

Generic Collections

"Overview, Collection Interfaces, Collection Classes, Examples"
Advanced in Programming

Shakirullah Waseeb
shakir.waseeb@gmail.com

Nangarhar University

March 28, 2018



Agenda

- 1 Collections Overview
- 2 Primitive Data Types as Object data type
 - Type-Wrapper-Classes
 - Auto-Boxing, and Auto-Unboxing
- 3 Collection and Collections
- 4 Generic Collection Example
 - Class to be stored in collection
 - Using ArrayList to store Employee collection objects
- 5 Questions and Discussion



What are collections?

- **Collection**: is a data structure (an object), that can hold references to other objects
- **Java collections framework** provides many prebuilt generic data structures
- Collections-framework interfaces [see Figure 1] declare a number of operations (e.g add, compare, remove, get, sort etc.) to be performed generically on various types of collections
- Several implementations of these interfaces are also available in the framework (e.g ArrayList, LinkedList, HashMap, PriorityQueue etc.)

Interface	Description
Collection	The root interface in the collections hierarchy from which interfaces Set, Queue and List are derived.
Set	A collection that does <i>not</i> contain duplicates.
List	An ordered collection that <i>can</i> contain duplicate elements.
Map	A collection that associates keys to values and <i>cannot</i> contain duplicate keys. Map does not derive from Collection.
Queue	Typically a <i>first-in, first-out</i> collection that models a <i>waiting line</i> ; other orders can be specified.

[1]

Figure: Various Interfaces of collection-framework



Which collection to use?

- Choosing collection depends on:
 - required memory
 - methods' performance characteristics for operations such as adding, removing, searching, sorting and many more
- Review documentation of each category of collection before choosing them (e.g List, Set, Queue, and Map etc.)
- Choose appropriate implementation regarding your application's requirements (e.g LinkedList, ArrayList, HashMap, PriorityQueue etc.)



Agenda

- 1 Collections Overview
- 2 Primitive Data Types as Object data type
 - Type-Wrapper-Classes
 - Auto-Boxing, and Auto-Unboxing
- 3 Collection and Collections
- 4 Generic Collection Example
 - Class to be stored in collection
 - Using ArrayList to store Employee collection objects
- 5 Questions and Discussion



Type-Wrapper-Classes

- Java provides type-wrapper-classes corresponding each primitive type as:
 - Integer for int
 - Boolean for bool
 - Short for short
 - Character for char
 - Double for double
 - etc.
- By having these classes we can manipulate primitive types as objects
- This is helpful because data-structures such as collections can't operate on primitive data type
- All numeric type-wrapper classes extend class Number



Agenda

- 1 Collections Overview
- 2 Primitive Data Types as Object data type
 - Type-Wrapper-Classes
 - Auto-Boxing, and Auto-Unboxing
- 3 Collection and Collections
- 4 Generic Collection Example
 - Class to be stored in collection
 - Using ArrayList to store Employee collection objects
- 5 Questions and Discussion



Primitive types to type-wrapper objects

- Java provides boxing and unboxing which facilitates automatic conversion between primitive-type values and type-wrapper objects
- **Boxing conversion** : converts a value of a primitive type to an object of the corresponding type-wrapper class
- **Unboxing conversion** : converts an object of a type-wrapper class to a value of the corresponding primitive type
- These conversions are performed automatically—called **autoboxing** and **auto-unboxing**

Auto-conversion example code snippet

```
Integer[] intArray = new Integer[3]; // create an array of Integer type-wrapper  
int x = 5; // create a primitive-data type variable  
intArray[0] = x; // auto-boxing; assigning a primitive type value to Integer type-wrapper  
x = intArray[0]; // auto-unboxing; assigning an Integer type-wrapper to primitive type value
```



Collection interface and Collections class

- **Collection Interface**: contains bulk operations that can be performed on entire collection, such as *adding*, *clearing*, and *comparing* objects in a collection
Can also be converted into array
In addition, provides a method that return an **Iterator** object that walk through the collection objects and can remove objects from collection during iteration
- **Collections Class** : provides static methods that *search*, *sort*, *shuffle*, *copy*, *fill*, *min*, *max* and other operations on collections
Also provides **wrapper methods** that enables to user collection as a *synchronized collection*
Synchronized collections are for use with multithreading, which enables programs to perform operations in parallel



Agenda

- 1 Collections Overview
- 2 Primitive Data Types as Object data type
 - Type-Wrapper-Classes
 - Auto-Boxing, and Auto-Unboxing
- 3 Collection and Collections
- 4 Generic Collection Example
 - Class to be stored in collection
 - Using ArrayList to store Employee collection objects
- 5 Questions and Discussion



Employee Class

Example code for class Employee

// File Name : Employee.java

```
public class Employee{
    static int total=0;
    int id;
    String name;
    double salary;
    public Employee(String empName, double empSalary){
        total++;
        id = total;
        name = empName;
        salary = empSalary;
    }
    public void display(){
        System.out.printf("Employee name: %s \t salary: %f \n",name, salary);
    }
}
```



Agenda

- 1 Collections Overview
- 2 Primitive Data Types as Object data type
 - Type-Wrapper-Classes
 - Auto-Boxing, and Auto-Unboxing
- 3 Collection and Collections
- 4 Generic Collection Example
 - Class to be stored in collection
 - Using ArrayList to store Employee collection objects
- 5 Questions and Discussion



ArrayList Example: EmployeeDemo Class

Example code for class EmployeeDemo

```
// File Name : EmployeeDemo.java
import java.util.*;
public class EmployeeDemo{
    public static void main(String arg[]){
        ArrayList<Employee> employees = new ArrayList<Employee>();
        Scanner scanner = new Scanner(System.in);
        for(int x=0; x<3; x++){
            System.out.println("Enter employee name for :"+(x+1));
            String empName = scanner.nextLine();
            System.out.printf("Enter salary for employee %d :", (x+1));
            double empSalary = scanner.nextDouble();
            //Employee emp = new Employee(empName,empSalary);
            //employees.add(emp);
            employees.add(new Employee(empName,empSalary));
        }
        for(Employee emp: employees){
            emp.display();
        }
    }
}
```



Your Turn: Time to hear from you!



1



¹<https://fensafitters.files.wordpress.com/2013/07/3d095.jpg>

References

-  P.J. Deitel, H.M. Deitel
Java How to program, 10th Edition .
Prentice Hall, 2015.
-  P.J. Deitel, H.M. Deitel
Java How to program, 9th Edition .
Prentice Hall, 2012.
-  Herbert Schildt
The complete reference Java2, 5th Edition .
McGraw-Hill/Osborne, 2002.

