Chapter 12

How to work with collections and generics

Objectives

Applied

1. Write code that creates and works with an array list or a linked list that stores one or more elements.
2. Write code that creates and works with a hash map or a tree map that stores one or more key-value pairs.
3. Given Java code that uses any of the language elements presented in this chapter, explain what each statement does.

Knowledge

1. Describe the similarities and differences between arrays and collections.
2. Name the two main types of collections defined by the collection framework and explain how they differ.
3. Describe the generics feature and explain how you use it to create typed collections and classes.

Objectives (cont.)

4. Describe how the diamond operator works.
5. Explain what autoboxing is.
6. Explain how the elements in array lists and linked lists are stored.
7. Explain how you would decide whether to use an array list or a linked list for a given application.
8. Explain what a queue is and describe the two basic operations that a queue provides.
9. Describe the main difference between a hash map and a tree map.

How arrays and collections are similar

- Both can store multiple elements of the same type.

How arrays and collections differ

- Arrays are fixed in size and require the programmer to increase the size if necessary. Collections automatically increase their size if necessary.
- Arrays can store primitive types without using wrapper classes. Collections must use wrapper classes to store primitive types.
- Arrays don’t provide methods for operations such as adding, replacing, and removing elements. Collections often provide methods that perform these operations.

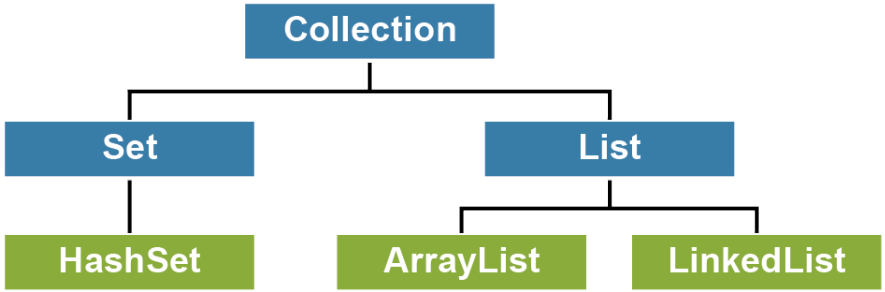
Code that uses an array

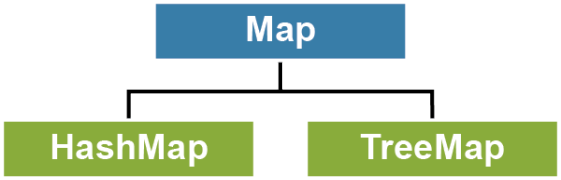
```
String[] codes = new String[3];
codes[0] = "java";
codes[1] = "jsp";
codes[2] = "mysql";
for (String s : codes) {
    System.out.println(s);
}
```

Code that uses a collection

```
ArrayList<String> codes = new ArrayList<>();
codes.add("java");
codes.add("jsp");
codes.add("mysql");
for (String s : codes) {
    System.out.println(s);
}
```

The collection framework





Collection interfaces

- `Collection`
- `Set`
- `List`
- `Map`

Common collection classes

- `ArrayList`
- `LinkedList`
- `HashSet`
- `HashMap`
- `TreeMap`

The syntax for specifying the type of elements in a collection

```
CollectionClass<Type> collectionName =
    new CollectionClass<Type>();
```

Statements that create an array list

Of String objects

```
ArrayList<String> codes = new ArrayList<String>();
```

Of integers

```
ArrayList<Integer> numbers = new ArrayList<Integer>();
```

Of Product objects

```
ArrayList<Product> products = new ArrayList<Product>();
```

The syntax for using type inference (Java 7 or later)

```
CollectionClass<Type> collectionName =
    new CollectionClass<>();
```

Create an array list of String objects

```
ArrayList<String> codes = new ArrayList<>();
```

The ArrayList class

```
java.util.ArrayList
```

Common constructors of the class

```
ArrayList<E>()
ArrayList<E>(intCapacity)
```

Code that creates an array list of String objects

With the default starting capacity of 10 elements

```
ArrayList<String> codes = new ArrayList<>();
```

With a specified starting capacity of 200 elements

```
ArrayList<String> codes = new ArrayList<>(200);
```

Use the List type to store an ArrayList object

```
List<Product> products = new ArrayList<>();
```

Common methods of the ArrayList class

```
add(object)
add(index, object)
get(index)
size()
```

Code that adds three elements to an array list

```
codes.add("jsp");
codes.add("mysql");
codes.add(0, "java");
```

Code that gets the last element

```
int lastIndex = codes.size() - 1;
String lastCode = codes.get(lastIndex);  // "mysql"
```

Code that gets and displays each element

```
for (String code : codes) {
    System.out.println(code);
}
```

Resulting output

```
java
jsp
mysql
```

An easy way to display a collection

```
System.out.println(codes);
```

Resulting output

```
[java, jsp, mysql]
```

More methods of the ArrayList class

```
clear()
contains(object)
indexOf(object)
isEmpty()
remove(index)
remove(object)
set(index, object)
toArray()
```

Code that replaces an element

```
codes.set(2, "andr");
System.out.println(codes);
```

Resulting output

```
[java, jsp, andr]
```

Code that removes an element

```
String code = codes.remove(1);   // removes "jsp"
System.out.println("'" + code + "' was removed.");
System.out.println(codes);
```

Resulting output

```
'jsp' was removed.
[java, andr]
```

Code that searches for an element

```
String searchCode = "andr";
if (codes.contains(searchCode)) {
    System.out.println("This list contains: '" +
        searchCode + "'");
}
```

Resulting output

```
This list contains: 'andr'
```

Code that stores primitive types in an array list

```
ArrayList<Integer> numbers = new ArrayList<>();
numbers.add(1);
numbers.add(2);
numbers.add(3);
System.out.println(numbers);
```

Resulting output

```
[1, 2, 3]
```

Code that gets a primitive type from an array list

```
int firstNumber = numbers.get(0);  // 1
```

Code that gets primitive types from an array list

```
for (int number : numbers) {
    System.out.println(number);
}
```

Resulting output

```
1
2
3
```

The console for the Invoice application

```
Welcome to the Invoice application

Enter product code: java
Enter quantity:     2
Another line item? (y/n): y

Enter product code: jsp
Enter quantity:     1
Another line item? (y/n): n

Description                     Price      Qty  Total
Murach's Java Programming       $57.50     2    $115.00
Murach's Java Servlets and JSP  $57.50     1    $57.50

Invoice total: $172.50
```

The Invoice class

```
package murach.business;

import java.text.NumberFormat;
import java.util.List;
import java.util.ArrayList;

public class Invoice {
    // the instance variable
    private List<LineItem> lineItems;

    // the constructor
    public Invoice() {
        lineItems = new ArrayList<>();
    }

    // a method that adds a line item
    public void addItem(LineItem lineItem) {
        lineItems.add(lineItem);
    }
```

The Invoice class (cont.)

```
    // the get accessor for the line item collection
    public List<LineItem> getLineItems() {
        return lineItems;
    }

    // a method that gets the invoice total
    public double getTotal() {
        double invoiceTotal = 0;
        for (LineItem lineItem : lineItems) {
            invoiceTotal += lineItem.getTotal();
        }
        return invoiceTotal;
    }

    // returns the invoice total in currency format
    public String getTotalFormatted() {
        NumberFormat currency =
            NumberFormat.getCurrencyInstance();
        return currency.format(getTotal());
    }
}
```

The InvoiceApp class

```
package murach.ui;

import murach.db.ProductDB;
import murach.business.Invoice;
import murach.business.LineItem;
import murach.business.Product;

public class InvoiceApp {

    public static void main(String args[]) {
        System.out.println(
            "Welcome to the Invoice application\n");
        Invoice invoice = new Invoice();
        getLineItems(invoice);
        displayInvoice(invoice);
    }
```

The InvoiceApp class (cont.)

```
public static void getLineItems(Invoice invoice) {
    String choice = "y";
    while (choice.equalsIgnoreCase("y")) {
        String productCode = Console.getString(
            "Enter product code: ");
        int quantity = Console.getInt(
            "Enter quantity:     ");

        Product product =
            ProductDB.getProduct(productCode);
        invoice.addItem(
            new LineItem(product, quantity));

        choice = Console.getString(
            "Another line item? (y/n): ");
        System.out.println();
    }
}
```

The InvoiceApp class (cont.)

```
    public static void displayInvoice(Invoice invoice) {
        System.out.println(
            "Code\tDescription\t\t\tPrice\tQty\tTotal");
        System.out.println(
            "----\t-----------\t\t\t-----\t---\t-----");
        for (LineItem lineItem : invoice.getLineItems()) {
            Product product = lineItem.getProduct();
            String s = product.getCode()
                + "\t" + product.getDescription()
                + "\t" + product.getPriceFormatted()
                + "\t" + lineItem.getQuantity()
                + "\t" + lineItem.getTotalFormatted();
            System.out.println(s);
        }
        System.out.println("\n\t\t\t\t\tInvoice total:\t"
            + invoice.getTotalFormatted() + "\n");
    }
}
```

The LinkedList class

```
java.util.LinkedList
```

A constructor of the class

```
LinkedList<E>()
```

Code that creates a linked list of String objects

```
LinkedList<String> codes = new LinkedList<>();
```

Common methods of the LinkedList class

```
add(object)
add(index, object)
get(index)
size()
```

Code that adds three elements to the linked list

```
codes.add("mysql");
codes.add("jsp");
codes.add(1, "java");
System.out.println(codes);
```

Resulting output

```
[mysql, java, jsp]
```

Code that gets the last element of the linked list

```
int lastIndex = codes.size() - 1;
String lastCode = codes.get(lastIndex);    // "jsp"
```

Code that displays each element of the linked list

```
for (String code : codes) {
    System.out.println(code);
}
```

Resulting output

```
mysql
java
jsp
```

More methods of the LinkedList class

```
clear()
contains(object)
indexOf(object)
isEmpty()
remove(index)
remove(object)
set(index, object)
toArray()
```

Code that replaces an element

```
codes.set(2, "andr");
System.out.println(codes);
```

Resulting output

```
[mysql, java, andr]
```

Code that removes an element

```
String code = codes.remove(1);   // removes "java"
System.out.println("'" + code + "' was removed.");
System.out.println(codes);
```

Resulting output

```
'java' was removed.
[mysql, andr]
```

Code that searches for an element

```
String searchCode = "andr";
if (codes.contains(searchCode)) {
    System.out.println(
        "This list contains: '" + searchCode + "'");
}
```

Resulting output

```
This list contains: 'andr'
```

Methods of the Queue and Deque interfaces implemented by the LinkedList class

```
addFirst(object)
addLast(object)
remove()
removeFirst()
removeLast()
element()
getFirst()
getLast()
poll()
peek()
offer(object)
```

Add elements to the beginning and end of the list

```
codes.addFirst("java");
codes.addLast("jscr");
System.out.println(codes);
```

Resulting output

```
[java, mysql, andr, jscr]
```

Remove the last element of the list

```
String lastCode = codes.removeLast();
System.out.println(lastCode);
System.out.println(codes);
```

Resulting output

```
jscr
[java, mysql, andr]
```

Clear the list and then try to get the first element

```
String firstCode;
codes.clear();
firstCode = codes.getFirst();
                          // throws NoSuchElementException
firstCode = codes.peek();
                          // returns null
```

Methods of a queue

- `enqueue(element)`
- `dequeue()`
- `size()`

A class that uses generics to implement a queue

```
import java.util.LinkedList;

public class Queue<E> {
    private LinkedList<E> list = new LinkedList<>();

    public void enqueue(E item) {
        list.addLast(item);
    }

    public E dequeue() {
        return list.removeFirst();
    }

    public int size() {
        return list.size();
    }
}
```

Code that uses the Queue class

```
Queue<String> invoices = new Queue<>();
invoices.enqueue("Invoice 1");
invoices.enqueue("Invoice 2");
invoices.enqueue("Invoice 3");
System.out.println("The queue contains " +
    invoices.size() + " invoices");
while (invoices.size() > 0) {
    String invoice = invoices.dequeue();
    System.out.println("Processing: " + invoice);
}
System.out.println("The queue contains " +
invoices.size() + " invoices");
```

Resulting output

```
The queue contains 3 invoices
Processing: Invoice 1
Processing: Invoice 2
Processing: Invoice 3
The queue contains 0 invoices
```

The HashMap and TreeMap classes

```
java.util.HashMap
java.util.TreeMap
```

Common constructors

```
HashMap<K,V>()
TreeMap<K,V>()
```

Common methods

```
clear()
containsKey(key)
containsValue(value)
entrySet()
get(key)
put(key, value)
remove(key)
size()
```

Common methods of the Map.Entry interface

```
getKey()
getValue()
```

Code that uses a hash map

```
// create an empty hash map
Map<String,String> books = new HashMap<>();

// add three entries
books.put("jscr", "Murach's JavaScript and jQuery");
books.put("andr", "Murach's Android Programming");
books.put("java", "Murach's Java Programming");

// print the entries
for (Map.Entry book : books.entrySet()) {
    System.out.println(book.getKey() + ": " +
        book.getValue());
}

// print the entry whose key is "java"
System.out.println("\nCode java is " +
    books.get("java"));
```

Resulting output from the hash map

```
java: Murach's Java Programming
andr: Murach's Android Programming
jscr: Murach's JavaScript and jQuery

Code java is Murach's Java Programming
```

Code that uses a tree map

```
// create an empty tree map
Map<String,String> books = new TreeMap<>();

// add three entries
books.put("jscr", "Murach's JavaScript and jQuery");
books.put("andr", "Murach's Android Programming");
books.put("java", "Murach's Java Programming");

// print the entries
for (Map.Entry book : books.entrySet()) {
    System.out.println(book.getKey() + ": " +
        book.getValue());
}

// print the entry whose key is "java"
System.out.println("\nCode java is " +
    books.get("java"));
```

Resulting output from the tree map

```
andr: Murach's Android Programming
java: Murach's Java Programming
jscr: Murach's JavaScript and jQuery

Code java is Murach's Java Programming
```

The console for the Word Counter application

```
The Word Counter application

be: 2
is: 1
not: 1
or: 1
question: 1
that: 1
the: 1
to: 2
```

The code for the Word Counter application

```
import java.util.Map;
import java.util.TreeMap;

public class WordCounterApp {

    public static void main(String[] args) {
        System.out.println(
            "The Word Counter application\n");

        // define a string that contains text
        String text =
            "To be or not to be, that is the question.";

        // process the string
        text = text.replace(",", "");  // remove commas
        text = text.replace(".", "");  // remove periods
        text = text.toLowerCase();     // to lower case

        // split the string into an array
        String[] words = text.split(" ");
```

The code for the Word Counter application (cont.)

```
        // define map and fill with words and their counts
        Map<String,Integer> wordMap = new TreeMap<>();
        int count;
        for (String word : words) {
            if (wordMap.containsKey(word)) { // word in map
                count = wordMap.get(word);
                count++;
                wordMap.put(word, count);
            } else {                              // new word
                wordMap.put(word, 1);
            }
        }

        // print the entries
        for (Map.Entry entry : wordMap.entrySet()) {
            System.out.println(entry.getKey() + ": " +
                entry.getValue());
        }
    }
}
```