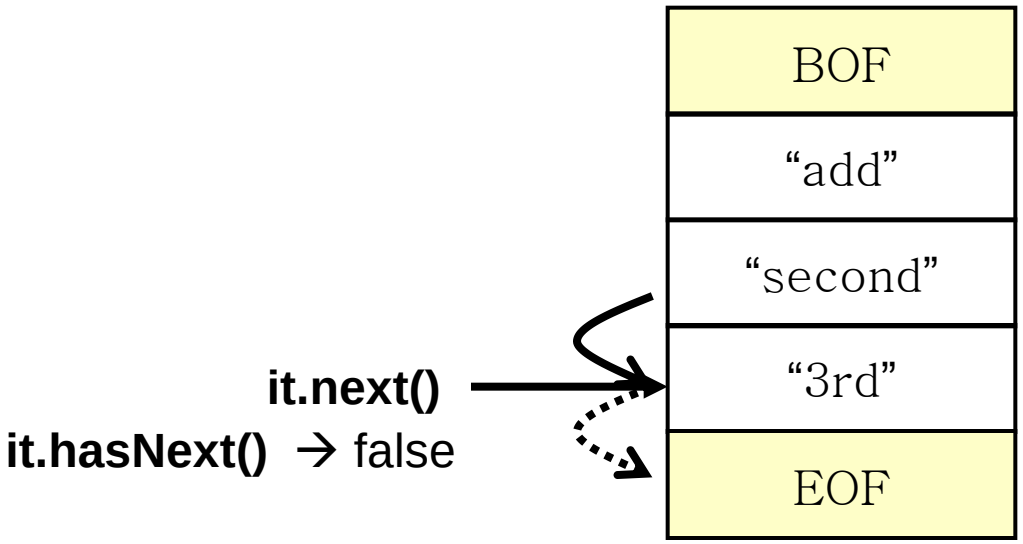
Iterator Interface

Example



Iterator Interface

Example



Enumeration Interface

- Iterator Interface와 마찬가지로, 자바에서 제공하는 Collection 객체들에 대해, 각 Collection의 항목들을 순차적으로 접근하는 방법을 제공
- 자바 초창기 버전에서, 제공 되었던 것으로 Vector, HashTable 지원
- java.util Package에 있음
- 제공 메소드

`public boolean hasMoreElements()`
• 다음 항목이 있는지 검사. 있다면 true, 없으면 false return
`public Object nextElement()`
• Collection 객체에서 다음 항목을 Object 타입으로 return

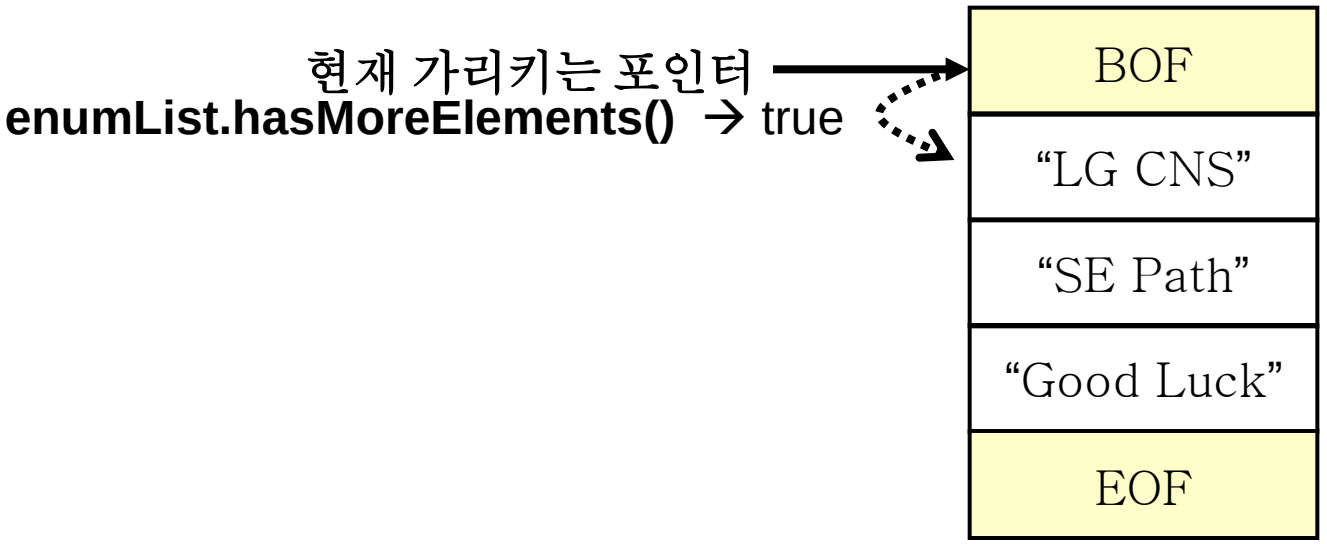
Enumeration Interface

Example

```
 1 import java.util.*;
 2
 3 public class EnumerationTest {
 4
 5     public static void main( String[] args ) {
 6         Vector vec = new Vector();
 7
 8         vec.add( "LG CNS" );
 9         vec.add( "SE Path" );
10         vec.add( “Good Luck!!!" );
11
12         Enumeration enumList = vec.elements();
13
14         String str = null;
15
16         while ( enumList.hasMoreElements() ) {
17             str = (String)enumList.nextElement();
18             System.out.println( str );
19         }
20     }
21 }
```

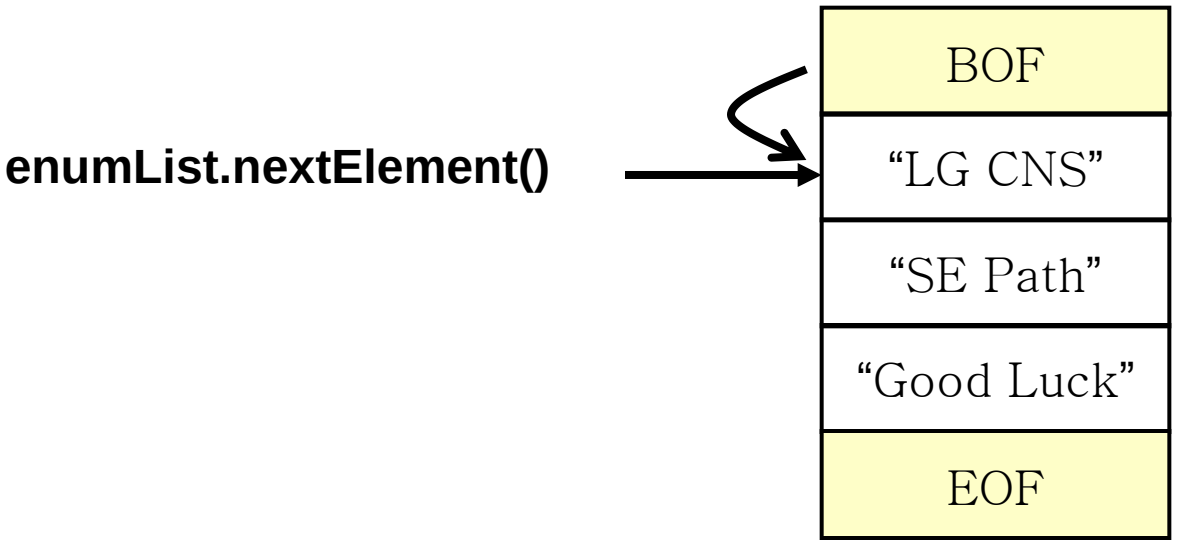
Enumeration Interface

Example



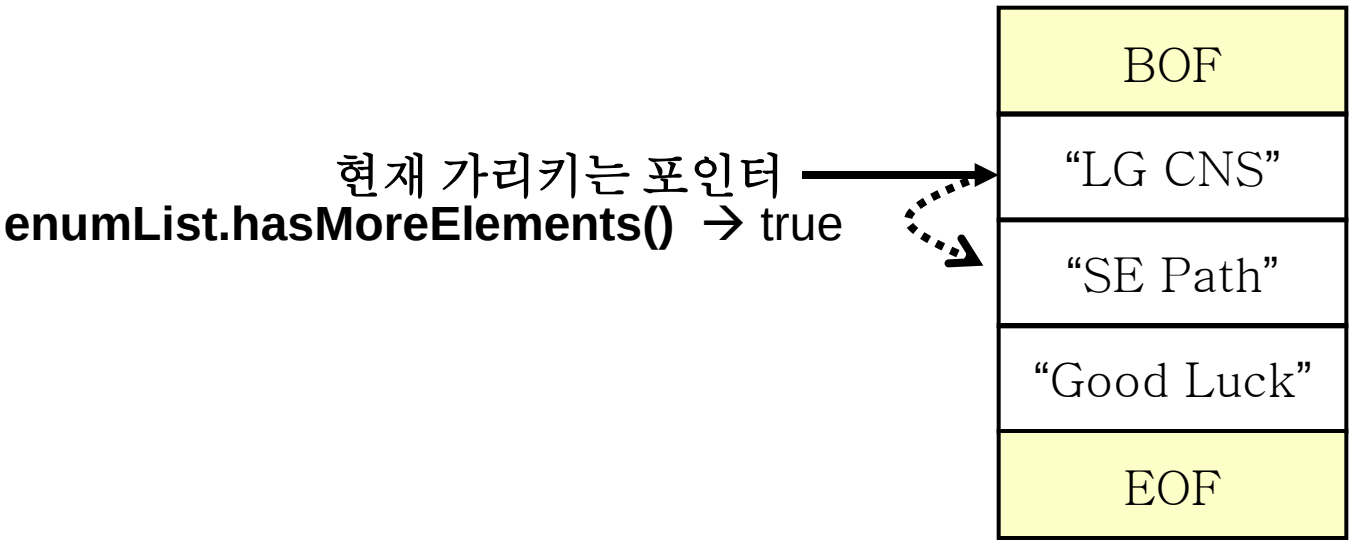
Enumeration Interface

Example



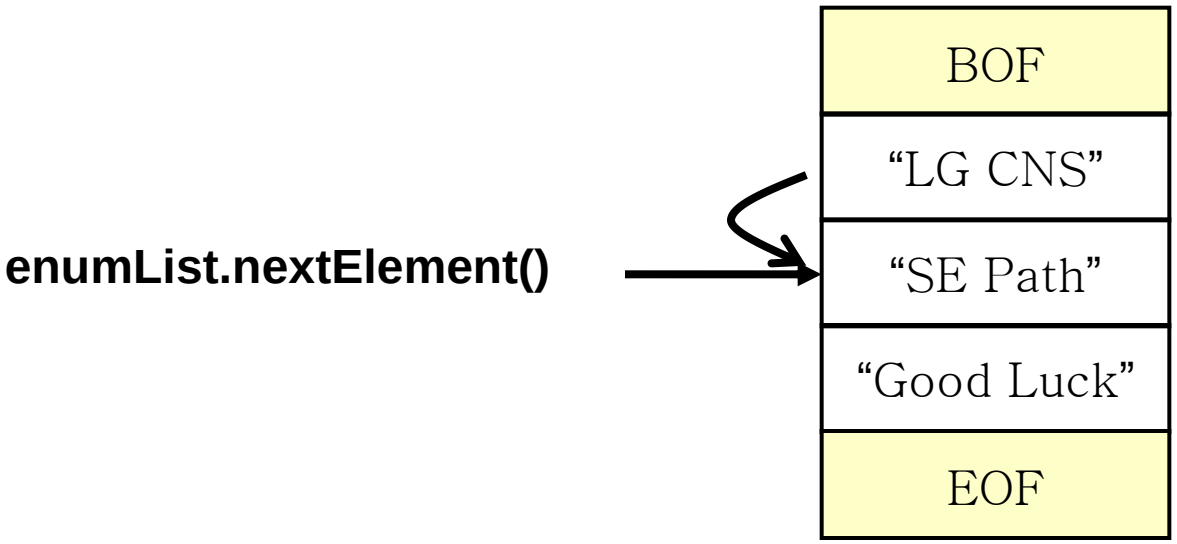
Enumeration Interface

Example



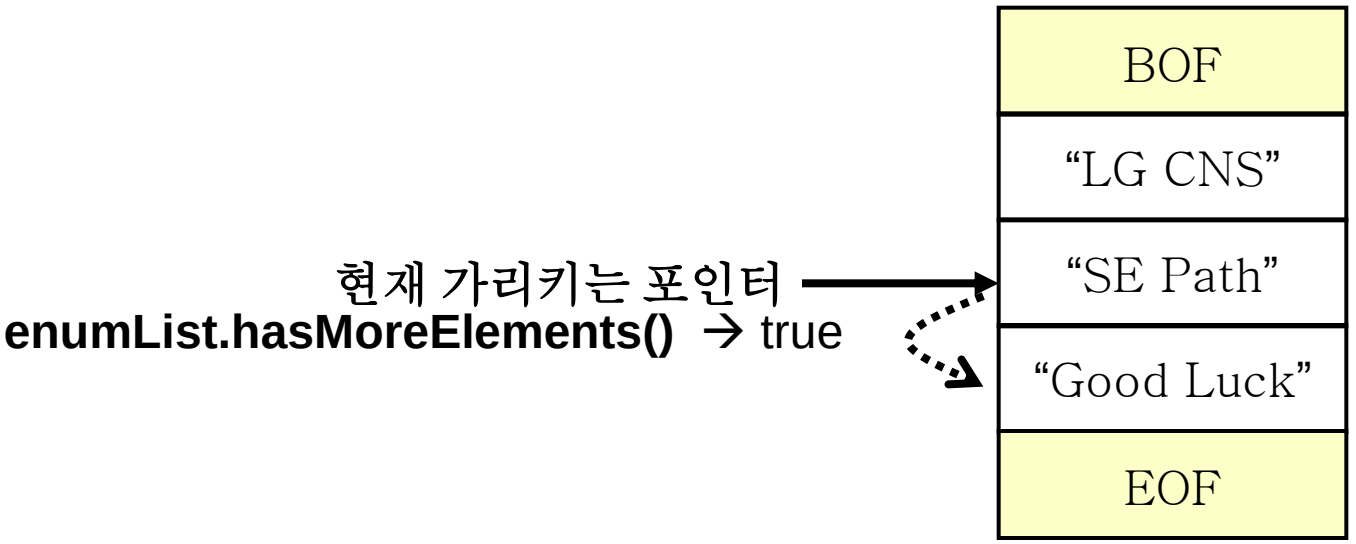
Enumeration Interface

Example



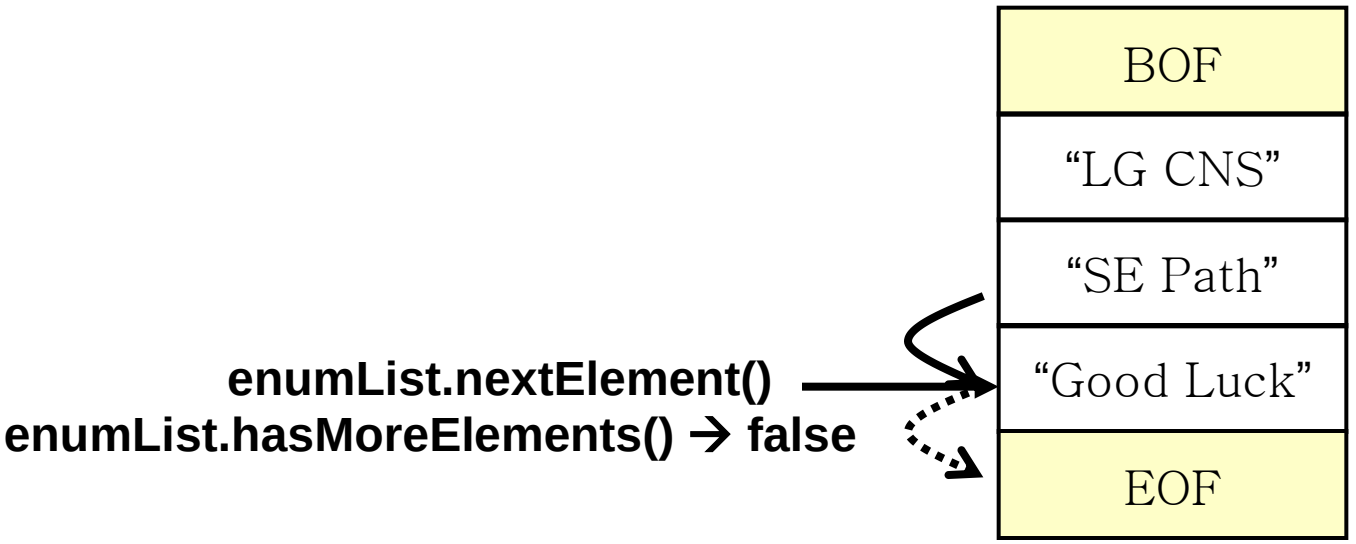
Enumeration Interface

Example



Enumeration Interface

Example



equals( ) Overriding & hashCodo( )

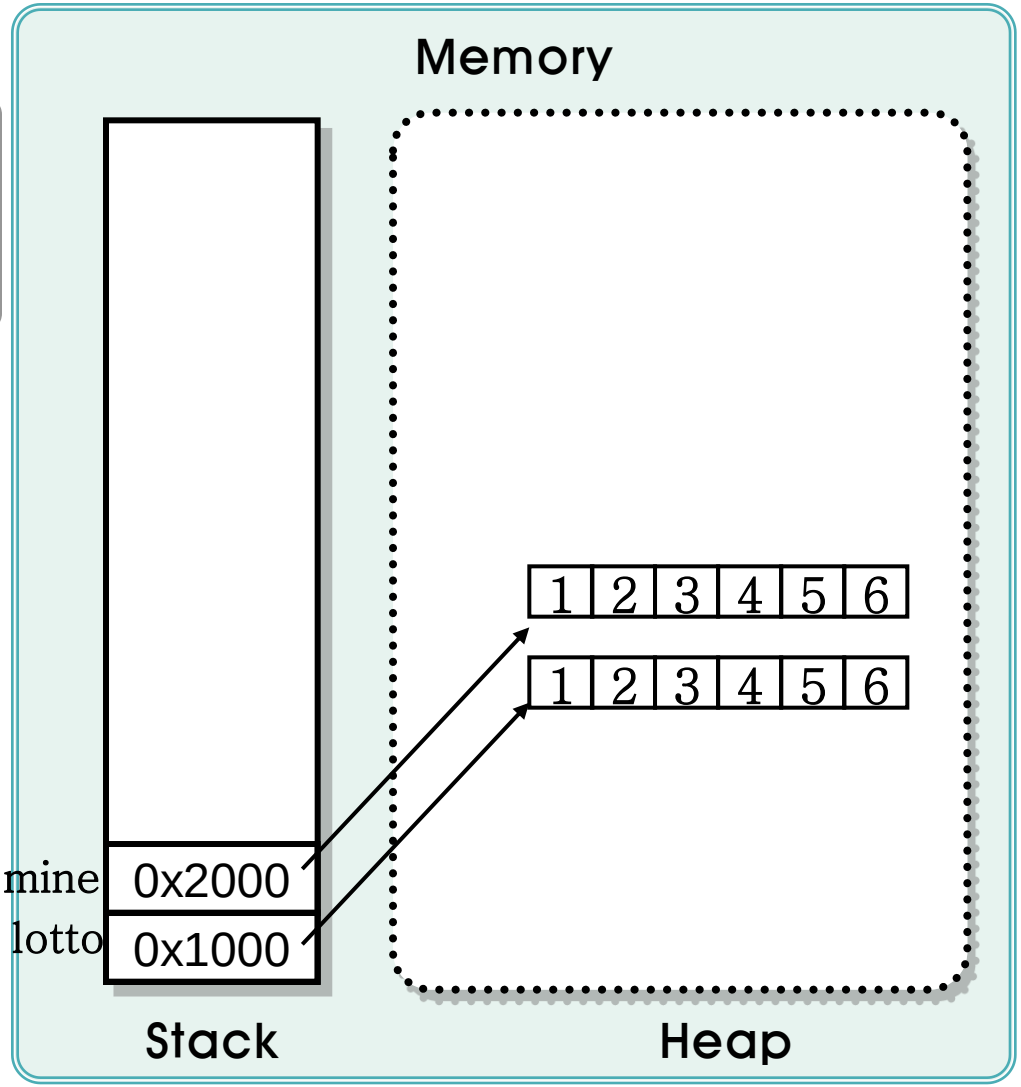
```
public class Lotto {
    private int numbers[] = new int[6];

    public Lotto( int a, int b, int c,
                  int d, int e, int f)  {
        numbers[0] = a;
        numbers[1] = b;
        numbers[2] = c;
        numbers[3] = d;
        numbers[4] = e;
        numbers[5] = f;
    }

    public int[] getNumbers()  {
        return numbers;
    }
}
```

```
public class TestLotto {
    public static void main( String[] args ) {
        Lotto lotto = new Lotto( 1, 2, 3, 4, 5, 6 );
        Lotto mine = new Lotto( 1, 2, 3, 4, 5, 6 );

        if ( lotto.equals( mine ) )  {
            System.out.println( "인생역전!!!" );
        } else {
            System.out.println( "다음 주에,,,ㅠㅠ" );
        }
}
```

equals( ) 를 Overriding 안 한 경우

Object 클래스

```
public boolean equals(Object obj) {
        return (this == obj);
}
```



equals( ) 를 Overriding

Lotto 클래스

```
public boolean equals( Object obj ) {

    boolean isMatch = true;

    if ( this == obj )
        return true;

    if ( obj instanceof Lotto ) {

        Lotto lot = (Lotto)obj;

        for ( int inx = 0 ; isMatch && inx < numbers.length ; inx++ ) {

            if ( numbers[inx] != lot.getNumbers()[inx] )  {
                isMatch = false;
            }
        }
    } else {
        isMatch = false;
    }

    return isMatch;
}
```

객체를 hash를 이용하는 Collection에 저장?

```
import java.util.*;

public class TestLotto {
    public static void main( String[] args ) {
        Lotto lotto = new Lotto( 1, 2, 3, 4, 5, 6 );
        Lotto mine = new Lotto( 1, 2, 3, 4, 5, 6 );

        HashMap hMap = new HashMap();

        hMap.put( lotto, “1등!!!" );

        System.out.println( (String)hMap.get(lotto) );
        System.out.println( (String)hMap.get(mine) );
    }
}
```

객체를 hash를 이용하는 Collection에 저장?

```
public class Lotto {
    private int numbers[] = new int[6];

    public Lotto( int a, int b, int c,
                  int d, int e, int f ) {

        numbers[0] = a;
        numbers[1] = b;
        numbers[2] = c;
        numbers[3] = d;
        numbers[4] = e;
        numbers[5] = f;
    }

    public int[] getNumbers() {

        return numbers;
    }


    public int hashCode() {

        return numbers[0] ^ numbers[1] ^
               numbers[2] ^ numbers[3] ^
               numbers[4] ^ numbers[5];
    }

    public boolean equals( Object obj ) {

        boolean isMatch = true;

        if ( this == obj )
            return true;

        if ( obj instanceof Lotto ) {
            Lotto lot = (Lotto)obj;

            for ( int inx = 0 ;
                  isMatch && inx < numbers.length ;
                  inx++   ) {
                if ( numbers[inx] != lot.getNumbers()[inx] ) {
                    isMatch = false;
                }
            }
        }

        return isMatch;
    }

}
```