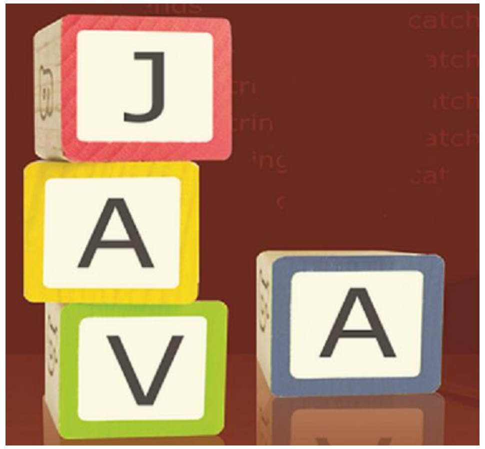


Java Data Structure

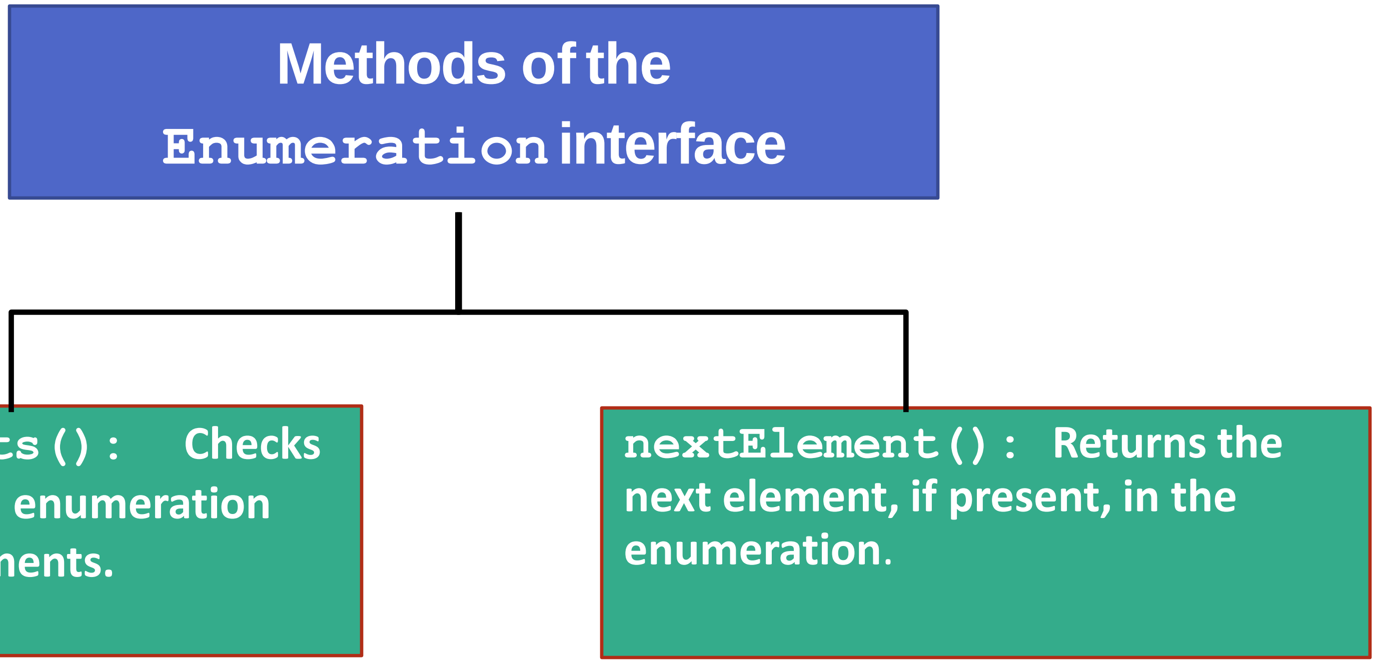
SESSION 16

Objectives

- Explain the `Enumeration` interface
- Describe the `BitSet` class
- Describe the `Stack` classes
- Explain the `Dictionary` classes

Enumeration [1/3]

- Enumeration is an interface in the `java.util` package that defines methods to iterate through the elements of a collection



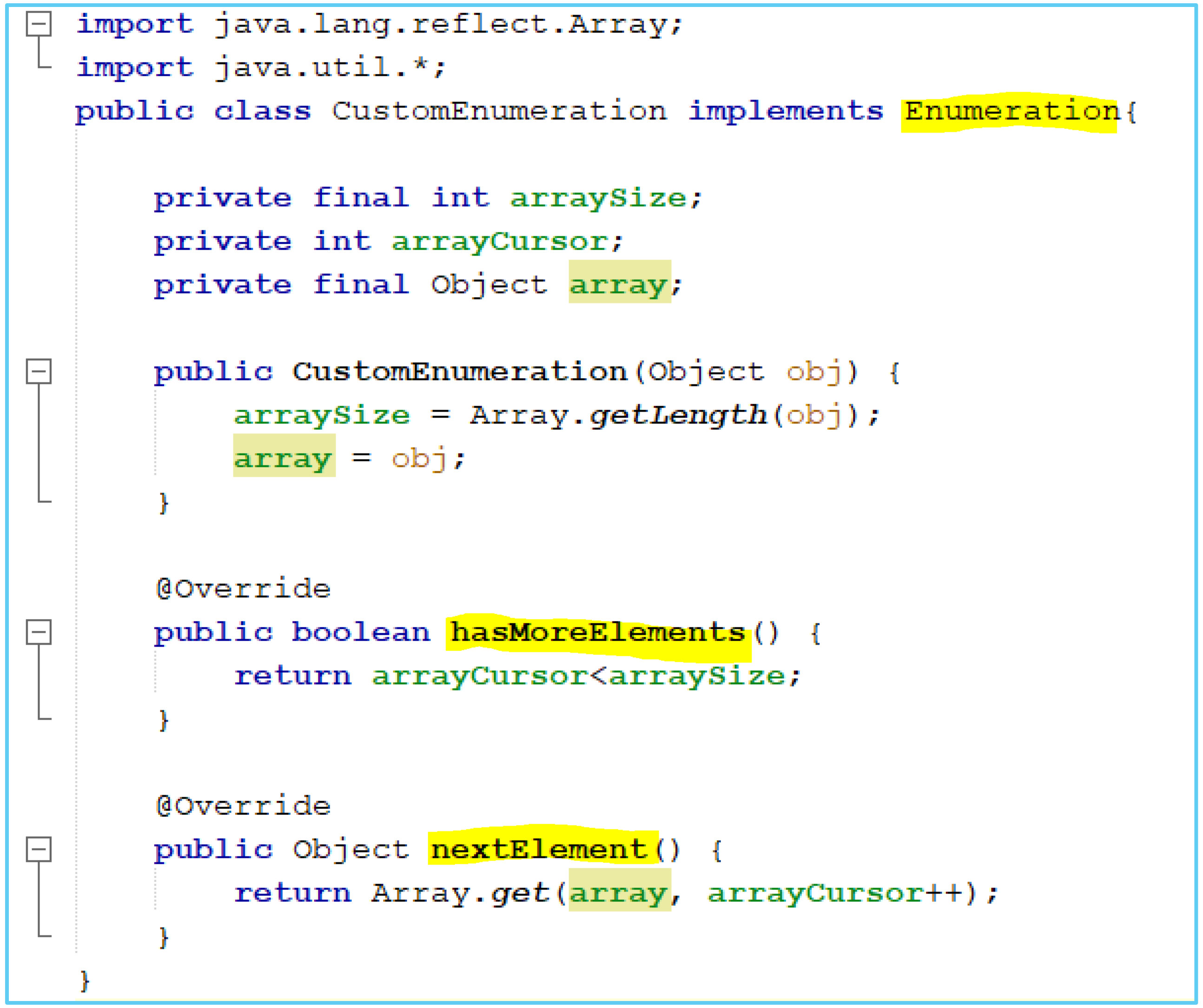
Methods of the `Enumeration` interface

`hasMoreElements()`: Checks whether or not the enumeration contains more elements.

`nextElement()`: Returns the next element, if present, in the enumeration.

Enumeration [2/3]

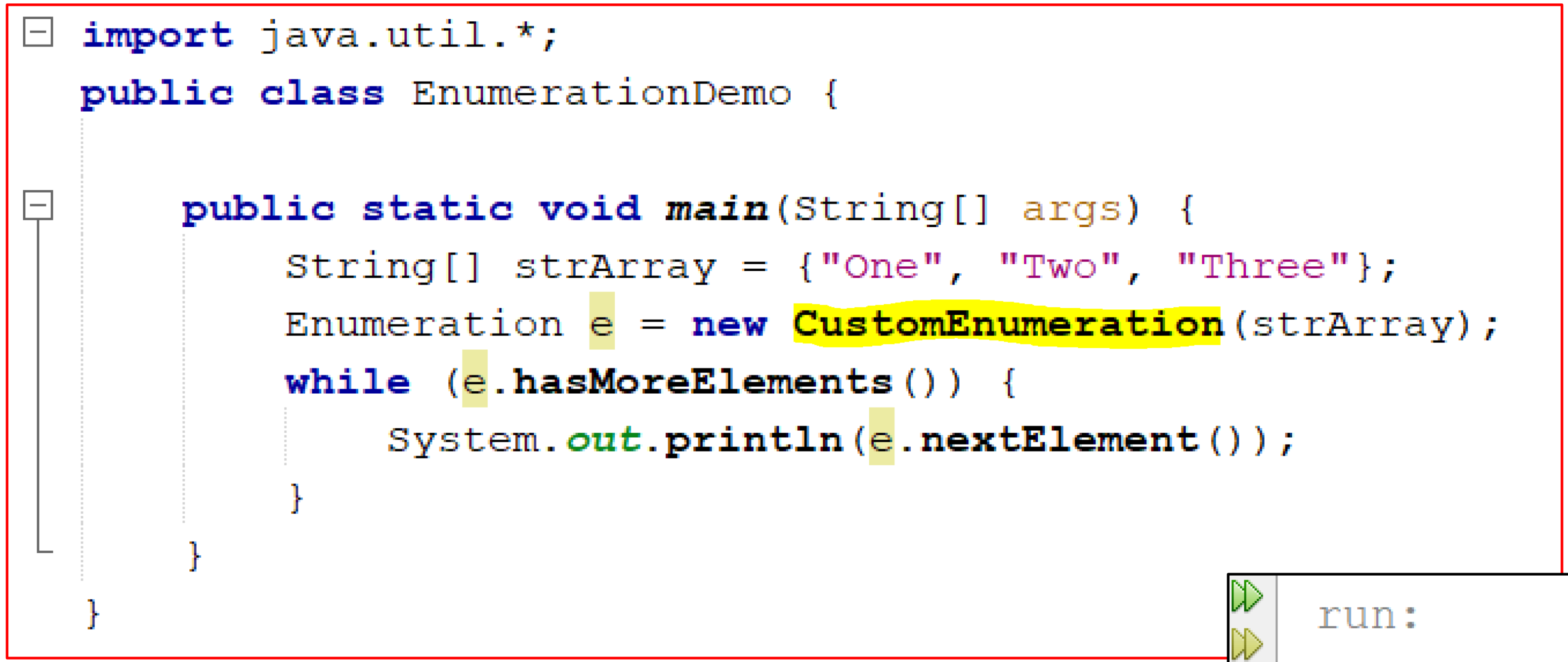
- The code for using an **`Enumeration`** to iterate through the elements of an array



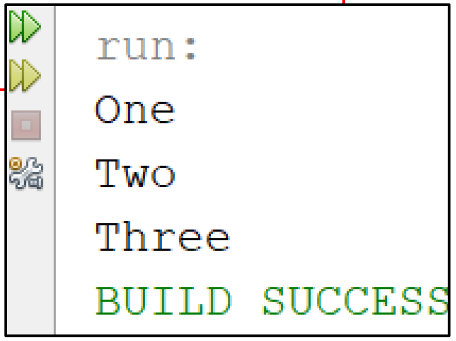
```java
import java.lang.reflect.Array;
import java.util.*;
public class CustomEnumeration implements Enumeration{

    private final int arraySize;
    private int arrayCursor;
    private final Object array;

    public CustomEnumeration(Object obj) {
        arraySize = Array.getLength(obj);
        array = obj;
    }

    @Override
    public boolean hasMoreElements() {
        return arrayCursor<arraySize;
    }

    @Override
    public Object nextElement() {
        return Array.get(array, arrayCursor++);
    }
}
```

Enumeration [3/3]

- The code for using the custom enumeration defined in earlier code.



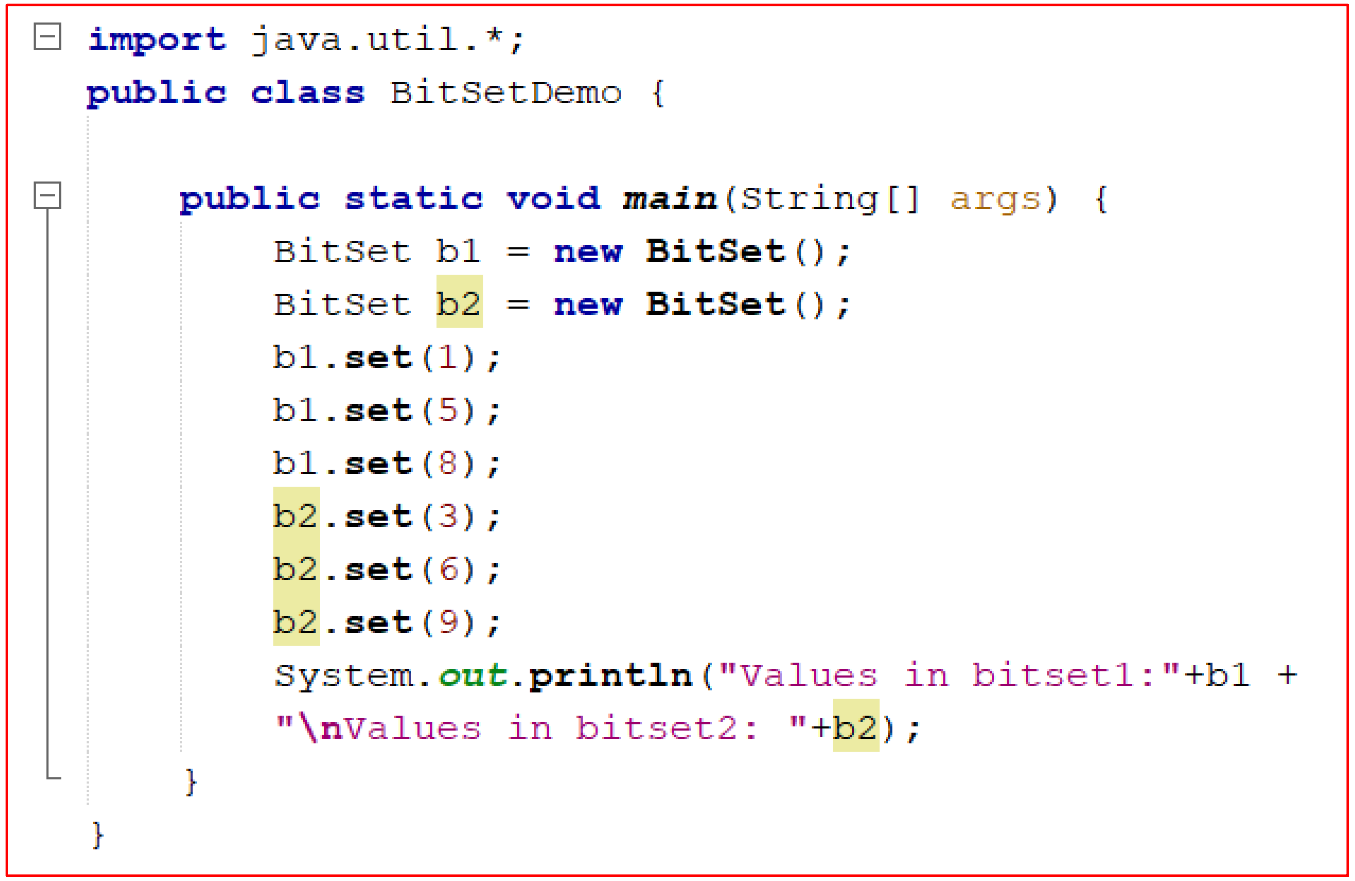
```java
import java.util.*;
public class EnumerationDemo {

    public static void main(String[] args) {
        String[] strArray = {"One", "Two", "Three"};
        Enumeration e = new CustomEnumeration(strArray);
        while (e.hasMoreElements()) {
            System.out.println(e.nextElement());
        }
    }
}
```



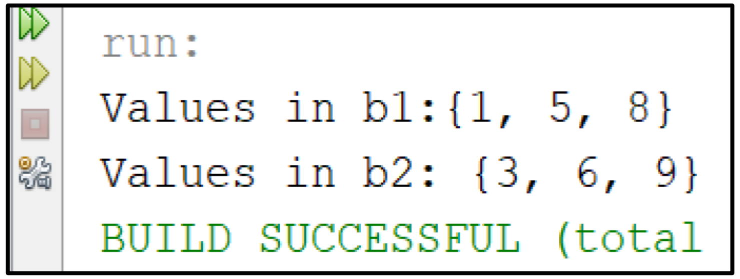
```
run:
One
Two
Three
BUILD SUCCESS
```

Bitset

- The code for using the **`BitSet`** class.



```
import java.util.*;
public class BitSetDemo {

    public static void main(String[] args) {
        BitSet b1 = new BitSet();
        BitSet b2 = new BitSet();
        b1.set(1);
        b1.set(5);
        b1.set(8);
        b2.set(3);
        b2.set(6);
        b2.set(9);
        System.out.println("Values in bitset1:"+b1 +
        "\nValues in bitset2: "+b2);
    }
}
```



```
run:
Values in b1:{1, 5, 8}
Values in b2: {3, 6, 9}
BUILD SUCCESSFUL (total
```

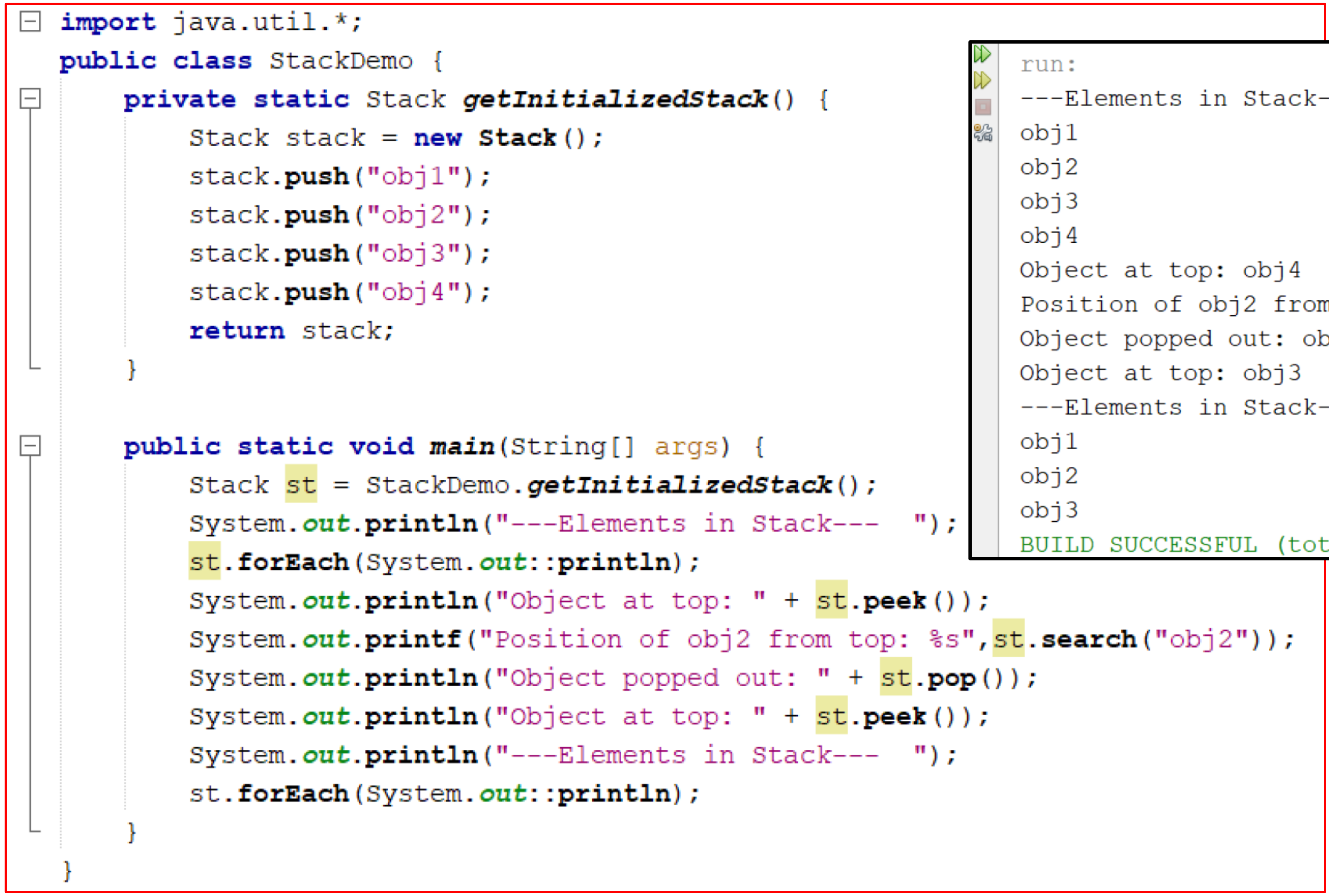
Stack [1/2]

- Following table lists the **`Stack`** methods:

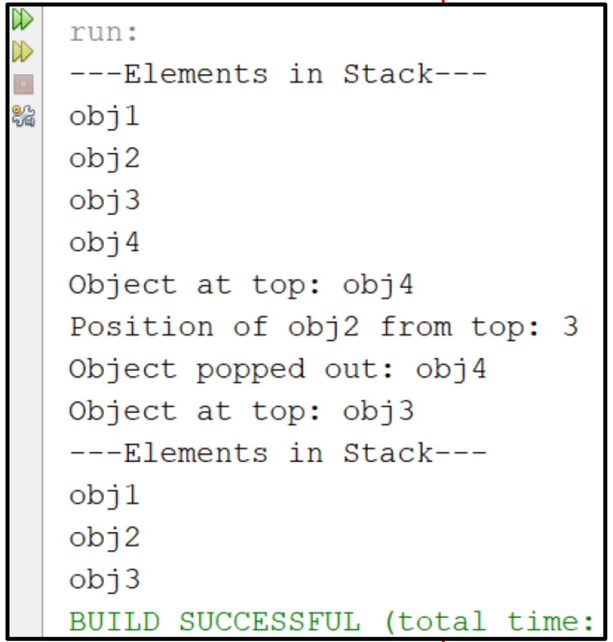
Abstract Method	Description
`empty()`	Checks whether or not the Stack is empty.
`peek()`	Returns the object at the top of the Stack without removing the object.
`pop()`	Returns the object at the top of the Stack after removing the object from the Stack.
`push(E item)`	Pushes an object onto the top of this Stack.
`search(Object o)`	Returns the position of an object from the top of the Stack. This method returns 1 for the object at the top of the Stack, 2 for the object below it, and so on. If an object is not found, this method returns -1.

Stack [2/2]

- The code demonstrates the use of the **`Stack`** class



```java
import java.util.*;
public class StackDemo {
    private static Stack getInitializedStack() {
        Stack stack = new Stack();
        stack.push("obj1");
        stack.push("obj2");
        stack.push("obj3");
        stack.push("obj4");
        return stack;
    }

    public static void main(String[] args) {
        Stack st = StackDemo.getInitializedStack();
        System.out.println("---Elements in Stack---   ");
        st.forEach(System.out::println);
        System.out.println("Object at top: " + st.peek());
        System.out.printf("Position of obj2 from top: %s",st.search("obj2"));
        System.out.println("Object popped out: " + st.pop());
        System.out.println("Object at top: " + st.peek());
        System.out.println("---Elements in Stack---   ");
        st.forEach(System.out::println);
    }
}
```



```
run:
---Elements in Stack---
obj1
obj2
obj3
obj4
Object at top: obj4
Position of obj2 from top: 3
Object popped out: obj4
Object at top: obj3
---Elements in Stack---
obj1
obj2
obj3
BUILD SUCCESSFUL (total time:
```

Dictionary Classes

Dictionary

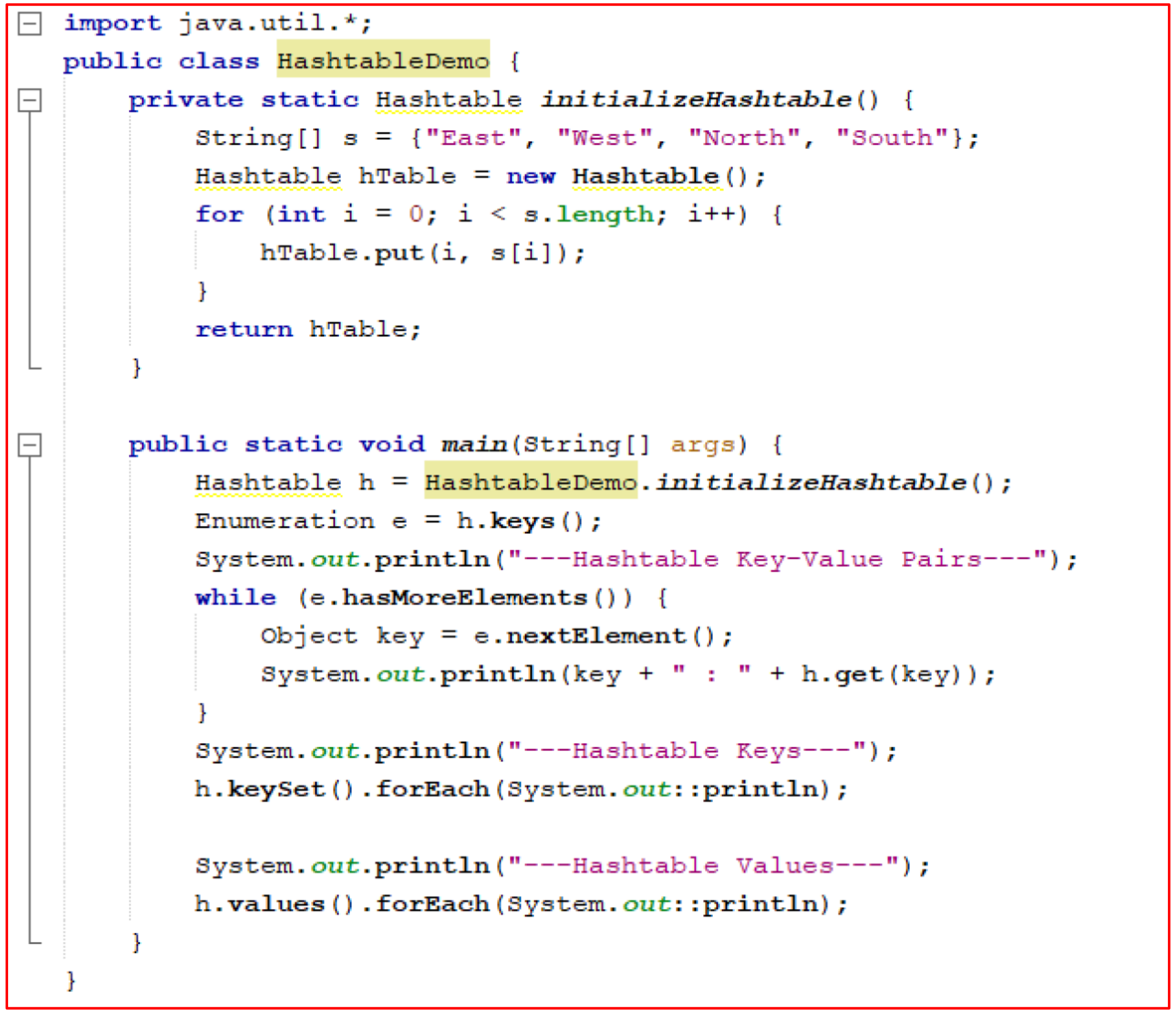
- Used to store key-value pairs.
- Every key and value in a dictionary is an object.
- The `Dictionary` abstract class of the `java.util` package is the super class of all dictionary implementation classes.

Hashtable Class [1/2]

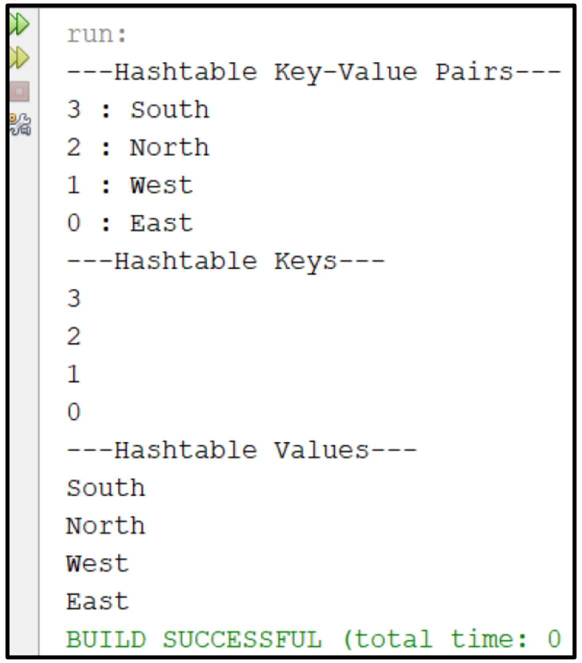
- The **`Hashtable`** class
  - implements a collection of key-value pairs that are organized based on the hash code of the key.
  - is significantly faster as compared to other dictionaries.
  - When elements are added to a Hashtable, the Hashtable automatically resizes itself by increasing its capacity.

- A hash code is a signed number that identifies the key. Based on the hash code, a key-value pair, when added to a Hashtable, gets stored into a particular bucket.

Hashtable Class [2/2]



```java
import java.util.*;
public class HashtableDemo {
    private static Hashtable initializeHashtable() {
        String[] s = {"East", "West", "North", "South"};
        Hashtable hTable = new Hashtable();
        for (int i = 0; i < s.length; i++) {
            hTable.put(i, s[i]);
        }
        return hTable;
    }

    public static void main(String[] args) {
        Hashtable h = HashtableDemo.initializeHashtable();
        Enumeration e = h.keys();
        System.out.println("---Hashtable Key-Value Pairs---");
        while (e.hasMoreElements()) {
            Object key = e.nextElement();
            System.out.println(key + " : " + h.get(key));
        }
        System.out.println("---Hashtable Keys---");
        h.keySet().forEach(System.out::println);

        System.out.println("---Hashtable Values---");
        h.values().forEach(System.out::println);
    }
}
```



```
run:
---Hashtable Key-Value Pairs---
3 : South
2 : North
1 : West
0 : East
---Hashtable Keys---
3
2
1
0
---Hashtable Values---
South
North
West
East
BUILD SUCCESSFUL (total time: 0
```

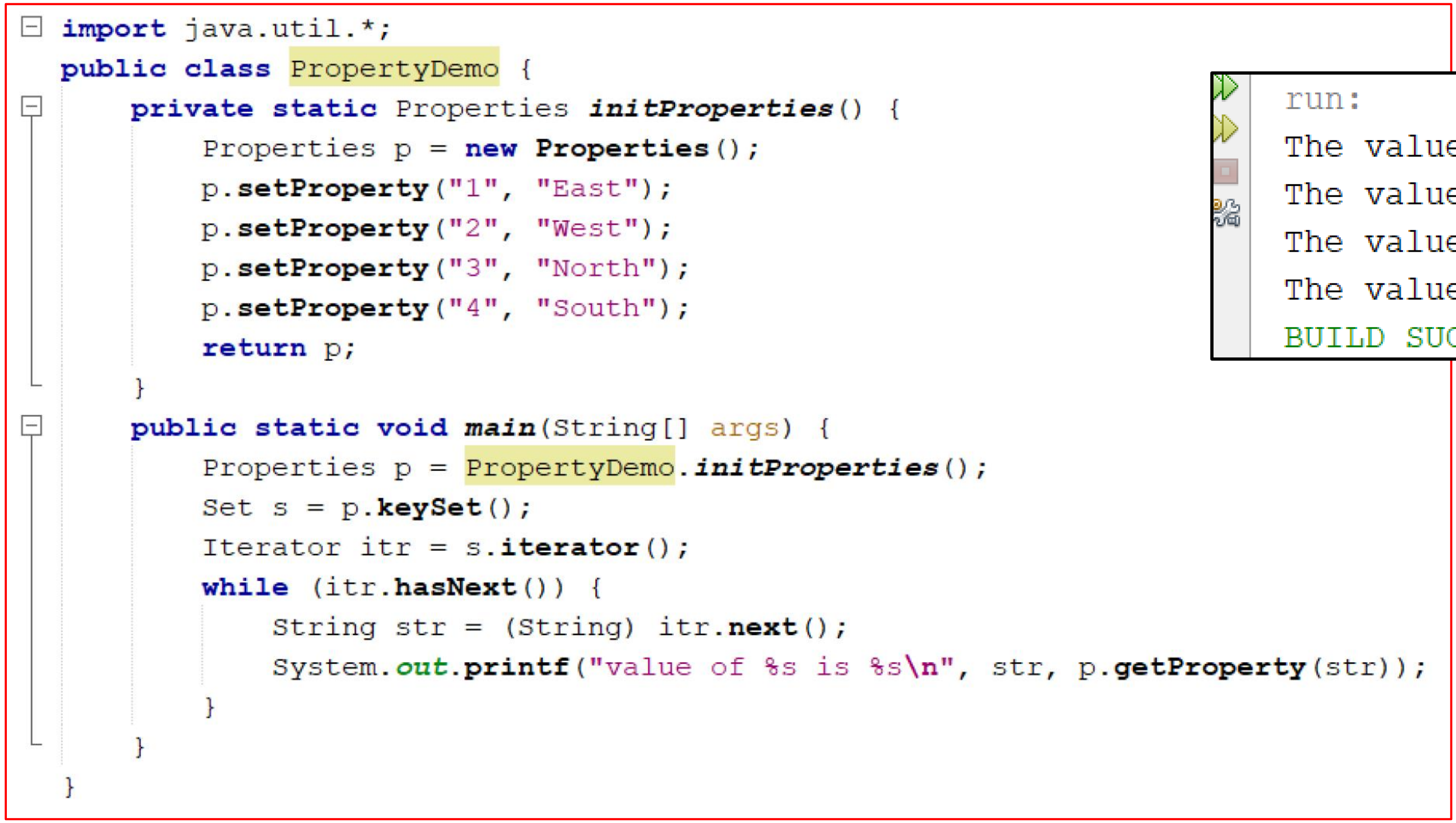
Properties Class [1/2]

Properties class

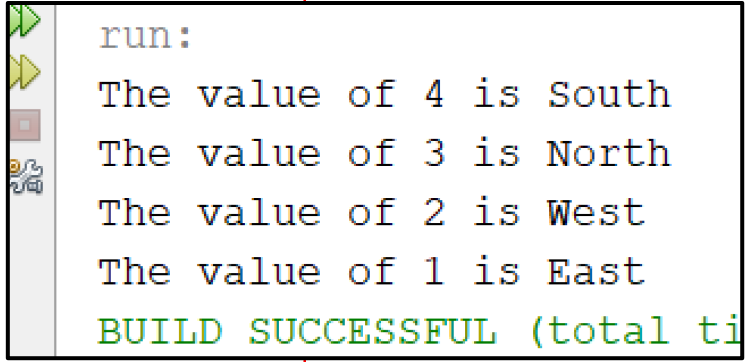
extends **`Hashtable`** to implement a collection of key-value pairs.

inherits the **`put()`** method to add a key-value pair, you should avoid it.

Properties Class [2/2]



```
import java.util.*;
public class PropertyDemo {
    private static Properties initProperties() {
        Properties p = new Properties();
        p.setProperty("1", "East");
        p.setProperty("2", "West");
        p.setProperty("3", "North");
        p.setProperty("4", "South");
        return p;
    }
    public static void main(String[] args) {
        Properties p = PropertyDemo.initProperties();
        Set s = p.keySet();
        Iterator itr = s.iterator();
        while (itr.hasNext()) {
            String str = (String) itr.next();
            System.out.printf("value of %s is %s\n", str, p.getProperty(str));
        }
    }
}
```



```
run:
The value of 4 is South
The value of 3 is North
The value of 2 is West
The value of 1 is East
BUILD SUCCESSFUL (total ti
```

Summary

- Java includes a few legacy data structures such as Enumeration, BitSet, and so on for backward compatibility.
- Enumeration interface is used to iterate through the elements of a collection.
- BitSet is a collection of bit values.
- Stack extends Vector to provide an implementation of a LIFO collection.
- Dictionary is used to store key-value pairs.
- Hashtable stores key-value pairs where keys are organized based on their hash code.
- Properties stores key-value pairs where both the types of the keys and values are String.