
ORACLE
Academy

Java Programming

3-3

Collections – Part 2

ORACLE
Academy

Overview

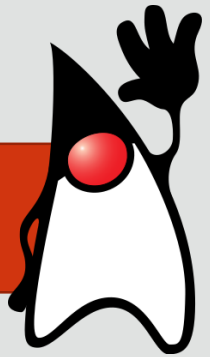
- This lesson covers the following topics:
  - Implement a HashMap
  - Implement a stack by using a deque
  - Define a linked list
  - Define a queue
  - Implement a Comparable interface

ORACLE Academy

Collections

- We have seen previously that we can use ArrayLists and HashSets to store multiple data
- Java has a rich library of collections that will handle a varied list of requirements

Understanding what collections are available and what they offer makes it easier to select the most relevant one.



Maps

- A map is a collection that links a key to a value
- Similar to how an array links an index to a value, a map links a key (one object) to a value (another object)
- Maps, like sets, cannot contain duplicates
- This means each key can only exist once and can only link to a single value
- Since Map is an interface, you must use one of the classes that implement Map such as HashMap to instantiate a map

HashMaps

- HashMaps are maps that link a Key to a Value
- The Key and Value can be of any type, but their types must be consistent for every element in the HashMap
- The KeyType and the ValueType can be the same or different types
- HashMaps, like sets contain no order
- They do allow one null for the key, and multiple nulls for the value

HashMap Methods

Method	Method Description
`boolean containsKey(Object Key)`	Returns true if the HashMap contains the specified Key.
`boolean containsValue(Object Value)`	Returns true if this map maps one or more keys to the specified value.
`V get(Object key)`	Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.
`Set<K> keySet()`	Returns a set of the keys contained in the HashMap.
`Collection<V> values()`	Returns a collection of the values contained in the HashMap.
`V remove(Object Key)`	Removes the mapping for the specified key from this map if present.
`int size()`	Returns the number of key-value mappings in the HashMap.

More methods can be found in the Java API

ORACLE Academy

HashMaps Example

- This is a generic breakdown of how to initialize a HashMap

```
HashMap<KeyType,ValueType> mapName = new HashMap<KeyType,ValueType>();
```

- If a program is required to group together and store many different fruits as well as their colors then a HashMap is an ideal Collection to implement




ORACLE Academy

HashMaps Example

1. Create a project named hashfruits
2. Create a HashMapExample class that includes a main method
3. Initialize a HashMap named fruitBowl that will hold two String values

```
import java.util.HashMap;

public class HashMapExample {

   public static void main(String[] args) {
      HashMap<String,String> fruitBowl = new HashMap<String, String>();

   }//end method main
}//end class HashMapExample
```

Add Fruits to fruitBowl Example

4. Create an AddElements method that accepts the HashMap as a parameter and add an apple using the put(Key,Value) function of the HashMap collection:

```
    public static void main(String[] args) {
        HashMap<String,String> fruitBowl = new HashMap<String, String>();

        addElements(fruitBowl);
    }//end method main

    static void addElements(HashMap<String, String> fruitBowl) {
        fruitBowl.put("Apple", "Green");
    }//end method addElements

}//end class HashMapExample
```

5. Add the following fruits to the fruitBowl

ORACLE Academy

Add Fruits to fruitBowl Example

6. Create a displayElements method that accepts the HashMap as a parameter and will be used to display the values of the HashMap

```
        displayElements(fruitBowl);
    }//end method main

    static void displayElements(HashMap<String, String> fruitBowl) {

    }//end method displayElements

    static void addElements(HashMap<String, String> fruitBowl) {
        fruitBowl.put("Apple", "Green");
        fruitBowl.put("Cherry", "Red");
        fruitBowl.put("Orange", "Orange");
        fruitBowl.put("Banana", "Yellow");
    }//end method addElements
}//end class HashMapExample
```

Add Fruits to fruitBowl Example

7. To simply check the contents of the HashMap you can include its name in an output statement

```
static void displayElements(HashMap<String, String> fruitBowl) {
   System.out.println(fruitBowl);
}//end method displayElements
```

- This will produce the following output:
  - {Apple=Green, Cherry=Red, Orange=Orange, Banana=Yellow}




ORACLE Academy

Add Fruits to fruitBowl Example

8. To format the output to suit your needs you can use an enhanced for loop

```
static void displayElements(HashMap<String, String> fruitBowl) {
   //System.out.println(fruitBowl);
   for (HashMap.Entry<String, String> fruit : fruitBowl.entrySet())
   {
      System.out.println("Fruit: " + fruit.getKey()
                         +" - Color: "+fruit.getValue());
   }//endfor
}//end method displayElements
```

- This uses the entrySet() method that returns a set of the map that can then be displayed:

```
Fruit: Apple  - Color: Green
Fruit: Cherry - Color: Red
Fruit: Orange - Color: Orange
Fruit: Banana - Color: Yellow
```

Add Fruits to fruitBowl Example

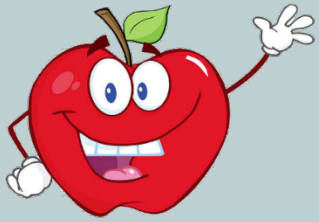
- Remember that a map like a set cannot contain duplicates, the key and value are unique so doing this:
  - hashmapExample.put(1,"b");
  - hashmapExample.put(1,"a");

  Will override the map 1->b to 1->a

9. Add a red apple to the bottom of the add elements method to see that you lose the green apple

ORACLE Academy

```
static void addElements(HashMap<String, String> fruitBowl){
   fruitBowl.put("Apple", "Green");
   fruitBowl.put("Cherry", "Red");
   fruitBowl.put("Orange", "Orange");
   fruitBowl.put("Banana", "Yellow");
   fruitBowl.put("Apple", "Red");
}//end method addElements
```



get(Key) Method of HashMap

- The Key of a HashMap can be thought of as the index linked to the element, even though it does not have to be an integer
- Getting the value stored is easy once it's understood that the key is the index: Use the get(Key) method of the HashMap
- To get the color of the Banana in the fruit bowl, use this method which searches through the HashMap until it finds a Key match to the parameter ("Banana") and returns the Value for that Key ("Yellow")

```
String bananaColor = fruitBowl.get("Banana");
```

Add Fruits to fruitBowl Example

10. Create the following method that will check if the fruit exists in the bowl and then display its value to screen or else display a fruit not found method

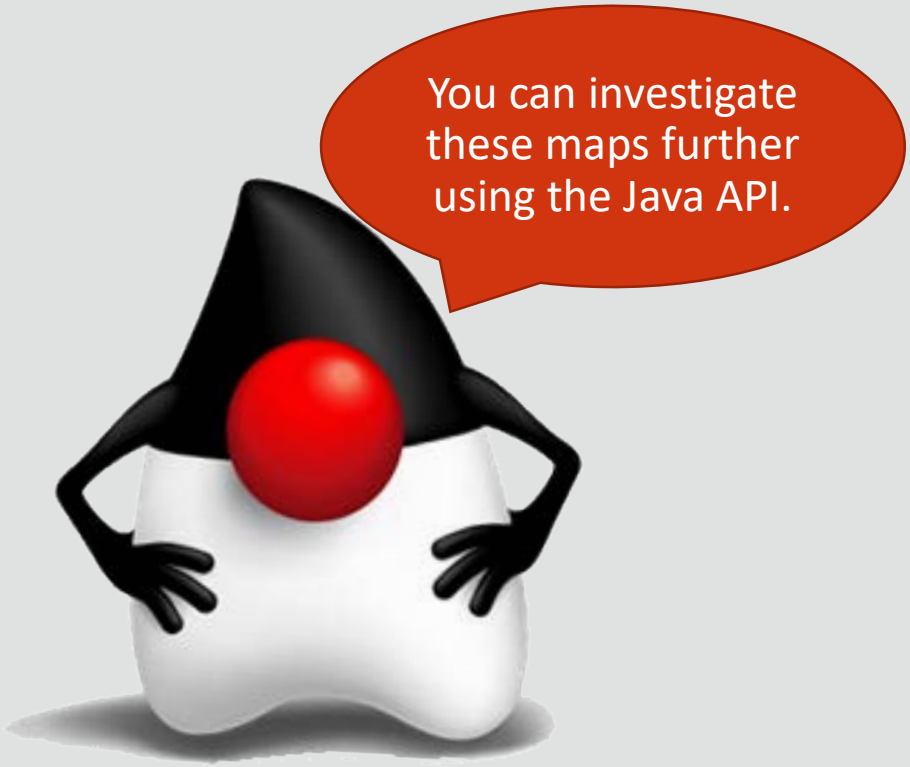
```
   findElement(fruitBowl, "Banana");
}//end method main

static void findElement(HashMap<String, String> fruitBowl,
                        String fruit) {
   if(fruitBowl.containsKey(fruit))
      System.out.println("The " + fruit + " is "
                         + fruitBowl.get(fruit));
   else
      System.out.println("There is no " + fruit + " in the bowl");
   //endif
}//end method findElement
```

- Test the method by using an existing fruit and a "Pear"

Maps

- HashMap is just one of many Map collections.
- There are also:
  - Hashtable
  - EnumMap
  - IdentityHashMap
  - LinkedHashMap
  - Properties
  - TreeMap
  - WeakHashMap



LinkedList Methods

- A Queue is a LinkedList that operates a First In First Out system known as FIFO
- Stacks are a LinkedList that operates a Last In First Out system known as LIFO

FIFO LinkedList Methods	LIFO LinkedList Methods
`add(E e)` Appends the given element to the end of the list.	**`push(E e)`** Pushes an element onto the stack represented by this list.
`removeFirst()` Removes the first element from the list and returns it.	**`pop()`** Pops an element from the stack represented by this list.
`set(int index, E element)` Replaces the element at the specified position in the list.	**`size()`** Returns the number of elements in this list.

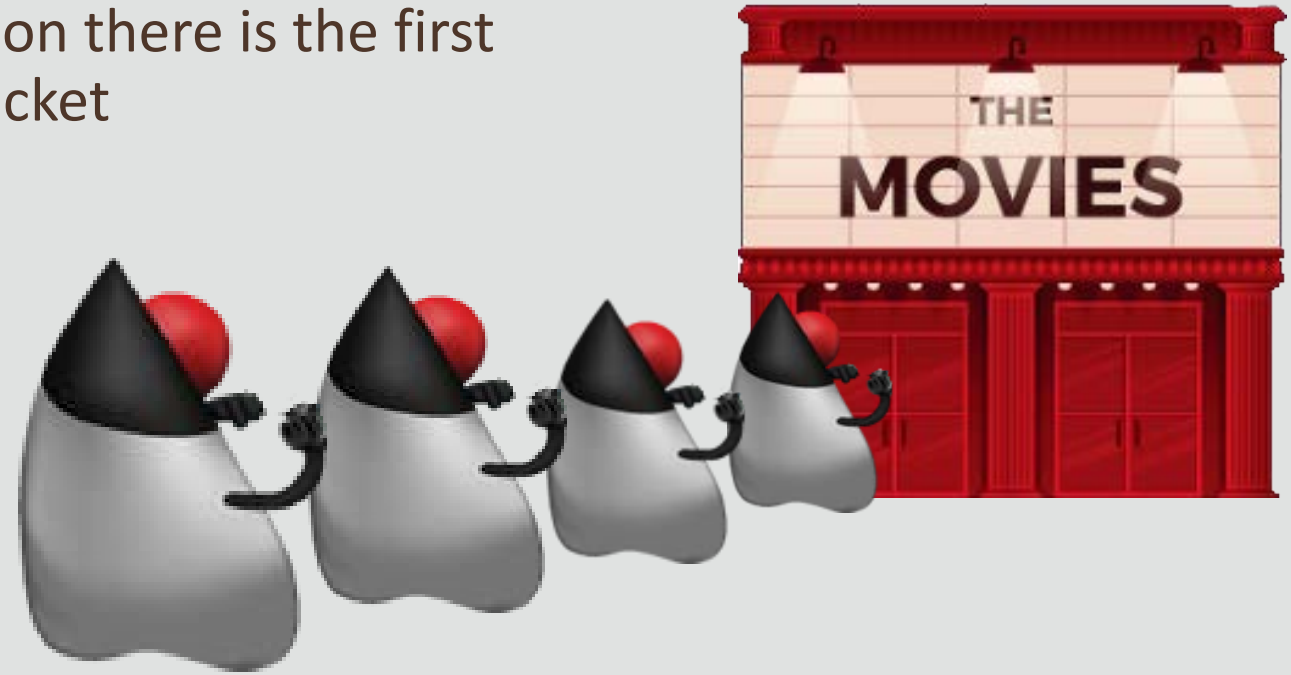
More LinkedList methods can be found in the Java API

LinkedList: Queues

- A Queue is a list of elements with a first in first out ordering (First In First Out, also known as FIFO)
- When you enqueue (Add) an element, it adds it to the end of the list
- When you dequeue (Remove) an element, it returns the element at the front of the list and removes that element from the list

LinkedList: Queues

- Queues are commonly used when the order of adding an object is important, with the first one added being the first one removed
- For example, picture a line at the movie theater
  - The first person there is the first to get their ticket



Using a LinkedList as a Queue

- LinkedList is initialized in the same way as an ArrayList

1. Create a project named lettersqueue
2. Create a class named LettersQueue
3. Initialize a LinkedList of String named lettersQ

```
import java.util.LinkedList;

public class LettersQueue {

   public static void main(String[] args) {
      //This will be implemented as a queue
      LinkedList<String> lettersQ = new LinkedList<String>();
```

Using a LinkedList as a Queue

4. Use the add method to add elements to the Stack

```
//Adding elements to the end of the queue
lettersQ.add("A");
lettersQ.add("B");
lettersQ.add("C");
lettersQ.add("D");
```

5. Display the contents and size of the Queue

```
//display the contents of the linked list
System.out.println("Linked list : " + lettersQ);
//display the size of the linked list
System.out.println("Queue Size: " + lettersQ.size());
```

Using a LinkedList as a Queue

6. Use a while loop to remove the first element from the Queue and display them to the console while the Queue is not empty
7. Display the empty Queue after

```
        //while the Queue is not empty remove each element
        while(!lettersQ.isEmpty()) {
           System.out.println(lettersQ.removeFirst());
        }//endwhile
        //display the contents of the linked list
        System.out.println("Linked list : " + lettersQ);
    }//end method main
}//end class LettersQueue
```

8. Your output should look like this:

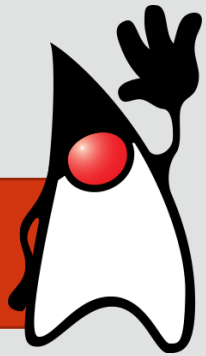
```
Linked list : [A, B, C, D]
Queue Size: 4
A
B
C
D
Linked list : []
```

ORACLE Academy

LinkedList: Stacks

- Stacks are Queues that have reverse ordering to the standard Queue
- Instead of FIFO ordering (like a queue or line at the theater), the ordering of a stack is last in first out
- This can be represented by the acronym LIFO (Last In First Out

The undo method on software typically uses this method where the last action is the one that is used.



Stack of Pancakes Example

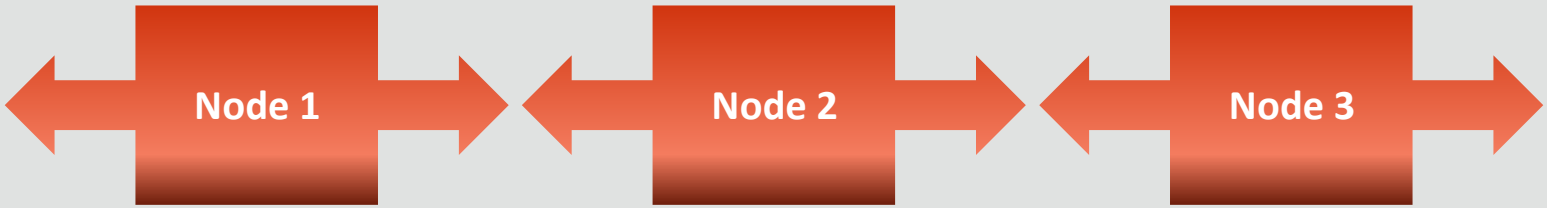
- If there was a pile of pancakes it would be typically called a “stack” of pancakes because the pancakes are added on top of the previous leaving the most recently added pancake at the top of the stack
- To remove a pancake, you would have to take off the one that was most recently added: The pancake on the top of the stack
- If you tried to remove the pancake that was added first, you would most likely make a very large mess

Implementing a Stack: Deque

- One way to implement a Stack is by using a Double-Ended Queue (or deque, pronounced “deck”, for short)
- These allow us to insert and remove elements from either end of the queue using methods inside the Deque class
- Deques like building blocks, allow you to put pieces on the bottom of your structure or on the top, and likewise pull pieces off from the bottom or top
- Deques can be implemented by LinkedLists

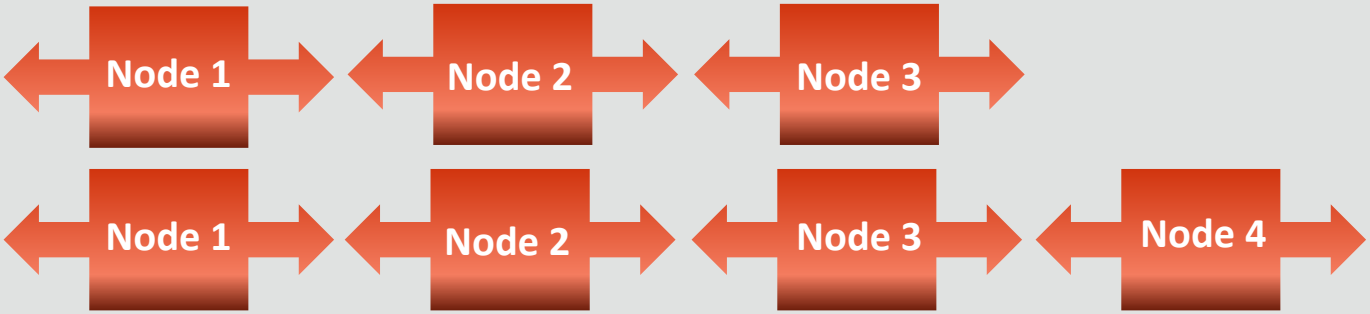
LinkedLists

- A LinkedList is a list of dynamically-stored elements
- Like an ArrayList, it changes size and has an explicit ordering, but it doesn't use an array to store information
- It uses an object known as a Node
- Nodes are like roadmaps: they tell you where you are (the element you are looking at), and where you can go (the previous element and the next element)



Adding Nodes to LinkedLists

- Ultimately, we have a list of Nodes, which point to other Nodes and have an element attached to them
- To add a Node, set its left Node to the one on its left, and its right Node to the one on its right
- Do not forget to change the Nodes around it as well
- A fourth node was added to the end of this linked list:



Using a LinkedList as a Stack

- LinkedList is initialized in the same way as ArrayList

1. Create a project named lettersstack
2. Create a class named LettersStack
3. Initialize a LinkedList of String named lettersStack

```
import java.util.LinkedList;

public class LettersStack {

   public static void main(String[] args) {
      //This will be implemented as a stack
      LinkedList<String> letterS = new LinkedList<String>();
```

Using a LinkedList as a Stack

4. Use the push method to add elements to the Stack

```
//Adding elements to the top of the stack
letterS.push("A");
letterS.push("B");
letterS.push("C");
letterS.push("D");
```

5. Display the contents and size of the Stack

```
//display the contents of the linked list
System.out.println("Linked list : " + letterS);

//display the size of the linked list
System.out.println("Stack Size: " + letterS.size());
```

Using a LinkedList as a Stack

6. Use a while loop to remove (pop) elements from the Stack and display them to the console while the Stack is not empty
7. Display the empty Stack after

```
        //while the stack is not empty remove each element
        while(!letterStack.isEmpty()) {
            System.out.println(letterS.pop());
        }//endwhile
        //display the contents of the linked list
        System.out.println("Linked list : " + letterS);
    }//end method main
}//end class Letters
```

8. Your output should look like this:

```
Linked list : [D, C, B, A]
4
D
C
B
A
Linked list : []
```

ORACLE Academy

Sorting a Collection

- Using the sort method of the Collections class has already been discussed in this course in regards to sorting using simple elements
- In the classlist example the ArrayList of student names were sorted using the Collections.sort() method as they were stored as Strings

```
Collections.sort(studentNames);
```

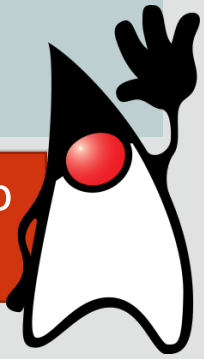
- This sorted the ArrayList in its natural alphabetic order
- Sort is a static method within the class Collections, so doesn't have to be initialized

Sorting a Collection

- This is fine with simple elements but what if the students' details had been stored in a class instead of being represented by a String?
- What if there were additional fields, which field should the students be sorted on?

```
public class Student {
    private String firstName;
    private String lastName;
    private int mark;
        .
}//end class Student
```

A class may have many fields, you would have to decide which field to base the sort on!



Sorting a Collection Example

1. Create a classlistobj project
2. Create a Student class that does not have a main method
3. It should have the following instance fields and constructor:

```
public class Student {
   private String firstName;
   private String lastName;
   private int mark;

   public Student(String firstName, String lastName, int mark) {
      this.firstName = firstName;
      this.lastName = lastName;
      this.mark = mark;
   }//end constructor
}//end class Student
```

Sorting a Collection Example

4. Create getters and setters for all the instance fields

```
public String getFirstName() {
    return firstName;
}//end method getFirstName

public void setFirstName(String firstName) {
    this.firstName = firstName;
}//end method etFirstName

public String getLastName() {
    return lastName;
}//end method getLastName

public void setLastName(String lastName) {
    this.lastName = lastName;
}//end method setLastName
```

Code continued on next slide...

Sorting a Collection Example

4. Create getters and setters for all the instance fields

```
    public int getMark() {
        return mark;
    }//end method getMark

    public void setMark(int mark) {
        this.mark = mark;
    }//end method setMark
}//end class Student
```

Sorting a Collection Example

5. Create a toString method that will return the value of the instance fields:

```
public class Student {
        .
        .
        .

 public String toString() {
    return "Student Details: " + firstName
    + " " + lastName + " " + mark;
    }//end method toString
}//end class Student
```

- The student class will be updated later in this lesson to allow the sorting of students

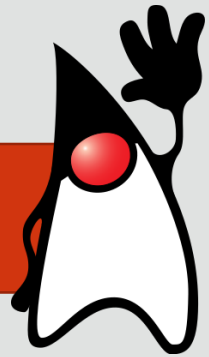
The Comparable Interface

- For the classes to have a natural order the interface java.lang.Comparable can be implemented

```
public interface Comparable<T> {
  int compareTo(T o);
}//end method main
```

- If the Comparable interface was implemented then the code for compareTo must be included in the class
- Eclipse will prompt you to add unimplemented methods to help you with this

The Comparable interface is an excellent example of using interfaces in your design.



compareTo

- The compareTo() method will return an integer based on the following:
  - Return a negative value if this object is smaller than the other object
  - Return 0 (zero) if this object is equal to the other object
  - Return a positive value if this object is larger than the other object
- compareTo() expects an integer value to be returned which can be used to determine the ordering of the objects

Sorting a Collection Example

6. Implement the Comparable interface in the Student class

```
public class Student implements Comparable<Student> {
```

7. When the Comparable interface is implemented it forces the inclusion of the compareTo() method, this will be used for the Collections.sort method

```
public int compareTo(Student stud2) {
   if(lastName.compareTo(stud2.getLastName()) < 0 )
      return -1;
   if(lastName.compareTo(stud2.getLastName()) == 0 )
      return 0;
   return 1;
}//end method compareTo
```

- Sorts the students based on their last name!

Sorting a Collection Example

8. Create a driver class that will add students and then display them to screen

```
public class StudentDriver {
   public static void main(String[] args) {
      ArrayList<Student> studentNames = new ArrayList();

      addStudents(studentNames);
      displayStudents(studentNames);
      Collections.sort(studentNames);
      displayStudents(studentNames);
   }//end method main

   static void displayStudents(ArrayList<Student> studentNames) {
      for(Student student: studentNames)
         System.out.println(student);
      //endfor
   }//end method displayStudents
```

Sorting a Collection Example

9. Create a driver class that will add students and then display them to screen

```
   static void addStudents(ArrayList<Student> studentNames) {
      //Add the following student objects
      studentNames.add(new Student("Mark", "Mywords", 95));
      studentNames.add(new Student("Andrew", "Apic", 45));
      studentNames.add(new Student("Beth", "Tween", 78));
    }//end method addStudents
}//end class StudentDriver
```

10. Run and test your code to test that it works

```
Student Details: Mark Mywords 95
Student Details: Andrew Apic  45
Student Details: Beth Tween   78
Student Details: Andrew Apic  45
Student Details: Mark Mywords 95
Student Details: Beth Tween   78
```

Comparable Interface

- The compareTo method implementation could have used any or multiple fields from our class
- In this example the String field lastName was used to sort the students based on their last name

```
@Override
public int compareTo(Student stud2) {
    if(lastName.compareTo(stud2.getLastName()) < 0 )
       return -1;
    if(lastName.compareTo(stud2.getLastName()) == 0 )
       return 0;
    return 1;
}//end method compareTo
```

Sorting a Collection Example

11. Update the code to sort the students by their mark

```
public int compareTo(Student stud2) {
   if(mark.compareTo(stud2.getMark()) < 0 )
      return -1;
   if(mark.compareTo(stud2.getMark()) == 0 )
      return 0;
   return 1;
}//end method compareTo
```

- When you try to base the compareTo on a primitive data type (mark is an int) you get the following error message:

```
Cannot invoke compareTo(int) on the primitive type int
```

Sorting a Collection Example

12. To compare based on a primitive data type the valueOf method must be invoked
13. This returns an Integer instance based on the value of the primitive
14. Update the code to use this method:

```
public int compareTo(Student stud2) {
   if((Integer.valueOf(mark).compareTo(Integer.valueOf(stud2.getMark()))
       < 0 ))
      return -1;
   if((Integer.valueOf(mark).compareTo(Integer.valueOf(stud2.getMark()))
       == 0 ))
      return 0;
   return 1;
}//end method compareTo
```

Student Output

15. Driver class method calls.

```
    addStudents(studentNames);
    displayStudents(studentNames);
    Collections.sort(studentNames);
    displayStudents(studentNames);
}//end method main
```

- Output

```
Student Details: Mark Mywords 95
Student Details: Andrew Apic  45
Student Details: Beth Tween   78
Student Details: Andrew Apic  45
Student Details: Beth Tween   78
Student Details: Mark Mywords 95
```

Terminology

- Key terms used in this lesson included:
  - Comparable
  - Deque
  - HashMap
  - LinkedList
  - Node
  - Queue
  - Stack

Summary

- In this lesson, you should have learned how to:
  - Implement a HashMap
  - Implement a stack by using a deque
  - Implement Comparable Interface

ORACLE Academy


ORACLE
Academy