CSE 1020: Unit 10

Topics: Collections

To do: Chapter 10, Lab 10

1

Outline

- **Framework**
- **APIs**
- **Method summary**
- **Further usage**

2

Outline

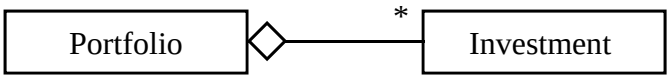
- **Framework**
- APIs
- Method summary
- Further usage

3

Framework

Motivation

- In many cases, an object has a whole collection of components.
- Moreover, the number of components can vary dynamically.
- Examples
  - A course has a collection of students.
  - A portfolio has a collection of investments.

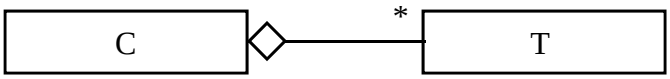


4

Framework

Collection

- An aggregation between an aggregate class C and an aggregated class T is called a collection if, rather than forcing all components to be created with the aggregate, an app is allowed to add/delete components at any time.



5

Framework

Inherent to collections

- Ability to add elements.
- Ability to delete elements.
- Ability to access elements.
  - Individual specification
  - Sequential access

6

Framework

Where we are going

- In Java, there are several mechanisms to deal with collections.
- We already have seen two.
  - arrays
  - the class Vector
- Now, we will examine several others.
  - list
  - set
  - map

7

Framework

Interfaces

- Because there are many plausible approaches to implementing the fundamental collection types (list, set, map) …
- … Java does not force a single implementation to be provided.

8

Framework

Interfaces

- Because there are many plausible approaches to implementing the fundamental collection types (list, set, map) …
- … Java does not force a single implementation to be provided.
- Instead, it specifies the desirable features (attributes and methods) that must be present for each collection type without providing particular implementations.
- This allows the definition of multiple implementing classes for each collection abstraction.

Framework

Interfaces

- Because there are many plausible approaches to implementing the fundamental collection types (list, set, map) …
- … Java does not force a single implementation to be provided.
- Instead, it specifies the desirable features (attributes and methods) that must be present for each collection type without providing particular implementations.
- This allows the definition of multiple implementing classes for each collection abstraction.
- A specification of features without implementation is called an interface.
  - Any class that seeks to implement an interface must satisfy its specification.

10

Framework

Collections: Example usage

- To compare and contrast the various collection approaches, we will see how each realizes the following pseudocode.

```
create collection
loop until sentry
{ prompt user for an element
  read response
  add element to collection
}
report on final collection
```

11

Framework

Lists

- Lists are collections in which:
  - Duplicates are allowed
  - Order is important
- Example
  - A list of things to do today.

12

Framework

Lists

- Lists are collections in which:
  - Duplicates are allowed
  - Order is important
- Example
  - A list of things to do today.
- Java interface of interest:
  - List
- Implementing classes of interest:
  - ArrayList: The default choice.
  - LinkedList: When much insertion/deletion is to be done, especially at front of list.
  - and others…

13

Framework

Lists: Example usage

```
public class ListEg
{ public static void main(String[ ] args)
    { create collection
      loop until sentry
      { prompt user for an element
        read response
        add element to collection
      }
      report on final collection
    }
}
```

Remark: In addition to the usual code from the 1020 template, you also need to import java.util.* to gain access to the various collections used in this and upcoming examples.

14

Framework

Lists: Example usage

```
public class ListEg
{ public static void main(String[ ] args)
    { ArrayList myList = new ArrayList();









    }
}
```

15

Framework

Lists: Example usage

```
public class ListEg
{ public static void main(String[ ] args)
    { ArrayList myList = new ArrayList();
      while (true)
      {



      }


    }
}
```

16

Framework

Lists: Example usage

```
public class ListEg
{ public static void main(String[ ] args)
    { ArrayList myList = new ArrayList();
      while (true)
      { output.print("Enter element (empty line to end): ");
        String str = input.nextLine();


      }


    }
}
```

17

Framework

Lists: Example usage

```
public class ListEg
{ public static void main(String[ ] args)
    { ArrayList myList = new ArrayList();
      while (true)
      { output.print("Enter element (empty line to end): ");
        String str = input.nextLine();
        if (str.length()==0) break; // this is the way out

      }


    }
}
```

18

Framework

Lists: Example usage

```
public class ListEg
{ public static void main(String[ ] args)
    { ArrayList myList = new ArrayList();
      while (true)
      { output.print("Enter element (empty line to end): ");
        String str = input.nextLine();
        if (str.length()==0) break; // this is the way out
        myList.add(str);
      }


    }
}
```

19

Framework

Lists: Example usage

```
public class ListEg
{ public static void main(String[ ] args)
    { ArrayList myList = new ArrayList();
      while (true)
      { output.print("Enter element (empty line to end): ");
        String str = input.nextLine();
        if (str.length()==0) break; // this is the way out
        myList.add(str);
      }
      output.println("Your list is…");
      output.println(myList);
    }
}
```

20

Framework

Lists: Example usage

```
public class ListEg
{ public static void main(String[ ] args)
    { ArrayList myList = new ArrayList();
      while (true)
      { output.print("Enter element (empty line to end): ");
        String str = input.nextLine();
        if (str.length()==0) break; // this is the way out
        myList.add(str);
      }
      output.println("Your list is…");
      output.println(myList);
    }
}
```

21

Framework

Lists: Example usage

```
public class ListEg
{ public static void main(String[ ] args)
    { LinkedList myList = new LinkedList();
      while (true)
      { output.print("Enter element (empty line to end): ");
        String str = input.nextLine();
        if (str.length()==0) break; // this is the way out
        myList.add(str);
      }
      output.println("Your list is…");
      output.println(myList);
    }
}
```

22

Framework

Lists: Example usage

```
public class ListEg
{ public static void main(String[ ] args)
    { LinkedList myList = new LinkedList();
      while (true)
      { output.print("Enter element (empty line to end): ");
        String str = input.nextLine();
        if (str.length()==0) break; // this is the way out
        myList.add(str);
      }
      output.println("Your list is…");
      output.println(myList);
    }
}
```

23

Framework

Lists: Example usage

```
public class ListEg
{ public static void main(String[ ] args)
    { List myList = new LinkedList();
      while (true)
      { output.print("Enter element (empty line to end): ");
        String str = input.nextLine();
        if (str.length()==0) break; // this is the way out
        myList.add(str);
      }
      output.println("Your list is…");
      output.println(myList);
    }
}
```

24

Framework

Lists: Example usage

```
public class ListEg
{ public static void main(String[ ] args)
    { List myList = new LinkedList();
      while (true)
      { output.print("Enter element (empty line to end): ");
        String str = input.nextLine();
        if (str.length()==0) break; // this is the way out
        myList.add(str);
      }
      output.println("Your list is…");
      output.println(myList);
    }
}
```

25

Framework

Lists: Example usage

```
public class ListEg
{ public static void main(String[ ] args)
    { List myList = new ArrayList();
      while (true)
      { output.print("Enter element (empty line to end): ");
        String str = input.nextLine();
        if (str.length()==0) break; // this is the way out
        myList.add(str);
      }
      output.println("Your list is…");
      output.println(myList);
    }
}
```

26

Framework

Lists: Example usage

```
public class ListEg
{ public static void main(String[ ] args)
    { List myList = new List(); // does not work!
      while (true)
      { output.print("Enter element (empty line to end): ");
        String str = input.nextLine();
        if (str.length()==0) break; // this is the way out
        myList.add(str);
      }
      output.println("Your list is…");
      output.println(myList);
    }
}
```

27

Framework

Lists: Example usage

```
% java ListEg
Enter element (empty line ends): X
Enter element (empty line ends): Y
Enter element (empty line ends): Y
Enter element (empty line ends): Z
Enter element (empty line ends):
Your list is…
[X, Y, Y, Z]
%
```

28

Framework

Sets

- Sets are collections in which:
  - Duplicates are not allowed
  - Order is not important
- Example
  - The set of programming languages.

29

Framework

Sets

- Sets are collections in which:
  - Duplicates are not allowed
  - Order is not important
- Example
  - The set of programming languages.
- Java interface of interest:
  - Set
- Implementing classes of interest:
  - HashSet: The default choice.
  - TreeSet: When it is necessary to sort the elements in the set.
  - and others…

30

Framework

Sets: Example usage

```
public class SetEg
{ public static void main(String[ ] args)
    { create collection
      loop until sentry
      { prompt user for an element
        read response
        add element to collection
      }
      report on final collection
    }
}
```

31

Framework

Sets: Example usage

```
public class SetEg
{ public static void main(String[ ] args)
    { HashSet mySet = new HashSet();









    }
}
```

32

Framework

Sets: Example usage

```
public class SetEg
{ public static void main(String[ ] args)
    { HashSet mySet = new HashSet();
      while (true)
      {




      }


    }
}
```

33

Framework

Sets: Example usage

```
public class SetEg
{ public static void main(String[ ] args)
    { HashSet mySet = new HashSet();
      while (true)
      { output.print("Enter elements (empty line ends): ");
        String str = input.nextLine();


      }


    }
}
```

34

Framework

Sets: Example usage

```
public class SetEg
{ public static void main(String[ ] args)
    { HashSet mySet = new HashSet();
      while (true)
      { output.print("Enter elements (empty line ends): ");
        String str = input.nextLine();
        if (str.length()==0) break; // this is the way out

      }


    }
}
```

35

Framework

Sets: Example usage

```
public class SetEg
{ public static void main(String[ ] args)
    { HashSet mySet = new HashSet();
      while (true)
      { output.print("Enter elements (empty line ends): ");
        String str = input.nextLine();
        if (str.length()==0) break; // this is the way out
        mySet.add(str);
      }


    }
}
```

36

Framework

Sets: Example usage

```
public class SetEg
{ public static void main(String[ ] args)
    { HashSet mySet = new HashSet();
      while (true)
      { output.print("Enter elements (empty line ends): ");
        String str = input.nextLine();
        if (str.length()==0) break; // this is the way out
        mySet.add(str);
      }
      output.println("Your set is…");
      output.println(mySet);
    }
}
```

37

Framework

Sets: Example usage

```
public class SetEg
{ public static void main(String[ ] args)
    { HashSet mySet = new HashSet();
      while (true)
      { output.print("Enter elements (empty line ends): ");
        String str = input.nextLine();
        if (str.length()==0) break; // this is the way out
        mySet.add(str);
      }
      output.println("Your set is…");
      output.println(mySet);
    }
}
```

38

Framework

Sets: Example usage

```
public class SetEg
{ public static void main(String[ ] args)
    { TreeSet mySet = new TreeSet();
      while (true)
      { output.print("Enter elements (empty line ends): ");
        String str = input.nextLine();
        if (str.length()==0) break; // this is the way out
        mySet.add(str);
      }
      output.println("Your set is…");
      output.println(mySet);
    }
}
```

39

Framework

Sets: Example usage

```
public class SetEg
{ public static void main(String[ ] args)
    { TreeSet mySet = new TreeSet();
      while (true)
      { output.print("Enter elements (empty line ends): ");
        String str = input.nextLine();
        if (str.length()==0) break; // this is the way out
        mySet.add(str);
      }
      output.println("Your set is…");
      output.println(mySet);
    }
}
```

40

Framework

Sets: Example usage

```
public class SetEg
{ public static void main(String[ ] args)
    { Set mySet = new TreeSet();
      while (true)
      { output.print("Enter elements (empty line ends): ");
        String str = input.nextLine();
        if (str.length()==0) break; // this is the way out
        mySet.add(str);
      }
      output.println("Your set is…");
      output.println(mySet);
    }
}
```

41

Framework

Sets: Example usage

```
public class SetEg
{ public static void main(String[ ] args)
    { Set mySet = new TreeSet();
      while (true)
      { output.print("Enter elements (empty line ends): ");
        String str = input.nextLine();
        if (str.length()==0) break; // this is the way out
        mySet.add(str);
      }
      output.println("Your set is…");
      output.println(mySet);
    }
}
```

42

Framework

Sets: Example usage

```
public class SetEg
{ public static void main(String[ ] args)
    { Set mySet = new HashSet();
      while (true)
      { output.print("Enter elements (empty line ends): ");
        String str = input.nextLine();
        if (str.length()==0) break; // this is the way out
        mySet.add(str);
      }
      output.println("Your set is…");
      output.println(mySet);
    }
}
```

43

Framework

Sets: Example usage

```
public class SetEg
{ public static void main(String[ ] args)
    { Set mySet = new Set(); // does not work!
      while (true)
      { output.print("Enter elements (empty line ends): ");
        String str = input.nextLine();
        if (str.length()==0) break; // this is the way out
        mySet.add(str);
      }
      output.println("Your set is…");
      output.println(mySet);
    }
}
```

44

Framework

Sets: Example usage

```
% java SetEg
Enter element (empty line ends): X
Enter element (empty line ends): Y
Enter element (empty line ends): Y
Enter element (empty line ends): Z
Enter element (empty line ends):
Your set is…
[Z, Y, X]
%
```

45

Framework

Sets: Example usage

```
% java SetEg
Enter element (empty line ends): X
Enter element (empty line ends): Y
Enter element (empty line ends): Y
Enter element (empty line ends): Z
Enter element (empty line ends):
Your set is…
[Z, Y, X]
```

Remark

- The above is the output when the Set object is an instance of HashSet.
- A slightly different result is found when you use TreeSet; try it and see.

46

Framework

Maps

- Maps are collections in which:
  - A pairing is maintained between key/value pairs.
  - Each key must correspond to a single value.
  - There may be multiple keys with the same value.
- Example
  - PIN/person pairings on a given system.

47

Framework

Maps

- Maps are collections in which:
  - A pairing is maintained between key/value pairs
  - Each key must correspond to a single value
  - There may be multiple keys with the same value
- Example
  - PIN/person pairings on a given system.
- Java interface of interest:
  - Map
- Implementing classes of interest:
  - HashMap: The default choice.
  - TreeMap: When it is desirable to sort the elements.
  - and others…

48

Framework

Maps: Example usage

```
public class MapEg
{ public static void main(String[ ] args)
    { create collection
      loop until sentry
      { prompt user for an element
        read response
        add element to collection
      }
      report on final collection
    }
}
```


Framework

Maps: Example usage

```
public class MapEg
{ public static void main(String[ ] args)
    { HashMap myMap = new HashMap();









    }
}
```

Framework

Maps: Example usage

```
public class MapEg
{ public static void main(String[ ] args)
    { HashMap myMap = new HashMap();
      while (true)
      {




      }


    }
}
```


Framework

Maps: Example usage

```
public class MapEg
{ public static void main(String[ ] args)
    { HashMap myMap = new HashMap();
      while (true)
      { output.print("Enter key,value pair (empty line ends): ");
        String str = input.nextLine();
        if (str.length()==0) break; // this is the way out




      }


    }
}
```

Framework

Maps: Example usage

```
public class MapEg
{ public static void main(String[ ] args)
    { HashMap myMap = new HashMap();
      while (true)
      { output.print("Enter key,value pair (empty line ends): ");
        String str = input.nextLine();
        if (str.length()==0) break; // this is the way out
```

Need to parse str into key and value.

```
      }

    }
}
```


Framework

Maps: Example usage

```
public class MapEg
{ public static void main(String[ ] args)
    { HashMap myMap = new HashMap();
      while (true)
      { output.print("Enter key,value pair (empty line ends): ");
        String str = input.nextLine();
        if (str.length()==0) break; // this is the way out
        int comma = str.indexOf(",");

      }

    }
}
```

Framework

Maps: Example usage

```
public class MapEg
{ public static void main(String[ ] args)
    { HashMap myMap = new HashMap();
      while (true)
      { output.print("Enter key,value pair (empty line ends): ");
        String str = input.nextLine();
        if (str.length()==0) break; // this is the way out
        int comma = str.indexOf(",");
        String key = str.substring(0,comma);


      }


    }
}
```

55

Framework

Maps: Example usage

```
public class MapEg
{ public static void main(String[ ] args)
    { HashMap myMap = new HashMap();
      while (true)
      { output.print("Enter key,value pair (empty line ends): ");
        String str = input.nextLine();
        if (str.length()==0) break; // this is the way out
        int comma = str.indexOf(",");
        String key = str.substring(0,comma);
        String value = str.substring(comma+1);
        |
      }


    }
}
```

56

Framework

Maps: Example usage

```
public class MapEg
{ public static void main(String[ ] args)
    { HashMap myMap = new HashMap();
      while (true)
      { output.print("Enter key,value pair (empty line ends): ");
        String str = input.nextLine();
        if (str.length()==0) break; // this is the way out
        int comma = str.indexOf(",");
        String key = str.substring(0,comma);
        String value = str.substring(comma+1);
        myMap.put(key, value);
      }


    }
}
```

57

Framework

Maps: Example usage

```
public class MapEg
{ public static void main(String[ ] args)
    { HashMap myMap = new HashMap();
      while (true)
      { output.print("Enter key,value pair (empty line ends): ");
        String str = input.nextLine();
        if (str.length()==0) break; // this is the way out
        int comma = str.indexOf(",");
        String key = str.substring(0,comma);
        String value = str.substring(comma+1);
        myMap.put(key, value);
      }
      output.println("Your map is…");
      output.println(myMap);
    }
}
```

58

Framework

Maps: Example usage

```
public class MapEg
{ public static void main(String[ ] args)
    { HashMap myMap = new HashMap();
      while (true)
      { output.print("Enter key,value pair (empty line ends): ");
        String str = input.nextLine();
        if (str.length()==0) break; // this is the way out
        int comma = str.indexOf(",");
        String key = str.substring(0,comma);
        String value = str.substring(comma+1);
        myMap.put(key, value);
      }
      output.println("Your map is…");
      output.println(myMap);
    }
}
```

59

Framework

Maps: Example usage

```
public class MapEg
{ public static void main(String[ ] args)
    { TreeMap myMap = new TreeMap();
      while (true)
      { output.print("Enter key,value pair (empty line ends): ");
        String str = input.nextLine();
        if (str.length()==0) break; // this is the way out
        int comma = str.indexOf(",");
        String key = str.substring(0,comma);
        String value = str.substring(comma+1);
        myMap.put(key, value);
      }
      output.println("Your map is…");
      output.println(myMap);
    }
}
```

60

Framework

Maps: Example usage

```
public class MapEg
{ public static void main(String[ ] args)
    { TreeMap myMap = new TreeMap();
      while (true)
      { output.print("Enter key,value pair (empty line ends): ");
        String str = input.nextLine();
        if (str.length()==0) break; // this is the way out
        int comma = str.indexOf(",");
        String key = str.substring(0,comma);
        String value = str.substring(comma+1);
        myMap.put(key, value);
      }
      output.println("Your map is…");
      output.println(myMap);
    }
}
```

61

Framework

Maps: Example usage

```
public class MapEg
{ public static void main(String[ ] args)
    { Map myMap = new TreeMap();
      while (true)
      { output.print("Enter key,value pair (empty line ends): ");
        String str = input.nextLine();
        if (str.length()==0) break; // this is the way out
        int comma = str.indexOf(",");
        String key = str.substring(0,comma);
        String value = str.substring(comma+1);
        myMap.put(key, value);
      }
      output.println("Your map is…");
      output.println(myMap);
    }
}
```

62

Framework

Maps: Example usage

```
public class MapEg
{ public static void main(String[ ] args)
    { Map myMap = new TreeMap();
      while (true)
      { output.print("Enter key,value pair (empty line ends): ");
        String str = input.nextLine();
        if (str.length()==0) break; // this is the way out
        int comma = str.indexOf(",");
        String key = str.substring(0,comma);
        String value = str.substring(comma+1);
        myMap.put(key, value);
      }
      output.println("Your map is…");
      output.println(myMap);
    }
}
```

63

Framework

Maps: Example usage

```
public class MapEg
{ public static void main(String[ ] args)
    { Map myMap = new HashMap();
      while (true)
      { output.print("Enter key,value pair (empty line ends): ");
        String str = input.nextLine();
        if (str.length()==0) break; // this is the way out
        int comma = str.indexOf(",");
        String key = str.substring(0,comma);
        String value = str.substring(comma+1);
        myMap.put(key, value);
      }
      output.println("Your map is…");
      output.println(myMap);
    }
}
```

64

Framework

Maps: Example usage

```
public class MapEg
{ public static void main(String[ ] args)
    { Map myMap = new Map(); // does not work
      while (true)
      { output.print("Enter key,value pair (empty line ends): ");
        String str = input.nextLine();
        if (str.length()==0) break; // this is the way out
        int comma = str.indexOf(",");
        String key = str.substring(0,comma);
        String value = str.substring(comma+1);
        myMap.put(key, value);
      }
      output.println("Your map is…");
      output.println(myMap);
    }
}
```

65

Framework

Maps: Example usage

```
% java MapEg
Enter key,value pair (empty line ends): no,non
Enter key,value pair (empty line ends): yes,si
Enter key,value pair (empty line ends): yes,oui
Enter key,value pair (empty line ends):
Your map is…
[no=non, yes=oui]
%
```

66

Framework

Generics: What

- The collection framework supports so called generics.
- Use of generics allows us to restrict the allowable type that the collection can support.
- The syntax is, e.g.,

```
List<type> myList = new ArrayList<type>();
```

67

Framework

Generics: What

- The collection framework supports so called generics.
- Use of generics allows us to restrict the allowable type that the collection can support.
- The syntax is, e.g.,

```
List<type> myList = new ArrayList<type>();
```

Generics: Why

- Components that take Object parameters are very flexible because they can handle any type.
- This flexibility thwarts the benefits of strong typing.
- Generics are a “client-defined” solution to this dilemma.

68

Framework

Lists: Example usage with generics

```
public class ListEg
{ public static void main(String[ ] args)
    { List myList = new ArrayList(); // Anything allowed
      while (true)
      { output.print("Enter element (empty line to end): ");
        String str = input.nextLine();
        if (str.length()==0) break; // this is the way out
        myList.add(str);
      }
      output.println("Your list is…");
      output.println(myList);
    }
}
```

69

Framework

Lists: Example usage with generics

```
public class ListEg
{ public static void main(String[ ] args)
    { List<String> myList = new ArrayList<String>(); // Only Strings allowed
      while (true)
      { output.print("Enter element (empty line to end): ");
        String str = input.nextLine();
        if (str.length()==0) break; // this is the way out
        myList.add(str);
      }
      output.println("Your list is…");
      output.println(myList);
    }
}
```

70

Framework

Lists: Example usage with generics

```
public class ListEg
{ public static void main(String[ ] args)
    { ArrayList<String> myList = new ArrayList<String>(); // Only Strings allowed
      while (true)
      { output.print("Enter element (empty line to end): ");
        String str = input.nextLine();
        if (str.length()==0) break; // this is the way out
        myList.add(str);
      }
      output.println("Your list is…");
      output.println(myList);
    }
}
```

71

Framework

Lists: Example usage with generics

```
public class ListEg
{ public static void main(String[ ] args)
    { ArrayList<String> myList = new ArrayList<String>(); // Only Strings allowed
      while (true)
      { output.print("Enter element (empty line to end): ");
        String str = input.nextLine();
        if (str.length()==0) break; // this is the way out
        myList.add(str);
      }
      output.println("Your list is…");
      output.println(myList);
    }
}
```

Remark: You can similarly modify all the examples we have introduced so far in this unit to make use of generics. Try it and see.

72

Outline

- Framework
- **APIs**
- Method summary
- Further usage

73

APIs

Constructors

- If we look in the APIs for the Java interfaces List, Set and Map…
- …we find that there are no constructors.

74

APIs

Constructors

- If we look in the APIs for the Java interfaces List, Set and Map…
- …we find that there are no constructors.
- The instances of these abstractions are constructed at the level of the implementing class.
- Hence, constructors are only documented for the implementing classes
  - List: ArrayList, LinkedList, …
  - Set: HashSet, TreeSet, …
  - Map: HashMap, TreeMap, …

75

APIs

Constructors: List

- Class ArrayList

```
ArrayList myList = new ArrayList();
```

alternatively

```
List myList = new ArrayList();
```

76

APIs

Constructors: List

- Class ArrayList

  ```
  ArrayList myList = new ArrayList();
  ```

  alternatively

  ```
  List myList = new ArrayList();
  ```

- Class LinkedList

  ```
  LinkedList myList = new LinkedList();
  ```

  alternatively

  ```
  List myList = new LinkedList();
  ```

77

APIs

Constructors: List

- Class ArrayList

  ```
  ArrayList myList = new ArrayList();
  ```

  alternatively

  ```
  List myList = new ArrayList();
  ```

- Class LinkedList

  ```
  LinkedList myList = new LinkedList();
  ```

  alternatively

  ```
  List myList = new LinkedList();
  ```

- There also are other implementations of List (e.g., Vector); however, we will not deal with those here.

78

APIs

Constructors: List

- Class ArrayList

  ```
  ArrayList myList = new ArrayList();
  ```

  alternatively

  ```
  List myList = new ArrayList();
  ```

- Class LinkedList

  ```
  LinkedList myList = new LinkedList();
  ```

  alternatively

  ```
  List myList = new LinkedList();
  ```

- There also are other implementations of List (e.g., Vector); however, we will not deal with those here.
- In general, the implementing constructors may be overloaded.

79

APIs

Constructors: Set

- Class HashSet

  ```
  HashSet mySet = new HashSet();
  ```

  alternatively

  ```
  Set mySet = new HashSet();
  ```

80

APIs

Constructors: Set

- Class HashSet

  HashSet mySet = new HashSet();

  alternatively

  Set mySet = new HashSet();
- Class TreeSet

  TreeSet mySet = new TreeSet();

  alternatively

  Set mySet = new TreeSet();

81

APIs

Constructors: Set

- Class HashSet

  HashSet mySet = new HashSet();

  alternatively

  Set mySet = new HashSet();
- Class TreeSet

  TreeSet mySet = new TreeSet();

  alternatively

  Set mySet = new TreeSet();
- There also are other implementations of Set; however, we will not deal with those here.
- In general, the implementing constructors may be overloaded.

82

APIs

Constructors: Map

- Class HashMap

  HashMap myMap = new HashMap();

  alternatively

  Map myMap = new HashMap();

83

APIs

Constructors: Map

- Class HashMap

  HashMap myMap = new HashMap();

  alternatively

  Map myMap = new HashMap();

- Class TreeMap

  TreeMap myMap = new TreeMap();

  alternatively

  Map myMap = new TreeMap();

84

APIs

Constructors: Map

- Class HashMap

  ```
  HashMap myMap = new HashMap();
  ```

  alternatively

  ```
  Map myMap = new HashMap();
  ```

- Class TreeMap

  ```
  TreeMap myMap = new TreeMap();
  ```

  alternatively

  ```
  Map myMap = new TreeMap();
  ```

- There also are other implementations of Map; however, we will not deal with those here.
- In general, the implementing constructors may be overloaded.

85

APIs

Adding elements

- Inherent to the notion of a collection is the ability to include additional elements.
  - One or more appropriate methods are always provided.

86

APIs

Adding elements

- Inherent to the notion of a collection is the ability to include additional elements.
  - One or more appropriate methods are always provided.
- List (e.g., ArrayList, LinkedList, …):

  boolean add(Object o)

  will always return true as duplicates are allowed (and we are dealing with dynamic allocation); o is placed at the list end.

87

APIs

Adding elements

- Inherent to the notion of a collection is the ability to include additional elements.
  - One or more appropriate methods are always provided.
- List (e.g., ArrayList, LinkedList, …):

  boolean add(Object o)

  will always return true as duplicates are allowed (and we are dealing with dynamic allocation); o is placed at the list end.

  void add(int index, Object element)

  also is provided to support positional ordering in the list context.

88

APIs

Adding elements

- Inherent to the notion of a collection is the ability to include additional elements.
  - One or more appropriate methods are always provided.
- Set (e.g., HashSet, TreeSet, …):

  boolean add(Object o)

  will return false, if o already present (o will not be added); otherwise, return true (o added).

89

APIs

Adding elements

- Inherent to the notion of a collection is the ability to include additional elements.
  - One or more appropriate methods are always provided.
- Map (e.g., HashMap, TreeMap, …):

  Object put(Object key, Object value)

  if key not already present, associates key with value and returns null; otherwise, overwrites the old mapping and returns the old value.

90

APIs

Adding elements

- Example usage

APIs

Adding elements

- Example usage

```
List myList = new ArrayList();
```

APIs

Adding elements

- Example usage

```
List myList = new ArrayList();
Set mySet = new HashSet();
```

APIs

Adding elements

- Example usage

```
List myList = new ArrayList();
Set mySet = new HashSet();
Map myMap = new HashMap();
```

APIs

Adding elements

- Example usage

```
List myList = new ArrayList();
Set mySet = new HashSet();
Map myMap = new HashMap();
String test1 = “Test1”;
String test2 = “Test2”;
```

APIs

Adding elements

- Example usage

```
List myList = new ArrayList();
Set mySet = new HashSet();
Map myMap = new HashMap();
String test1 = “Test1”;
String test2 = “Test2”;
myList.add(test1);
```

APIs

Adding elements

- Example usage

```
List myList = new ArrayList();
Set mySet = new HashSet();
Map myMap = new HashMap();
String test1 = “Test1”;
String test2 = “Test2”;
myList.add(test1);
mySet.add(test1);
```

APIs

Adding elements

- Example usage

```
List myList = new ArrayList();
Set mySet = new HashSet();
Map myMap = new HashMap();
String test1 = “Test1”;
String test2 = “Test2”;
myList.add(test1);
mySet.add(test1);
myMap.put(new Integer(1), test1);
```

APIs

Adding elements

- Example usage

```
List myList = new ArrayList();
Set mySet = new HashSet();
Map myMap = new HashMap();
String test1 = “Test1”;
String test2 = “Test2”;
myList.add(test1);
mySet.add(test1);
myMap.put(new Integer(1), test1);
```

Remark: Remember that both key and value must be of type Object to go into a map.

APIs

Adding elements

- Example usage

```
List myList = new ArrayList();
Set mySet = new HashSet();
Map myMap = new HashMap();
String test1 = “Test1”;
String test2 = “Test2”;
myList.add(test1);
mySet.add(test1);
myMap.put(new Integer(1), test1);
myList.add(test2); // (“Test1”, Test2”)
```

APIs

Adding elements

- Example usage

```
List myList = new ArrayList();
Set mySet = new HashSet();
Map myMap = new HashMap();
String test1 = “Test1”;
String test2 = “Test2”;
myList.add(test1);
mySet.add(test1);
myMap.put(new Integer(1), test1);
myList.add(test2); // (“Test1”, Test2”)
mySet.add(test2); // both “Test1” and “Test2” present, order?
```

APIs

Adding elements

- Example usage

```
List myList = new ArrayList();
Set mySet = new HashSet();
Map myMap = new HashMap();
String test1 = “Test1”;
String test2 = “Test2”;
myList.add(test1);
mySet.add(test1);
myMap.put(new Integer(1), test1);
myList.add(test2); // (“Test1”, Test2”)
mySet.add(test2); // both “Test1” and “Test2” present, order?
myList.add(test2); // (“Test1”, “Test2”, Test2”)
```

APIs

Adding elements

- Example usage

```
List myList = new ArrayList();
Set mySet = new HashSet();
Map myMap = new HashMap();
String test1 = “Test1”;
String test2 = “Test2”;
myList.add(test1);
mySet.add(test1);
myMap.put(new Integer(1), test1);
myList.add(test2); // (“Test1”, Test2”)
mySet.add(test2); // both “Test1” and “Test2” present, order?
myList.add(test2); // (“Test1”, “Test2”, Test2”)
mySet.add(test2); // mySet unchanged, no duplicates allowed
```

APIs

Adding elements

- Example usage

```
List myList = new ArrayList();
Set mySet = new HashSet();
Map myMap = new HashMap();
String test1 = “Test1”;
String test2 = “Test2”;
myList.add(test1);
mySet.add(test1);
myMap.put(new Integer(1), test1);
myList.add(test2); // (“Test1”, Test2”)
mySet.add(test2); // both “Test1” and “Test2” present, order?
myList.add(test2); // (“Test1”, “Test2”, Test2”)
mySet.add(test2); // mySet unchanged, no duplicates allowed
myList.add(0, test2); // (“Test2”, “Test1”, “Test2”, “Test2”)
```

APIs

Adding elements

- Example usage

```
List myList = new ArrayList();
Set mySet = new HashSet();
Map myMap = new HashMap();
String test1 = “Test1”;
String test2 = “Test2”;
myList.add(test1);
mySet.add(test1);
myMap.put(new Integer(1), test1);
myList.add(test2); // (“Test1”, Test2”)
mySet.add(test2); // both “Test1” and “Test2” present, order?
myList.add(test2); // (“Test1”, “Test2”, Test2”)
mySet.add(test2); // mySet unchanged, no duplicates allowed
myList.add(0, test2); // (“Test2”, “Test1”, “Test2”, “Test2”)
mySet.add(1, test2); // compile error, no such HashSet method
```

105

APIs

Adding elements

- Example usage

```
List myList = new ArrayList();
Set mySet = new HashSet();
Map myMap = new HashMap();
String test1 = “Test1”;
String test2 = “Test2”;
myList.add(test1);
mySet.add(test1);
myMap.put(new Integer(1), test1);
myList.add(test2); // (“Test1”, Test2”)
mySet.add(test2); // both “Test1” and “Test2” present, order?
myList.add(test2); // (“Test1”, “Test2”, Test2”)
mySet.add(test2); // mySet unchanged, no duplicates allowed
myList.add(0, test2); // (“Test2”, “Test1”, “Test2”, “Test2”)
mySet.add(1, test2); // compile error, no such HashSet method
myMap.put(new Integer(1), test2); // overwrite of “Test1” by “Test2”
```

106

APIs

Deleting elements

- Inherent to the notion of a collection is the ability to delete elements.
  - One or more appropriate methods are always provided.

107

APIs

Deleting elements

- Inherent to the notion of a collection is the ability to delete elements.
  - One or more appropriate methods are always provided.
- List (e.g., ArrayList, LinkedList, …):

  boolean remove(Object o)

  will return true, if o is found and removed; returns false, if o is not found. If more than one occurrence of o is present, then only the first is removed.

APIs

Deleting elements

- Inherent to the notion of a collection is the ability to delete elements.
  - One or more appropriate methods are always provided.
- List (e.g., ArrayList, LinkedList, …):

  boolean remove(Object o)

  will return true, if o is found and removed; returns false, if o is not found. If more than one occurrence of o is present, then only the first is removed.

  Object remove(int index)

  also is provided to support positional ordering in the list context; removes and returns element at position index, indices of all following elements adjusted.

109

APIs

Deleting elements

- Inherent to the notion of a collection is the ability to delete elements.
  - One or more appropriate methods are always provided.
- Set (e.g., HashSet, TreeSet, …):

  boolean remove(Object o)

  will return true, if o is found and removed; returns false, if o is not found. (Note: Cannot be more than one occurrence of o, as no duplicates in a set.)

110

APIs

Deleting elements

- Inherent to the notion of a collection is the ability to delete elements.
  - One or more appropriate methods are always provided.
- Map (e.g., HashMap, TreeMap, …):

  Object remove(Object key)

  If key is present, then the value associated with it is returned. If key is not present, then return of null.

111

APIs

Deleting elements

- Example usage

112

APIs

Deleting elements

- Example usage

```
/* our previous example left us with
    myList: (“Test2”, “Test1”, “Test2”, “Test2”)
    mySet: has “Test1” and “Test2”
    myMap: 1,”Test2”
*/
```


APIs

Deleting elements

- Example usage

```
/* our previous example left us with
    myList: (“Test2”, “Test1”, “Test2”, “Test2”)
    mySet: has “Test1” and “Test2”
    myMap: 1,”Test2”
*/
myList.remove(test2); // (“Test1”, “Test2”, “Test2”)
```

APIs

Deleting elements

- Example usage

```
/* our previous example left us with
    myList: (“Test2”, “Test1”, “Test2”, “Test2”)
    mySet: has “Test1” and “Test2”
    myMap: 1,”Test2”
*/
myList.remove(test2); // (“Test1”, “Test2”, “Test2”)
mySet.remove(test2); // has “Test1”
```

115

APIs

Deleting elements

- Example usage

```
/* our previous example left us with
    myList: (“Test2”, “Test1”, “Test2”, “Test2”)
    mySet: has “Test1” and “Test2”
    myMap: 1,”Test2”
*/
myList.remove(test2); // (“Test1”, “Test2”, “Test2”)
mySet.remove(test2); // has “Test1”
myMap.remove(new Integer(1)); // size=0
```

116

APIs

Deleting elements

- Example usage

```
/* our previous example left us with
    myList: (“Test2”, “Test1”, “Test2”, “Test2”)
    mySet: has “Test1” and “Test2”
    myMap: 1,”Test2”
*/
myList.remove(test2); // (“Test1”, “Test2”, “Test2”)
mySet.remove(test2); // has “Test1”
myMap.remove(new Integer(1)); // size=0
myList.remove(1); // (“Test1”, “Test2”)
```


APIs

Deleting elements

- Example usage

```
/* our previous example left us with
    myList: (“Test2”, “Test1”, “Test2”, “Test2”)
    mySet: has “Test1” and “Test2”
    myMap: 1,”Test2”
*/
myList.remove(test2); // (“Test1”, “Test2”, “Test2”)
mySet.remove(test2); // has “Test1”
myMap.remove(new Integer(1)); // size=0
myList.remove(1); // (“Test1”, “Test2”)
myList.remove(“Test3”); // okay, return of false
```

APIs

Deleting elements

- Example usage

```
/* our previous example left us with
    myList: (“Test2”, “Test1”, “Test2”, “Test2”)
    mySet: has “Test1” and “Test2”
    myMap: 1,”Test2”
*/
myList.remove(test2); // (“Test1”, “Test2”, “Test2”)
mySet.remove(test2); // has “Test1”
myMap.remove(new Integer(1)); // size=0
myList.remove(1); // (“Test1”, “Test2”)
myList.remove(“Test3”); // okay, return of false
mySet.remove(test2); // okay, return of false
```


APIs

Deleting elements

- Example usage

```
/* our previous example left us with
    myList: (“Test2”, “Test1”, “Test2”, “Test2”)
    mySet: has “Test1” and “Test2”
    myMap: 1,”Test2”
*/
myList.remove(test2); // (“Test1”, “Test2”, “Test2”)
mySet.remove(test2); // has “Test1”
myMap.remove(new Integer(1)); // size=0
myList.remove(1); // (“Test1”, “Test2”)
myList.remove(“Test3”); // okay, return of false
mySet.remove(test2); // okay, return of false
myMap.remove(new Integer(1)); // okay, return of null
```

APIs

Deleting elements

- Example usage

```
/* our previous example left us with
    myList: (“Test2”, “Test1”, “Test2”, “Test2”)
    mySet: has “Test1” and “Test2”
    myMap: 1,”Test2”
*/
myList.remove(test2); // (“Test1”, “Test2”, “Test2”)
mySet.remove(test2); // has “Test1”
myMap.remove(new Integer(1)); // size=0
myList.remove(1); // (“Test1”, “Test2”)
myList.remove(“Test3”); // okay, return of false
mySet.remove(test2); // okay, return of false
myMap.remove(new Integer(1)); // okay, return of null
myList.remove(3); // runtime error
```

121

APIs

Accessing elements

- Inherent to the notion of a collection is the ability to access elements.
- We can distinguish two broad ways to access elements.
  1. Access to a single, particular element.
  2. Access to all elements, via systematic traversal of the entire collection.
- One or more appropriate methods always are provided.

122

APIs

Accessing elements: Access to a single, particular element

- Access to particular element is only defined with respect to the List and Map.

123

APIs

Accessing elements: Access to a single, particular element

- Access to particular element is only defined with respect to the List and Map.
- List

  Object get(int index)

  returns the element at the given index.
  - e.g., (given our running examples)

    myList.get(0); // “Test1”

124

APIs

Accessing elements: Access to a single, particular element

- Access to particular element is only defined with respect to the List and Map.
- List

  Object get(int index)

  returns the element at the given index.
  - e.g., (given our running examples)

    myList.get(0); // “Test1”
- Map

  Object get(Object key)

  returns value associated with key, if key is found; otherwise, returns null.
  - e.g., (given our running examples)

    myMap.get(new Integer(1)); // null

125

APIs

Accessing elements: Systematic traversal of the entire collection

- Given a collection of elements, traversal of all the elements must entail some sort of iteration.
- Correspondingly, in the List and Set interfaces a method is specified

  Iterator iterator()

  that returns an instance of an iterator object.

126

APIs

Accessing elements: Systematic traversal of the entire collection

- Given a collection of elements, traversal of all the elements must entail some sort of iteration.
- Correspondingly, in the List and Set interfaces a method is specified

  Iterator iterator()

  that returns an instance of an iterator object.
- For example (given our running examples):

  Iterator myListIt = myList.iterator();

  Iterator mySetIt = mySet.iterator();

127

APIs

Accessing elements: Systematic traversal of the entire collection

- To enable the actual traversal, two methods are specified for an iterator object

APIs

Accessing elements: Systematic traversal of the entire collection

- To enable the actual traversal, two methods are specified for an iterator object

  boolean hasNext();

  returns true if unvisited elements remain; otherwise, false

‘

APIs

Accessing elements: Systematic traversal of the entire collection

- To enable the actual traversal, two methods are specified for an iterator object

  boolean hasNext();

  returns true if unvisited elements remain; otherwise, false

  Object next();

  returns the next (unvisited) element in the collection; throws an exception, if no unvisited elements remain.

APIs

Accessing elements: Systematic traversal of the entire collection

- To enable the actual traversal, two methods are specified for an iterator object

  boolean hasNext();

  returns true if unvisited elements remain; otherwise, false

  Object next();

  returns the next (unvisited) element in the collection; throws an exception, if no unvisited elements remain.
- For example (given our running examples):

  ```
  while (myListIt.hasNext())
    output.println(myListIt.next());
  ```

  prints all elements in myList (and similarly for mySet).

131

APIs

Accessing elements: Systematic traversal of the entire collection

- Interestingly, the Map interface does not specify an iterator type object. (This is related to the fact that, strictly speaking, a Map is not a Java Collection!)

APIs

Accessing elements: Systematic traversal of the entire collection

- Interestingly, the Map interface does not specify an iterator type object. (This is related to the fact that, strictly speaking, a Map is not a Java Collection!)
- We can get around this limitation in the following fashion.
- The Map interface provides a method

  Set keySet()

  that can be invoked on a Map instance to return a Set instance consisting of all of the keys associated with the Map instance.

APIs

Accessing elements: Systematic traversal of the entire collection

- Interestingly, the Map interface does not specify an iterator type object. (This is related to the fact that, strictly speaking, a Map is not a Java Collection!)
- We can get around this limitation in the following fashion.
- The Map interface provides a method

  Set keySet()

  that can be invoked on a Map instance to return a Set instance consisting of all of the keys associated with the Map instance.
- For example (given our running examples):

  Set keys = myMap.keySet();

134

APIs

Accessing elements: Systematic traversal of the entire collection

- Now, we have a Set; so, we can define an iterator object on the Set.

APIs

Accessing elements: Systematic traversal of the entire collection

- Now, we have a Set; so, we can define an iterator object on the Set.
- For example (given our running examples):

```
Iterator it = keys.iterator();
```

APIs

Accessing elements: Systematic traversal of the entire collection

- Now, we have a Set; so, we can define an iterator object on the Set.
- For example (given our running examples):

```
Iterator it = keys.iterator();
```

- Finally, we can use our iterator object to systematically traverse the keys of the Map

```
while (it.hasNext())
{ // something involving it.next()
}
```

APIs

Accessing elements: Systematic traversal of the entire collection

- Now, we have a Set; so, we can define an iterator object on the Set.
- For example (given our running examples):

```
Iterator it = keys.iterator();
```

- Finally, we can use our iterator object to systematically traverse the keys of the Map

```
while (it.hasNext())
{ // something involving it.next()
}
```

- And use the keys to systematically invoke the get method of the Map

```
while (it.hasNext())
{ output.println(myMap.get(it.next())); // peruse values in myMap
}
```

138

APIs

Accessing elements: Enhanced for loop

- Java provides an enhanced for loop to enable iterating over any collection regardless of type.
- The general syntax is

```
for (T variable : collection)
  { // do something with the current element
  }
```

139

APIs

Accessing elements: Enhanced for loop

- Java provides an enhanced for loop to enable iterating over any collection regardless of type.
- The general syntax is

```
for (T variable : collection)
  { // do something with the current element
  }
```

- For example, to iterate over a <String> list myList

```
for (String s : myList)
  { output.println(s);
  }
```

140

APIs

Accessing elements: Enhanced for loop

- Java provides an enhanced for loop to enable iterating over any collection regardless of type.
- The general syntax is

```
for (T variable : collection)
  { // do something with the current element
  }
```

- For example, to iterate over a <String> list myList

```
for (String s : myList)
  { output.println(s);
  }
```

- or, to iterate over a <String,String> map myMap

```
for (String s : myMap.keySet())
  { output.println(s + ":" + myMap.get(s));
  }
```

141

APIs

Accessing elements (recap)

- Inherent to the notion of a collection is the ability to access elements.
- We can distinguish two broad ways to access elements.
  1. Access to a single, particular element.
  2. Access to all elements, via systematic traversal of the entire collection.
- One or more appropriate methods always are provided.

142

APIs

Sort

- In many situations, it is useful to have the items in a collection sorted according to some *natural ordering* (e.g., numerical, lexicographic, …).
- For current purposes, we will consider a collection sorted, if its elements appear in *non-descending order*.
  - We use the term non-descending (e.g., as opposed to ascending) to allow for duplicates.

143

APIs

Sort

- In many situations, it is useful to have the items in a collection sorted according to some *natural ordering* (e.g., numerical, lexicographic, …).
- For current purposes, we will consider a collection sorted, if its elements appear in *non-descending order*.
  - We use the term non-descending (e.g., as opposed to ascending) to allow for duplicates.
- Given that an inherent property of a list is that order of the elements is significant (e.g., in contrast to sets and maps), sorting makes most sense with respect to a list for the collection types we are considering.

144

APIs

Sort

- The Java Collections class provides a (static) method to sort any instance that implements the List interface.

  static void sort(List list)

145

APIs

Sort

- The Java Collections class provides a (static) method to sort any instance that implements the List interface.

  static void sort(List list)

- Example usage

```
List myList = new ArrayList();
// assume elements added to list so that it is not sorted
output.println(myList); // prints in unsorted order
Collections.sort(myList);
output.println(myList); // prints in sorted order
```

146

APIs

Sort

- For the Java sort method to work properly, the *elements* of the List passed to sort must be defined to have a natural ordering.
- Numbers: numerical value
- Strings: lexicographic
- Dates: chronological

147

APIs

Sort

- For the Java sort method to work properly, the *elements* of the List passed to sort must be defined to have a natural ordering.
- Numbers: numerical value
- Strings: lexicographic
- Dates: chronological
- Other: Make sure that the element's object type has a compareTo method defined in its class (otherwise suffer a runtime error, exception).

148

APIs

Search

- Search is concerned with determining whether or not a given collection contains a specific, given component.
- There are two possible outcomes of a search.
  1. Success: The desired component is found.
  2. Failure: The desired component is not found.

149

APIs

Search

- Search is concerned with determining whether or not a given collection contains a specific, given component.
- There are two possible outcomes of a search.
  1. Success: The desired component is found.
  2. Failure: The desired component is not found.
- We have seen search in conjunction with our earlier discussion of collections in Unit 8.
  - Indeed, we implemented exhaustive search for a specified Investment in a Portfolio collection.
- A similar approach could be used to search through the collections we are considering in the current Unit.

150

APIs

Exhaustive search

- Example implementation

```
List myList = new ArrayList();
// assume list has been populated with elements of type String
```

151

APIs

Exhaustive search

- Example implementation

```
List myList = new ArrayList();
// assume list has been populated with elements of type String
output.print("Enter a String for the search: ");
String target = input.nextLine();
```

152

APIs

Exhaustive search

- Example implementation

```
List myList = new ArrayList();
// assume list has been populated with elements of type String
output.print("Enter a String for the search: ");
String target = input.nextLine();
String result = null;
```

153

APIs

Exhaustive search

- Example implementation

```
List myList = new ArrayList();
// assume list has been populated with elements of type String
output.print("Enter a String for the search: ");
String target = input.nextLine();
String result = null;
Iterator myListIt = myList.iterator();
```

154

APIs

Exhaustive search

- Example implementation

```
List myList = new ArrayList();
// assume list has been populated with elements of type String
output.print("Enter a String for the search: ");
String target = input.nextLine();
String result = null;
Iterator myListIt = myList.iterator();
while (myListIt.hasNext() && result==null)
{


}
```

155

APIs

Exhaustive search

- Example implementation

```
List myList = new ArrayList();
// assume list has been populated with elements of type String
output.print("Enter a String for the search: ");
String target = input.nextLine();
String result = null;
Iterator myListIt = myList.iterator();
while (myListIt.hasNext() && result==null)
{ String temp = (String)  myListIt.next();


}
```

156

APIs

Exhaustive search

- Example implementation

```
List myList = new ArrayList();
// assume list has been populated with elements of type String
output.print("Enter a String for the search: ");
String target = input.nextLine();
String result = null;
Iterator myListIt = myList.iterator();
while (myListIt.hasNext() && result==null)
{ String temp = (String)  myListIt.next();
  if (temp.equals(target))
    result = temp;
}
```

157

APIs

Exhaustive search

- Example implementation

```
List myList = new ArrayList();
// assume list has been populated with elements of type String
output.print("Enter a String for the search: ");
String target = input.nextLine();
String result = null;
Iterator myListIt = myList.iterator();
while (myListIt.hasNext() && result==null)
{ String temp = (String)  myListIt.next();
  if (temp.equals(target))
    result = temp;
}
if (result==null)
  output.println("Search failed");
```

158

APIs

Exhaustive search

- Example implementation

```
List myList = new ArrayList();
// assume list has been populated with elements of type String
output.print("Enter a String for the search: ");
String target = input.nextLine();
String result = null;
Iterator myListIt = myList.iterator();
while (myListIt.hasNext() && result==null)
{ String temp = (String)  myListIt.next();
  if (temp.equals(target))
    result = temp;
}
if (result==null)
  output.println("Search failed");
else // found target
  output.println("Search succeeded");
```

159

APIs

Exhaustive search

- Example implementation: Now with generics

```
List<String> myList = new ArrayList<String>();
// assume list has been populated with elements of type String
output.print("Enter a String for the search: ");
String target = input.nextLine();
String result = null;
Iterator<String> myListIt = myList.iterator();
while (myListIt.hasNext() && result==null)
{ String temp = (String)  myListIt.next();
  if (temp.equals(target))
    result = temp;
}
if (result==null)
  output.println("Search failed");
else // found target
  output.println("Search succeeded");
```

160

APIs

Exhaustive search

- Example implementation: Now with generics

```
List<String> myList = new ArrayList<String>();
// assume list has been populated with elements of type String
output.print("Enter a String for the search: ");
String target = input.nextLine();
String result = null;
Iterator<String> myListIt = myList.iterator();
while (myListIt.hasNext() && result==null)
{ String temp =  myListIt.next();
  if (temp.equals(target))
    result = temp;
}
if (result==null)
  output.println("Search failed");
else // found target
  output.println("Search succeeded");
```


APIs

Search

- For the Java collections we have been studying, specialized search methods are available.

.

APIs

Search

- For the Java collections we have been studying, specialized search methods are available.
- The interfaces List and Set specify an instance method

  boolean contains(Object o)

  which returns true if the object instance has the passed object as an element, else false.

163

APIs

Search

- For the Java collections we have been studying, specialized search methods are available.
- The interfaces List and Set specify an instance method

  boolean contains(Object o)

  which returns true if the object instance has the passed object as an element, else false.
- The Map interface specifies

  boolean containsKey(Object key)

  which returns true if the passed key is present in one of the key,value pairs aggregated by the map, else false.

164

APIs

Search

- For the special case of a List instance, which has been sorted (e.g., by the sort method of Collections)…
- …there is a particularly fast way to search.

APIs

Search

- For the special case of a List instance, which has been sorted (e.g., by the sort method of Collections)…
- …there is a particularly fast way to search.
- The Collections class defines a method

  static int binarySearch(List list, Object x)

  which returns the position of x in list, if x is found, else a negative integer is returned.

APIs

Search

- For the special case of a List instance, which has been sorted (e.g., by the sort method of Collections)…
- …there is a particularly fast way to search.
- The Collections class defines a method

  static int binarySearch(List list, Object x)

  which returns the position of x in list, if x is found, else a negative integer is returned.
- Example usage

```
List myList = new ArrayList()
// somehow populate myList
Collections.sort(myList);
Object target = myList.get(1); // assuming long enough
int pos = Collections.binarySearch(myList, target); // 1
```

Outline

- Framework
- APIs
- **Method summary**
- Further usage

168

Method Summary

LIST	SET	MAP
Adding Elements		
`boolean add(E e)`	`boolean add(E e)`	`V put(K key, V value)`
`void add(int index, E e)`		
Removing Elements		
`boolean remove(E e)`	`boolean remove(E e)`	`V remove(K key)`
`E remove(int index)`		
Accessing an Element		
`E get(int index)`	*none*	`V get(K key)`
Searching the Elements		
`boolean contains(E o)`	`boolean contains(E o)`	`boolean containskey(K key)`
Traversing the Elements		
`Iterator iterator()`	`Iterator iterator()`	`Iterator keySet().iterator()`
invoke on it:	*invoke on it:*	*invoke on it:*
`E next()` `boolean hasNext()`	`E next()` `boolean hasNext()`	`E next()` `boolean hasNext()`
Other methods (available in all three interfaces)		
`equals, size, toString`		
Algorithms for lists only (`static` methods in the `Collections` `class`)		
`binarySearch, copy, fill, reverse, shuffle, sort`		

69

Outline

- Framework
- APIs
- Method summary
- **Further usage**

170

Further example usage

Detecting duplicates: Requirements

- **Input:** Accept a list of items provided by the user.
- **Output:** Return a list of duplicated items in the input.

171

Further example usage

Detecting duplicates: Design

loop over items in list
{

}

172

Further example usage

Detecting duplicates: Design

loop over items in list
{ current = current item of consideration

}

Further example usage

Detecting duplicates: Design

loop over items in list
{ current = current item of consideration
  loop over items in list
  { compare = current item of consideration

  }
}

Further example usage

Detecting duplicates: Design

```
loop over items in list
{   current = current item of consideration
    loop over items in list
    { compare = current item of consideration
      if (current.equals(compare)
          add current to collection of duplicates

    }
}
```

175

Further example usage

Detecting duplicates: Design

```
loop over items in list
{   current = current item of consideration
    loop over items in list
    { compare = current item of consideration
      if (current.equals(compare) && !(current==compare))
          add current to collection of duplicates

    }
}
```

176

Further example usage

Detecting duplicates: Design

```
loop over items in list
{   current = current item of consideration
    loop over items in list
    { compare = current item of consideration
      if (current.equals(compare) && !(current==compare))
          add current to collection of duplicates, but avoid
                duplicating duplicates
    }
}
```

177

Further example usage

Detecting duplicates: Design

```
loop over items in list
{   current = current item of consideration
    loop over items in list
    { compare = current item of consideration
      if (current.equals(compare) && !(current==compare))
          add current to collection of duplicates, but avoid
                duplicating duplicates: use a set
    }
}
```

178

Further example usage

Detecting duplicates: Implementation

```
import type.lang.*;
import java.util.*;
// assume all the usual template material
public class DetectDuplicates
{ public static void main(String[ ] args)
  { // declaration
    // input
    // computation
    // output
  }
}
```

179

Further example usage

Detecting duplicates: Implementation

```
// declaration
List<String> myList = new ArrayList<String>();
Set<String> duplicates = new HashSet<String>();
```

180

Further example usage

Detecting duplicates: Implementation

```
// input
while (true)
{ output.print("Enter element (empty line ends): ");
  String str = input.nextLine();
  if (str.length()==0) break; // this is the way out
  myList.add(str);
}
```

181

Further example usage

Detecting duplicates: Implementation

```
// computation
```

182

Further example usage

Detecting duplicates: Implementation

```
// computation
Iterator<String> it1 = myList.iterator();
```

183

Further example usage

Detecting duplicates: Implementation

```
// computation
Iterator<String> it1 = myList.iterator();
while (it1.hasNext())
{




}
```

184

Further example usage

Detecting duplicates: Implementation

```
// computation
Iterator<String> it1 = myList.iterator();
while (it1.hasNext())
{ String current = it1.next();




}
```

185

Further example usage

Detecting duplicates: Implementation

```
// computation
Iterator<String> it1 = myList.iterator();
while (it1.hasNext())
{ String current = it1.next();
  Iterator<String> it2 = myList.iterator();




}
```

186

Further example usage

Detecting duplicates: Implementation

```
// computation
Iterator<String> it1 = myList.iterator();
while (it1.hasNext())
{ String current = it1.next();
  Iterator<String> it2 = myList.iterator();
  while (it2.hasNext())
  {


  }
}
```

187

Further example usage

Detecting duplicates: Implementation

```
// computation
Iterator<String> it1 = myList.iterator();
while (it1.hasNext())
{ String current = it1.next();
  Iterator<String> it2 = myList.iterator();
  while (it2.hasNext())
  {  String compare = it2.next();


  }
}
```

188

Further example usage

Detecting duplicates: Implementation

```
// computation
Iterator<String> it1 = myList.iterator();
while (it1.hasNext())
{ String current = it1.next();
  Iterator<String> it2 = myList.iterator();
  while (it2.hasNext())
  {  String compare = it2.next();
     if ((current.equals(compare) && (!(current==compare))))

  }
}
```

189

Further example usage

Detecting duplicates: Implementation

```
// computation
Iterator<String> it1 = myList.iterator();
while (it1.hasNext())
{ String current = it1.next();
  Iterator<String> it2 = myList.iterator();
  while (it2.hasNext())
  {  String compare = it2.next();
     if ((current.equals(compare) && (!(current==compare))))
       duplicates.add(current);
  }
}
```

190

Further example usage

Detecting duplicates: Implementation

```
// output
```

191

Further example usage

Detecting duplicates: Implementation

```
// output
if (duplicates.size()==0)
  output.println(“No duplicates detected.”);
```

192

Further example usage

Detecting duplicates: Implementation

```
// output
if (duplicates.size()==0)
  output.println(“No duplicates detected.”);
else
  output.println(“Detected duplicates: ” + duplicates);
```

193

Further example usage

Detecting duplicates: Test

```
% java DetectDuplicates
Enter element (empty line ends): 1
Enter element (empty line ends): 2
Enter element (empty line ends): 2
Enter element (empty line ends): 3
Enter element (empty line ends): 2
Enter element (empty line ends): 1
Enter element (empty line ends): 4
Enter element (empty line ends):
Detected duplicates: [2, 1]
%
```

194

Further example usage

Word frequency: Requirements

- **Input:** Accept a text file.
- **Output:** Return word count for the input file text.

195

Further example usage

Word frequency: Design

loop over lines in file
{

}

196

Further example usage

Word frequency: Design

loop over lines in file
{ loop over words in line
  {

  }
}

Further example usage

Word frequency: Design

loop over lines in file
{ loop over words in line
  { if (current word already in collection)
        increment word count for current word

  }
}

Further example usage

Word frequency: Design

```
loop over lines in file
{ loop over words in line
  { if (current word already in collection)
        increment word count for current word
    else
    {

    }
  }
}
```

199

Further example usage

Word frequency: Design

```
loop over lines in file
{ loop over words in line
  { if (current word already in collection)
        increment word count for current word
    else
    {   add current word to collection
        increment word count for current word
    }
  }
}
```

200

Further example usage

Word frequency: Design

```
loop over lines in file
{ loop over words in line
  { if (current word already in collection)
        increment word count for current word
    else
    {   add current word to collection
        increment word count for current word
    }
  }
}
```

Remarks

- **We must maintain key, value pairs (word, count)**
  → **use a Map**

Further example usage

Word frequency: Design

```
loop over lines in file
{ loop over words in line
  { if (current word already in collection)
        increment word count for current word
    else
    {   add current word to collection
        increment word count for current word
    }
  }
}
```

Remarks

- **We must maintain key, value pairs (word, count)**
  → **use a Map**
- **It would be nice to output the words (keys) in, e.g., alphabetical, order**
  → **keep the Map sorted**
  → **use a TreeMap**

202

Further example usage

Word frequency: Implementation

```
import type.lang.*;
import java.util.*;
// assume all the usual
public class WordFrequency
{ public static void main(String[ ] args) throws java.io.IOException
  { // declaration
    // input
    // computation
    // output
  }
}
```

203

Further example usage

Word Frequency: Implementation

```
// declaration
final String WHITE = " \t\n\f\r"; // tab,newline,form feed,carriage return
final String PUNC = ",;.:\"\'`\\/?![](){}<>";
final String DELIMS = WHITE + PUNC;
```

204

Further example usage

Word Frequency: Implementation

```
// declaration
final String WHITE = " \t\n\f\r"; // tab,newline,form feed,carriage return
final String PUNC = ",;.:\"\'`\\/?![](){}<>";
final String DELIMS = WHITE + PUNC;
Map<String,Integer> wfMap = new TreeMap<String,Integer>();//maintain sort
```

205

Further example usage

Word Frequency: Implementation

```
// input
output.print("Enter name of file for analysis: ");
Scanner myReader = new Scanner(new File(input.nextLine()));
```

206

Further example usage

Word Frequency: Implementation

```
// computation
```

Further example usage

Word Frequency: Implementation

```
// computation
while (myReader.hasNextLine())




}
```

Further example usage

Word Frequency: Implementation

```
// computation
while (myReader.hasNextLine())
{ String line = myReader.nextLine();




}
```

Further example usage

Word Frequency: Implementation

```
// computation
while (myReader.hasNextLine())
{ String line = myReader.nextLine();
  StringTokenizer st = new StringTokenizer(line, DELIMS);




}
```

Further example usage

Word Frequency: Implementation

```
// computation
while (myReader.hasNextLine())
{ String line = myReader.nextLine();
  StringTokenizer st = new StringTokenizer(line, DELIMS);
  while (st.hasMoreTokens())
  {




} }
```

Further example usage

Word Frequency: Implementation

```
// computation
while (myReader.hasNextLine())
{ String line = myReader.nextLine();
  StringTokenizer st = new StringTokenizer(line, DELIMS);
  while (st.hasMoreTokens())
  { String word = st.nextToken().toUpperCase();




} }
```

Further example usage

Word Frequency: Implementation

```
// computation
while (myReader.hasNextLine())
{ String line = myReader.nextLine();
  StringTokenizer st = new StringTokenizer(line, DELIMS);
  while (st.hasMoreTokens())
  { String word = st.nextToken().toUpperCase();
    if (wfMap.containsKey(word))
    {

    }


} }
```

Further example usage

Word Frequency: Implementation

```
// computation
while (myReader.hasNextLine())
{ String line = myReader.nextLine();
  StringTokenizer st = new StringTokenizer(line, DELIMS);
  while (st.hasMoreTokens())
  { String word = st.nextToken().toUpperCase();
    if (wfMap.containsKey(word))
    { int count = wfMap.get(word).intValue();

    }


} }
```

Further example usage

Word Frequency: Implementation

```
// computation
while (myReader.hasNextLine())
{ String line = myReader.nextLine();
  StringTokenizer st = new StringTokenizer(line, DELIMS);
  while (st.hasMoreTokens())
  { String word = st.nextToken().toUpperCase();
    if (wfMap.containsKey(word))
    { int count = wfMap.get(word).intValue();
      wfMap.put(word, new Integer(count+1));
    }


} }
```

Further example usage

Word Frequency: Implementation

```
// computation
while (myReader.hasNextLine())
{ String line = myReader.nextLine();
  StringTokenizer st = new StringTokenizer(line, DELIMS);
  while (st.hasMoreTokens())
  { String word = st.nextToken().toUpperCase();
    if (wfMap.containsKey(word))
    { int count = wfMap.get(word).intValue();
      wfMap.put(word, new Integer(count+1));
    } else
    {
    }
} }
```

Further example usage

Word Frequency: Implementation

```
// computation
while (myReader.hasNextLine())
{ String line = myReader.nextLine();
  StringTokenizer st = new StringTokenizer(line, DELIMS);
  while (st.hasMoreTokens())
  { String word = st.nextToken().toUpperCase();
    if (wfMap.containsKey(word))
    { int count = wfMap.get(word).intValue();
      wfMap.put(word, new Integer(count+1));
    } else
    { wfMap.put(word, new Integer(1));
    }
} }
```

Further example usage

Word Frequency: Implementation

```
// output
output.println("Word count for specified file is as follows.");
```

218

Further example usage

Word Frequency: Implementation

```
// output
output.println("Word count for specified file is as follows.");
Iterator<String> it = wfMap.keySet().iterator();
```

219

Further example usage

Word Frequency: Implementation

```
// output
output.println("Word count for specified file is as follows.");
Iterator<String> it = wfMap.keySet().iterator();
while (it.hasNext())
{


}
```

220

Further example usage

Word Frequency: Implementation

```
// output
output.println("Word count for specified file is as follows.");
Iterator<String> it = wfMap.keySet().iterator();
while (it.hasNext())
{ String key = it.next();
  Integer value = wfMap.get(key);
  output.println(key + " " + value);
}
```

221

Further example usage

Word Frequency: Test

% more fear.txt

222

Further example usage

Word Frequency: Test

```
% more fear.txt
You have nothing to fear,
but fear itself.
%
```

223

Further example usage

Word Frequency: Test

```
% more fear.txt
You have nothing to fear,
but fear itself.
% java WordFrequency
```

224

Further example usage

Word Frequency: Test

% more fear.txt
You have nothing to fear,
but fear itself.
% java WordFrequency
Enter name of file for analysis:

Further example usage

Word Frequency: Test

% more fear.txt
You have nothing to fear,
but fear itself.
% java WordFrequency
Enter name of file for analysis: fear.txt

Further example usage

Word Frequency: Test

```
% more fear.txt
You have nothing to fear,
but fear itself.
% java WordFrequency
Enter name of file for analysis: fear.txt
Word count for specified file is as follows.
BUT 1
FEAR 2
HAVE 1
ITSELF 1
NOTHING 1
TO 1
YOU 1
```

227

Summary

- **Framework**
- **APIs**
- **Method summary**
- **Further usage**

228