Data Structures and Algorithms

Lecture 2

Mark Cudden
Mark.Cudden@ncirl.ie

2. Java Collections Framework

http://download.oracle.com/javase/6/docs/technotes/guides/collections/index.html

Help Documentation on Java Classes:

http://download.oracle.com/javase/6/docs/api/

2.1 Java Collections Framework

Principles

- A collection (container) is simply an object that groups multiple elements into a single unit.
  - It allows us to create an empty collection, add and remove items, and determine whether a particular item occurs in the collection.
- A collections framework is a unified architecture for representing and manipulating collections.
- Collections framework classes are part of `java.util` package
- So, you have to import `java.util.*;` in order to use these classes.
  - http://download.oracle.com/javase/6/docs/api/

2.1 Java Collections Framework

Benefits

- It reduces programming effort:
  - By providing useful data structures and algorithms.
- It increases program speed and quality:
  - By providing high-performance, high-quality implementations of useful data structures and algorithms.
- It reduces the effort to learn and use new APIs
- It reduces effort to design new APIs
- It fosters software reuse

2.1 Java Collections Framework

Example of Java Collections

A set of classes supported by the Java Collection Framework

- List - array
  - Linked List,
  - Vector,
  - ArrayList,
- Map – key/value
  - HashMap,
  - TreeMap,

2.1 Java Collections Framework

Methods provided by Java Collections

Most common methods provided by the majority of Java classes that implement different type of Java Collections

- **add(Object obj)** - Appends the specified element to the end of the collection
- **add(int index, Object obj)** - Inserts the specified element at the specified position in the collection (index>=0, index<size()).
- **remove(Object obj)** - Removes the specified the element from the collection
- **remove(int index)** - Removes the element at the specified position in the collection.

– .

2.1 Java Collections Framework

Methods provided by Java Collections

Most common methods provided by the majority of Java classes that implement different type of Java Collections

- **isEmpty()** - Returns true when the collection is empty
- **indexOf(Object obj)** - Returns the index of the first occurrence of the object,
  - Returns -1 when element is not found
- **size()** - Returns the size of the collection
- **sort (Collection myCollection)** – sorts the collection according to ascending order
- **shuffle(Collection myCollection)** - rearranges the elements from the collection in some random way.

– .

2.2 Java Collections: Vector class

Basic Methods **(java.util.Vector)**

Implements a growable array of objects.
Vector's size can grow or shrink when adding/removing items

- Declare a Vector type object
  - `Vector v`
- Vector( ) - creates an empty vector with size 10 (by default)
  - `v = new Vector();`
- Vector(int initialCapacity) – creates an empty vector with the specified initial capacity
  - `Vector v = new Vector(10);`
- int capacity( ) - Returns the current capacity of the Vector
  - `val = v.capacity();`

2.2 Java Collections: Vector class

Basic Methods (java.util.Vector)

- void add (int index, Object element) - Inserts the specified element at the specified position
  - `v.add(2,”John”);      or      v.add(1, 304);`
- void add(Object o) - Appends the specified object at the end of the Vector
  - `v.add(”Paul”);`
- Object remove (int index) - Removes the element at the specified position ; Returns the element that was removed
  - `String outElem;` `int outVal;`
  - `outElem = v.remove(2);` `outVal = v.remove(2);`
- Object elementAt (int index) – returns the element at the specified position (index)
  - `String elem;`
  - `elem = v.elementAt(2);`

2.2 Java Collections: Vector class

Using Vector

Import the package that allows us to work with Collections

Declare the object “fruits” type Vector

Create the object “fruits” type Vector

Print the size of “fruits”

Add one item to the vector

Sort the items

Shuffle the items

```
import java.util.*;

public class TestVector {
    public static void main(String args[]) {
        Vector fruits;
        fruits = new Vector();
        System.out.println("Vector is empty: " + fruits.isEmpty());
        System.out.println("Vector size: " + fruits.size());

        fruits.add("Strawberry");
        fruits.add("Banana");

        System.out.println("Vector contains " + fruits.toString());

        Collections.sort(fruits);
        System.out.println("After sorting: " + fruits.toString());

        Collections.shuffle(fruits);
        System.out.println("After shuffling:" + fruits.toString());

        }
```

2.2 Java Collections: Vector class

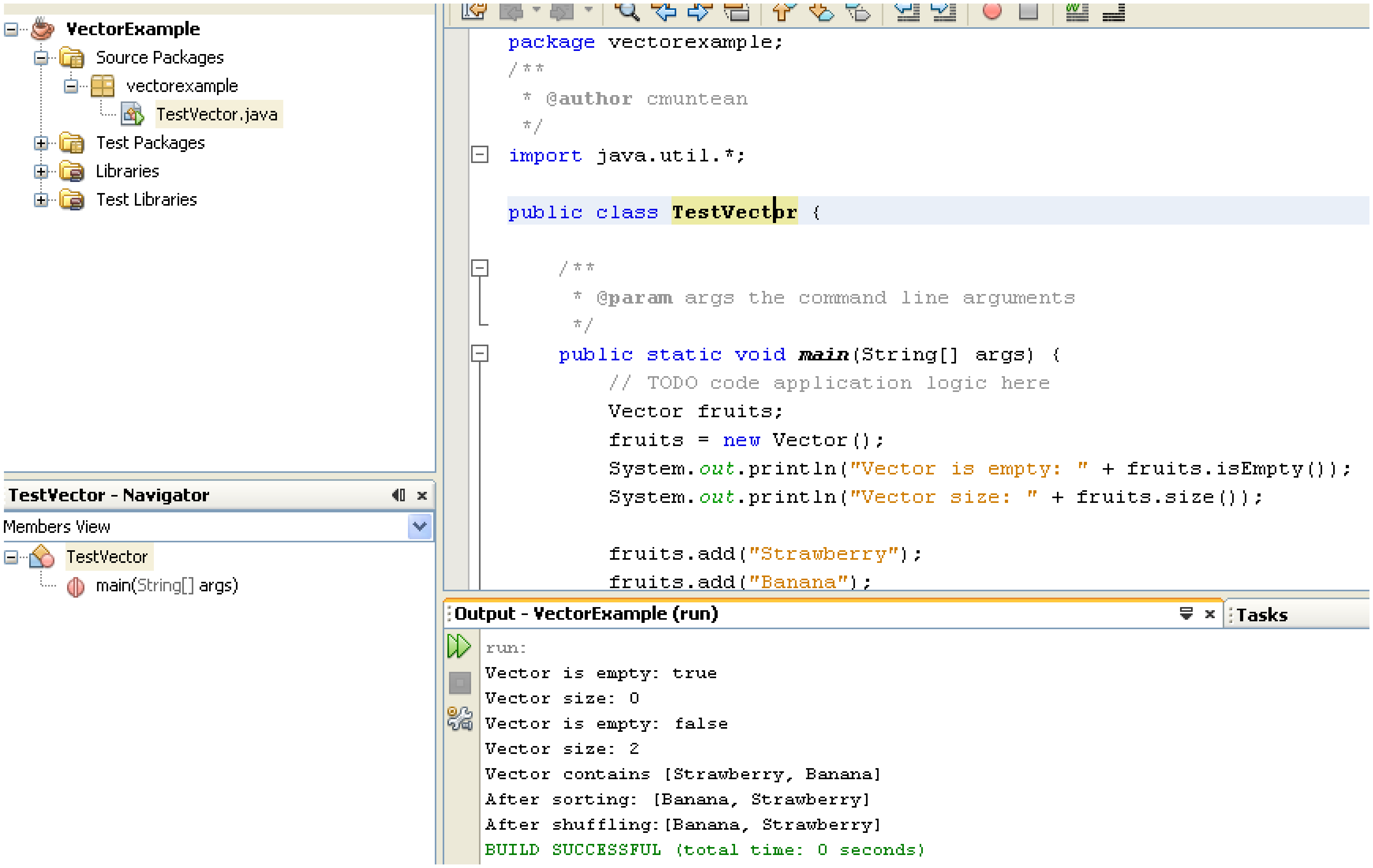
Task: NetBeans application using Vector

- Create a NetBeans project called VectorExample
- Save the project in your home folder (x012344) -> DSA directory
- Set the MainClass name as TestVector

- Edit the TestVector class with Java code that allows you to:
  - Create a Vector type collection object that stores elements (fruits) of String type
  - Add five elements into the Vector type object
  - Print a message if the Vector type object is empty
  - Print the size of your Vector type collection
  - Print all elements
  - Sort the elements from you collection and print them again
  - Remove one element from the Vector and print again all elements

2.2 Java Collections: Vector class

Task: NetBeans application using Vector



```
package vectorexample;
/**
 * @author cmuntean
 */
import java.util.*;

public class TestVector {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        Vector fruits;
        fruits = new Vector();
        System.out.println("Vector is empty: " + fruits.isEmpty());
        System.out.println("Vector size: " + fruits.size());

        fruits.add("Strawberry");
        fruits.add("Banana");
```

```
run:
Vector is empty: true
Vector size: 0
Vector is empty: false
Vector size: 2
Vector contains [Strawberry, Banana]
After sorting: [Banana, Strawberry]
After shuffling:[Banana, Strawberry]
BUILD SUCCESSFUL (total time: 0 seconds)
```

2.3 Iterating over a Collection

- When we want to process the elements from a collection we have to access each element, one by one
- => We iterate though the collection
- `Iterator` type object is assigned to the collection
  - The iterator supports `hasNext()` and `next()` methods
- Code Example for Vector type collection

```
. . .
Vector fruits = new Vector();
fruits.add("Strawberry");
. . .
Iterator i;
i = fruits.iterator();
while (i.hasNext())
{
    System.out.println(“I like “ + i.next());
}
```

Defines an iterator “i”

Assign the iterator “i” to the vector “fruits”

Check if the iteration has more elements to parse (true/false)

Returns the next element from the vector “fruits” in the iteration

2.3 Iterating over a Collection

Main steps

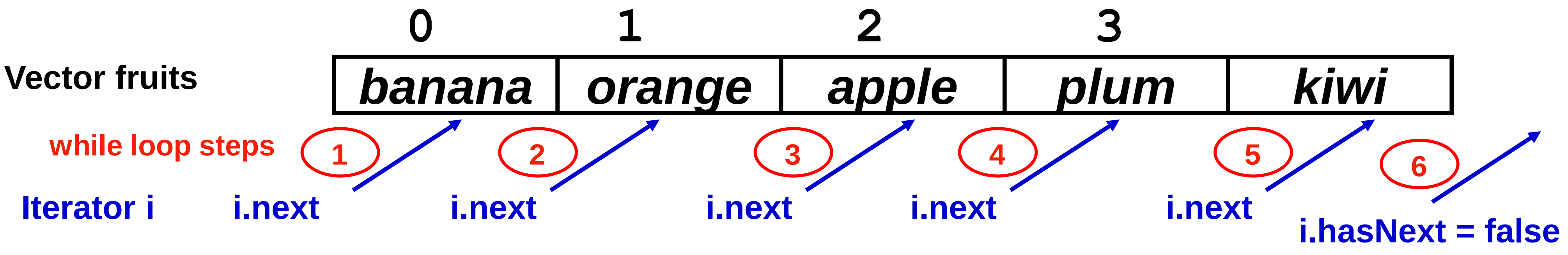
- 1. Associate an iterator to the collection

```
Iterator interatorName  = collectionName.iterator();
```

- 2. Iterate over (parse) the collection using the `while` loop

```
    while (iteratorName.hasNext())
    {
        //access the current element of the collection
    }
```

- 3. Access the current element of the collection (iterator)
  - **`iteratorName.next()`**



2.3 Iterating over a Collection

- A *for* loop may also be used to parse(iterate) though the collection

```
. . .
int j;
for(j = 0; j<fruits.size(); j++)
   System.out.println("Element " + fruits.elementAt(j));
```

Returns the element on position j

Learning Outcome

- Java Collection Framework
- <u>Vector</u> type Java Collection
  - Declare, create, add elements, remove elements, print all elements, sort and shuffle elements, determine the size of the Vector type collection
  - Iterating over a collection

- Lab Exercises
  - Develop more NetBeans applications that work with Vector type collection
  - Practice the mechanism to iterate over a Collection (exemplification on Vector)