Power Programming in Java

Session: 12

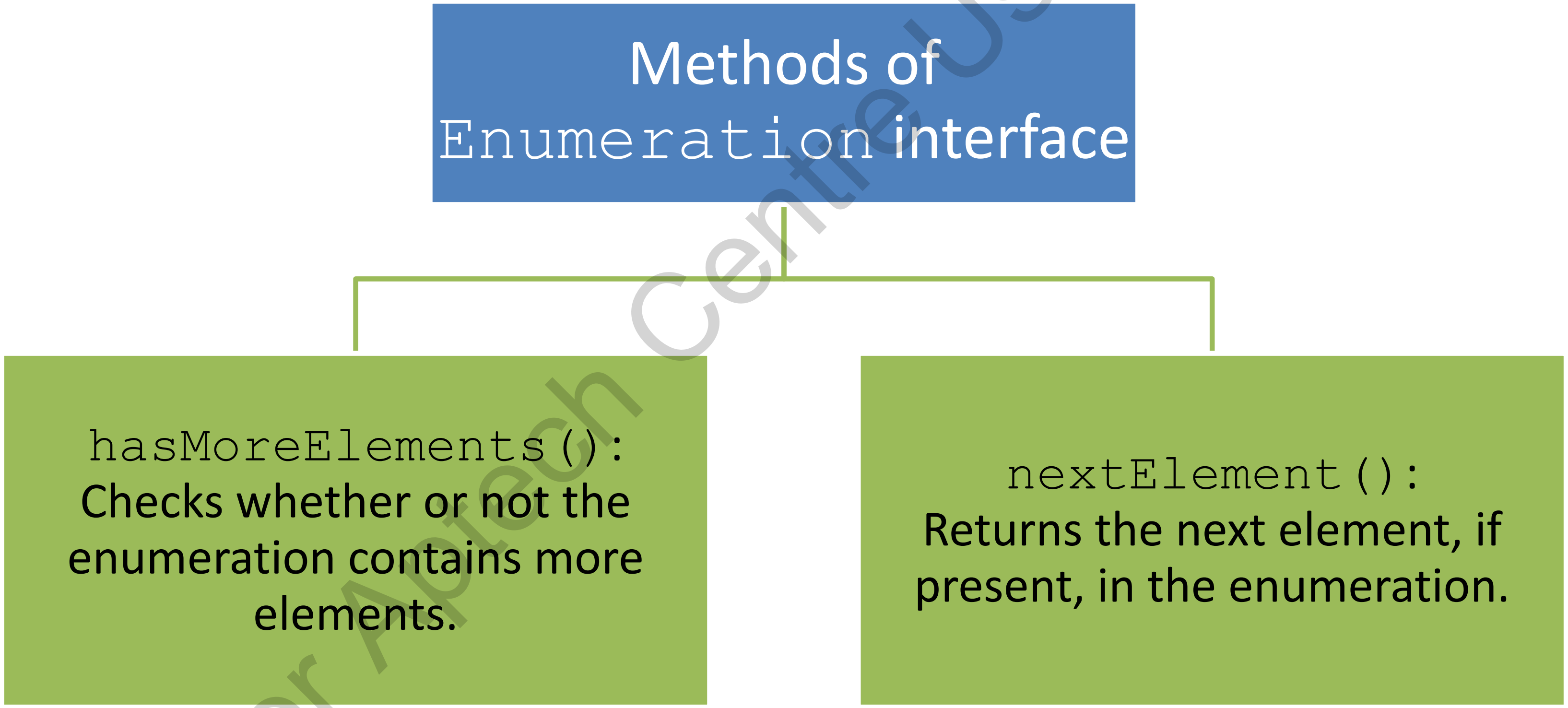
Java Data Structures

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 `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.

Code Snippet

```
package com.datastructures.demo;
import java.lang.reflect.Array;
import java.util.Enumeration;

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]

- Following code utilizes custom enumeration defined earlier.

Code Snippet

```
package com.datastructures.demo;
import java.util.Enumeration;
public class EnumerationDemo {
    public static void main(String[] args) {
    String[] strArray = new String[]{"One", "Two",
    "Three"};
    Enumeration customEnumeration = new
            CustomEnumeration(strArray);
    while (customEnumeration.hasMoreElements()) {
        System.out.println(customEnumeration.nextElement(
        ));
      }
    }
}
```

BitSet Class [1-2]

- Code Snippet demonstrates use of the BitSet class.

Code Snippet

```
package com.datastructures.demo;
import java.util.BitSet;
public class BitSetDemo {
    public static void main(String[] args) {
        BitSet bitSet1 = new BitSet();
        BitSet bitSet2 = new BitSet();
        bitSet1.set(1);
        bitSet1.set(5);
        bitSet1.set(8);
        bitSet2.set(3);
        bitSet2.set(6);
        bitSet2.set(9);
        System.out.println("Values in bitSet1:
        "+bitSet1+"\nValues in bitSet2: "+bitSet2);
        }
    }
```

`BitSet` Class [2-2]

- Following table lists various `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 Class [1-2]

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

Code Snippet

```
package com.datastructures.demo;
import java.util.Stack;
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;
}
```

Stack Class [2-2]

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

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 [1-3]

- 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 [2-3]

The code demonstrates the use of the `Hashtable` class.

Code Snippet

```
package com.datastructures.demo;
import java.util.Enumeration;
import java.util.Hashtable;
public class HashtableDemo {
     private static Hashtable initializeHashtable() {
          Hashtable hTable = new Hashtable();
          hTable.put("1", "East");
          hTable.put("2", "West");
          hTable.put("3", "North");
          hTable.put("4", "South");
          return hTable;
}
    public static void main(String[] args) {
         Hashtable initializedHtable =
                   HashtableDemo.initializeHashtable();
         Enumeration e = initializedHtable.keys();
```

Hashtable Class [3-3]

```
            System.out.println("---Hashtable Key-Value
            Pairs---");
            while (e.hasMoreElements()) {
                String key = (String) e.nextElement();
                System.out.println(key + " : " +
                initializedHtable.get(key));
            }
            e = initializedHtable.keys();
            System.out.println("---Hashtable Keys---");
            while (e.hasMoreElements()) {
                System.out.println(e.nextElement());
            }
            e = initializedHtable.elements();
            System.out.println("---Hashtable Values---");
            while (e.hasMoreElements()) {
              System.out.println(e.nextElement());
            }
        }
}
```

Properties Class [1-3]

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 [2-3]

- The code demonstrates the use of the `Properties` class.

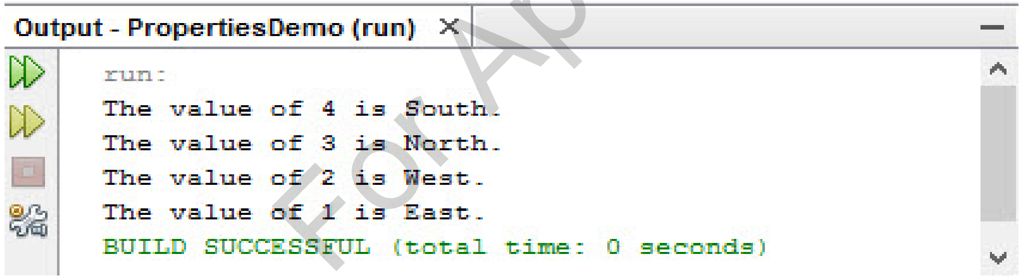
Code Snippet

```
package com.datastructures.demo;
import java.util.Enumeration;
import java.util.Iterator;
import java.util.Properties;
import java.util.Set;
public class PropertiesDemo {
   private static Properties initializeProperties() {
      Properties properties = new Properties();
      properties.setProperty("1", "East");
      properties.setProperty("2", "West");
      properties.setProperty("3", "North");
      properties.setProperty("4", "South");
      return properties;
   }
   public static void main(String[] args) {
      Properties initializedProperties =
          PropertiesDemo.initializeProperties();
```

Properties Class [3-3]

```
        Set set = initializedProperties.keySet();
        Iterator itr = set.iterator();
        while (itr.hasNext()) {
            String str = (String) itr.next();
            System.out.println("The value of "
            + str + " is " +
            initializedProperties.getProperty(str) +
            ".");
       }
     }
}
```

Following is the output of the code:



Output - PropertiesDemo (run)

```
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 time: 0 seconds)
```

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.