Mastering Java Fundamentals

Student Workbook 3

Version 2.0

Table of Contents

Module 1

File I/O Reading

Section 1–1

Handling Exceptions

Errors and Java

- **As you've no doubt seen by now, there are three types of errors a Java developer has to contend with**
  - syntax errors
  - logic errors
  - runtime errors
- **You can't even try to run a program when there are syntax errors**
  - The compiler points them out and you fix them
- **With logic errors, you wonder why the program is behaving as it is -- all the while it is doing exactly what you told it to do**
- **With runtime errors, the program has come to a screeching halt and you have to figure out why**
  - Sometimes this happens with bad input, or a missing file, or other unexpected reasons

Exceptions and `try/catch`

- **With runtime errors, Java throws an exception**
  - The type of the exception varies depending on what is wrong
- **You wouldn't want this sudden halt to happen to a program in production do developers can use something called `try/catch` statements to write to intercept the exception**
  - You might not be able to prevent the exception, but you can handle it somehow and have the program continue executing
- **The `try` statement allows you to define a block of code that executes**
  - But at the same time, it specifies a `catch` handler that exceptions route to in order to be handled

Syntax

```
try {
  // try to execute this code
}

catch(Exception e) {
  // unhandled exceptions route here
  // and are "handled" and then suppressed
}
```

- **We will examine `try/catch` in more detail after we learn a little more about classes**

Example: Handling Index Out of Bounds

Example

```
Scanner scanner = new Scanner(System.in);

try {
  String[] names = {
        "Ezra", "Elisha", "Ian",
        "Siddalee", "Pursalane", "Zephaniah"
  };

  System.out.print("Pick a kid (select #1 - #6): ");
  int index = scanner.nextInt();

  index--;   // change number from range 1-6 to range 0-5

  // as long as the user entered a number in the range
  // of 1 to 6, this will work.  Otherwise the index
  // will be out of range.

  System.out.println(names[index]);
}

catch (Exception e) {
  System.out.println("Your number was out of range!");
  e.printStackTrace();
}

scanner.close();
```

TRANSCRIPT OF TERMINAL SESSION

```
Pick a kid (select #1 - #6): 12
Your number was out of range!
```

Exercises

Create a new folder in your `java-development` directory. Name it `workbook-3`. This is the folder where you will complete all of your exercises this week.

In this exercise you will create an application to display famous sayings or quotes. A user should be able to select a quote by number, your application should display the corresponding quote.

EXERCISE 1

Create a newJava application named FamousQuotes.

Create an array of strings to store 10 quotes. Add 10 of your favorite quotes to the array.

Prompt the user to select a number between 1 and 10 and display that quote.

DO NOT use `try/catch` (yet) to handle the error.

Test the application. Select a number that exists. Does it work?

Enter a number that doesn't exist (any number other than 1 to 10). What happens?

Now add exception handling to the application and retest.

BONUS: Add a loop to the program asks the user if they want to see another saying. If they say yes, repeat the process. DO NOT shut the application down when if an exception occurs. Just pick up with the next iteration.

BONUS: Allow the user to select an option that will display a random quote.

Commit and push your code!

Section 1–2

Using the `Scanner` Class

Scanner

- **The `Scanner` class from the `java.io` package can be used to read a file**

- **When you create a `Scanner`, have it reference a file by wrapping the name of the file inside a `FileInputStream` object and passing that to the `Scanner` constructor**

Example

```
FileInputStream fis = new FileInputStream("poem.txt");
Scanner scanner = new Scanner(fis);
```

- **With `nextLine()`, you can read a line of text from the file**
  - The `hasNextLine()` method returns `true` if there are more lines in the file

Example

```
while(scanner.hasNextLine()) {
   input = scanner.nextLine();
   System.out.println(input);
}
```

- **When you are finished with the `Scanner`, call its `close()` method so that it releases resources**

Example

```
scanner.close();
```

Example: Using the `Scanner` to Read a File

- **In this example, we use `try/catch` to handle the exception generated if the file isn't found**
  - We will learn more about `try/catch` later

Example

```
import java.io.*;
import java.util.Scanner;

public class Program
{
   public static void main(String args[])
   {
      try
      {
         // create a FileInputStream object pointing to
         // a specific file
         FileInputStream fis = new FileInputStream("poem.txt");

         // create a Scanner to reference the file to be read
         Scanner scanner = new Scanner(fis);

         String input;

         // read until there is no more data
         while(scanner.hasNextLine()) {
            input = scanner.nextLine();
            System.out.println(input);
         }

         // close the scanner and release the resources
         scanner.close();
      }

      catch(IOException e) {
         // display stack trace if there was an error
         e.printStackTrace();
      }
   }
}
```

poem.txt

```
Mary had a little lamb
little lamb
little lamb
Mary had a little lab
it's fleece was white as snow
```

OUTPUT

```
Mary had a little lamb
little lamb
little lamb
Mary had a little lab
it's fleece was white as snow
```

Exercises

Create new subfolders in your `LearningToCode` folder named `Workbook3\Mod01` to hold the exercises for this week. The code for each exercise below should be in its own subfolder under `Mod01`.

In this exercise you will practice using a `Scanner` object to load and read a text file in your java application.

EXERCISE 2

Create a new Java application names BedtimeStories.

Unzip the `DataFiles.zip` file and copy the three children's stories to your BedtimeStories project folder. (`goldilocks.txt`, `hansel_and_gretel.txt` and `mary_had_a_little_lamb.txt`).

Each story text file contains a childrens bedtime story.

For example:

`Goldilocks.txt`

```
Goldilocks and the Three Bears

Once upon a time a girl named Goldilocks lived in
a house at the edge of the woods. In those days
curls of hair were called "locks."  She was
"Goldilocks" because golden hair ran down her head
and shoulders.
```

Prompt the user for the name of the story/file to read.

Use a new `Scanner` to load and read each line of the selected story in the file. Print the file to the screen, but add a line number to the beginning of each line

Example

```
Enter the name of a story: goldilocks.txt


1.  Goldilocks and the Three Bears
2.
3.  Once upon a time a girl named Goldilocks lived in
4.  a house at the edge of the woods. In those days
5.  curls of hair were called "locks."  She was
6.  "Goldilocks" because golden hair ran down her head
7.  and shoulders.
```

Commit and push your code!

Section 1–3

Using `BufferedReader`

BufferedReader

- **The `BufferedReader` from the `java.io` package reads text from a stream of characters**

- **It provides buffering to make the process very efficient**
  - To do this, it reads blocks of characters from the stream and stores them in a buffer while waiting to be read

- **To create a `BufferedReader`, pass a `FileReader` to its constructor**

Example

```
FileReader fileReader = new FileReader("poem.txt");
BufferedReader bufReader = new BufferedReader(fileReader);
```

  - You can do it in one step if you want to

Example

```
BufferedReader bufReader = new BufferedReader(new FileReader("poem.txt"));
```

- **Use `readLine()` to read a line of text from the file**
  - The line must have a `\n` or `\r` at the end of it

Example

```
String input = bufReader.readLine();
```

- **When you reach the end-of-file, `readLine()` returns `null`**
  - This means you can create a `while` loop to read until end-of-file

Example

```
while((input = bufReader.readLine()) != null) {
   System.out.println(input);
}
```

- **When you finish reading the file, you must call `close()` to release the system resources being used**

Example

```
bufReader.close();
```

Example: Reading a File with `BufferedReader`

Example

```
import java.io.*;

public class Program
{
   public static void main(String args[])
   {
      try
      {
         // create a FileReader object connected to the File
         FileReader fileReader = new FileReader("poem.txt");

         // create a BufferedReader to manage input stream
         BufferedReader bufReader = new BufferedReader(fileReader);

         String input;

         // read until there is no more data
         while((input = bufReader.readLine()) != null) {
            System.out.println(input);
         }

         // close the stream and release the resources
         bufReader.close();
      }

      catch(IOException e) {
         // display stack trace if there was an error
         e.printStackTrace();
      }
   }
}
```

poem.txt

```
Mary had a little lamb
little lamb
little lamb
Mary had a little lab
it's fleece was white as snow
```

OUTPUT

```
Mary had a little lamb
little lamb
little lamb
Mary had a little lab
it's fleece was white as snow
```

Exercises

Data is often shared in csv file because the data can be organized into rows and columns. Instead of parsing a text based story, in this exercise you will read the contents of a `.csv` and load the contents into the memory of your java application.

EXERCISE 3

Create a new Java application named PayrollCalculator. You will read employee data from a `.csv` file and generate the pay information for each employee. For this exercise read the file using either the `BufferedReader`.

The file will contain several lines of employee data in the form:

```
id|name|hours-worked|pay-rate
```

For example:

```
10|Dana Wyatt|52.5|12.50
20|Ezra Aiden|17|16.75
30|Brittany Thibbs|40|16.50
40|Zephaniah Hughes|2|10.0
```

Step 1

Create an `Employee` class as described below:

Private data members (attributes):

- `employeeId`
- `name`
- `hoursWorked`
- `payRate`

Methods:

- parameterized constructor
- `getEmployeeId()` and get/set for other attributes
- `getGrossPay()` that calculates and returns that employee's gross pay based on their hours worked and pay rate

Step 2

To make the main program work, you will need to:

1. Load the file using a `FileReader` object
2. Read each line of text
3. Split it into individual fields using the `|` character as the delimiter
4. Copy the values from the tokens array into variables that match the data and then use them to create a new `Employee` object
5. Display the employee using a `printf` and by calling the employee's `getEmployeeId()`, `getName()`, and `getGrossPay()` methods
6. Repeat for each line in the input file

Commit and push your code!

Section 1–4

CodeWars

CodeWars - Cat / Dog Years reversed

- **Cat Years and Dog Years**
  - If you have a cat and a dog, and you know old each is in Cat years and Dog years, calculate how old each is in Human Years
    - Input -> catYears, dogYears
    - Output -> array [catInHumanYears, dogInHumanYears]
  - https://www.codewars.com/kata/5a6d3bd238f80014a2000187/java

Module 2

File I/O Writing

Section 2–1

Writing to Files

FileWriter

- **The FileWriter class is used to write streams of characters to a file**

- **When you instantiate it, pass it the name of the file that it will create**
  - If you pass true as the second argument, it will append to an existing file rather than create a file or overwrite it if it exists

Example

```
import java.io.*;

public class WriterApp1 {

  public static void main(String[] args) {

    try {
      // open the file
      FileWriter writer = new FileWriter("skills.txt");

      // write to the file
      writer.write("Skills:\n");
      writer.write("Git, HTML, CSS, Bootstrap\n");
      writer.write(
              "JavaScript/ES6, jQuery, REST API, Node.js, Express\n");
      writer.write("Angular\n");
      writer.write("Java");

      // close the file when you are finished using it
      writer.close();
    }

    catch (IOException e) {
      System.out.println("ERROR:  An unexpected error occurred");
      e.printStackTrace();
    }
  }
}
```

BufferedWriter

- **The `BufferedWriter` writes text efficiently to a file**
  - It writes to an 8K buffer and only writes the buffer to the file when the buffer is full

Example

```
import java.io.*;

public class WriterApp2 {

  public static void main(String args[]) {

    try {
      // create a FileWriter
      FileWriter fileWriter = new FileWriter("text.txt");

      // create a BufferedWriter
      BufferedWriter bufWriter = new BufferedWriter(fileWriter);

      // write to the file
      String text;
      for(int i = 1; i <= 10; i++) {
         text = String.format("Counting %d\n", i);
          bufWriter.write(text);
      }

      // close the writer
      bufWriter.close();
    }
    catch (IOException e) {
      System.out.println("ERROR:  An unexpected error occurred");
      e.getStackTrace();
    }
  }
}
```

Exercises

EXERCISE 1

Continue working on the PayrollCalculator program.

Rather than displaying your payroll report to the screen, write it to a `.csv` file in the following format.

```
id|name|gross pay
111|Cameron Tay|3277.65
222|James Tee|2150.00
```

Prompt the user for the name of a file to read and process, then prompt them for the name of the payroll file to create.

```
Enter the name of the file employee file to process: employees.csv
Enter the name of the payroll file to create: payroll-sept-2023.csv
```

When your program finishes running, open the new file in Notepad to view the results.

BONUS: If the user chooses specifies a `.json` extension write the data as JSON instead of `csv`.

For example:

```
Enter the name of the file employee file to process: employees.csv
Enter the name of the payroll file to create: payroll-sept-2023.json
```

payroll-sept-2023.json

```
[
  { "id": 111, "name" : "Cameron Tay", "grossPay" : 3277.65 },
  { "id": 222, "name" : "James Tee", "grossPay" : 2150.00 }
]
```

Commit and push your code!

Section 2–2

Date Basics

Java 8 Dates

- **Prior to Java 8 working with Dates was painful in Java**
  - Java 8 introduced a few new Data Type, such as `LocalDate`
  - These data types are much more user friendly for performing date calculations and parsing
- **Java uses features from the `java.time` package to work with the date and time API**
- **It has classes that hold:**
  - only a date (`LocalDate`)
    - Useful for holding birthdays, etc
  - only a time (`LocalTime`)
    - Useful for holding date-independent times, like a store's closing time
  - both a date and a time (`LocalDateTime`)
    - Useful for holding a moment, like a timestamp on a bank deposit or a dentist's appointment
- **Dates are represented as ISO-8601 dates which correspond to the Gregorian calendar**
  - Times are stored as UTC times without any associated time zone

Getting the Current Date and/or Time

- **The local date and time classes have a static method named `now()` that returns the current value for the current date and/or time**

Example

```
import java.time.LocalDate;
LocalDate today = LocalDate.now();
System.out.println("Today is: " + today);
```

OUTPUT

```
Today is: 2021-09-05
```

Example

```
import java.time.LocalTime;
LocalTime currentTime = LocalTime.now();
System.out.println("The current time is: " + currentTime);
```

OUTPUT

```
The current time is: 02:17:11.770918
```

Example

```
import java.time.LocalDateTime;
LocalDateTime rightNow = LocalDateTime.now();
System.out.println("Right now, it is: " + rightNow);
```

OUTPUT

```
Right now, it is: 2021-09-05T02:17:11.771319
```

Example: Working with Dates/Times

Example

```
import java.time.LocalDateTime;

public class MainApp {

   public static void main(String[] args) {

      LocalDateTime today = LocalDateTime.now();
      System.out.println("Today is: " + today);
  }
}
```

OUTPUT

```
Today is: 2021-09-05T02:56:54.728564
```

Working with Date Parts

- **The `LocalDate` class has methods that allow you to get information about the date**

Example

```
import java.time.LocalDate;

LocalDate date = LocalDate.now();

System.out.println("Day of Week: " + date.getDayOfWeek());
System.out.println("Day of Month: " + date.getDayOfMonth());
System.out.println("Day of Year: " + date.getDayOfYear());
System.out.println("Month Name: " + date.getMonth());
System.out.println("Month Value: " + date.getMonthValue());
System.out.println("Year: " + date.getYear())
```

OUTPUT

```
Day of Week: WEDNESDAY
Day of Month: 15
Day of Year: 74
Month Name: MARCH
Month Value: 3
Year: 2023
```

Working with Date Parts

- **The `LocalTime` class has methods that allow you to get information about the time**

Example

```
Import java.time.LocalTime;

LocalTime time = LocalTime.now();

System.out.println("Hour: " + time.getHour());
System.out.println("Minute: " + time.getMinute());
System.out.println("Second: " + time.getSecond());
System.out.println("Nanosecond: " + time.getNano());
```

OUTPUT

```
Hour: 15
Minute: 7
Second: 16
Nanosecond: 14899100
```

Date/Time Formatting

- **The `DateTimeFormatter` class can be used to help you display dates/times in a format that you need**

- **The class is found in import `java.time.format`**

Example

```
import java.time.format.DateTimeFormatter;
```

- **You specify the desired format using the `DateTimeFormatter's ofPattern()` method**
  - Common formats include:

Value	Example
yyyy-MM-dd	"1988-09-29"
dd/MM/yyyy	"29/09/1988"
dd-MMM-yyyy	"29-Sep-1988"
E, MMM dd yyyy	"Thu, Sep 29 1988"

  - http://www.java2s.com/ref/java/java-datetimeformatter-patterns.html

- **A `LocalDateTime` object's `format()` method accepts a `DateTimeFormatter` object and returns the date object as a formatted string**

Example: Date/Time Formatting

Example

```
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class Program {

  public static void main(String[] args) {

    LocalDateTime today = LocalDateTime.now();
    System.out.println("Today is: " + today);

    // Specify the date/time format you will want to use
    DateTimeFormatter fmt =
      DateTimeFormatter.ofPattern("E, MMM dd, yyyy HH:mm:ss");

    String formattedDate = today.format(fmt);
    System.out.println("Today is: " + formattedDate);
 }
}
```

OUTPUT

```
Today is: 2021-09-05T03:02:10.846770
Today is: Sun, Sep 5, 2021 03:02:10
```

* NOTE: The actual time on the computer in this example was a little after 10pm CDT

Converting a String to a Date

- **Java also has the ability to convert a String into a Date**
  - `LocalDate.parse()`

Example

```
String userInput = "2002-10-17";

LocalDate birthDay = LocalDate.parse(userInput);
```

- **The `LocalDate.parse()` method requires the international date standard format**
  - This is the ISO 8601 standard
  - `YYYY-MM-DD`

- **However, it is possible to specify a different format when parsing a date by using the `DateTimeFormatter` class**
  - `DateTimeFormatter.ofPattern()`

Example

```
String userInput;
DateTimeFormatter formatter;

userInput = "10/17/2022";
formatter = DateTimeFormatter.ofPattern("MM/dd/yyyy")

LocalDate birthDay = LocalDate.parse(userInput, formatter);
```

Exercises

Remember… the code for each exercise below should be in its own subfolder under `Mod03`.

EXERCISE 2

Create a Java application named `FormatDates`. The application will get the current date and time and display that information in the following formats

```
09/05/2021
2021-09-05
September 5, 2021
Sunday, Sep 5, 2021  10:02     ← display in GMT time
```

CHALLENGE

```
5:02 on 05-Sep-2021            ← display in your local time zone
```

You may find the following helpful:

`https://beginnersbook.com/2013/05/java-date-timezone/`

EXERCISE 3

Create a Java application named SearchEngineLogger.

Create a method to log the actions of the user. Write each user action to a file name `logs.txt`.

Actions include:

a. Launching the application

b. Searching for a term

c. Exiting the application

In the `main()` method prompt the user for a search term that they wish to search.

Example

```
Enter a search term (X to exit):
```

Example

Entries in the `logs.txt` file should follow this format:

```
2023-09-06 12:42:20 launch
2023-09-06 12:42:45 search : How to use ChatGPT
2023-09-06 12:43:51 search : How to forge a camp knife
2023-09-06 12:45:32 exit
```

Section 2–3

CodeWars

CodeWars - Total Points

- **Total amount of points**
  - Calculate the number of points earned by a team in soccer matches
    - Win: 3 points
    - Tie: 1 point
    - Loss: 0 points
  - https://www.codewars.com/kata/5bb904724c47249b10000131/java

Module 3

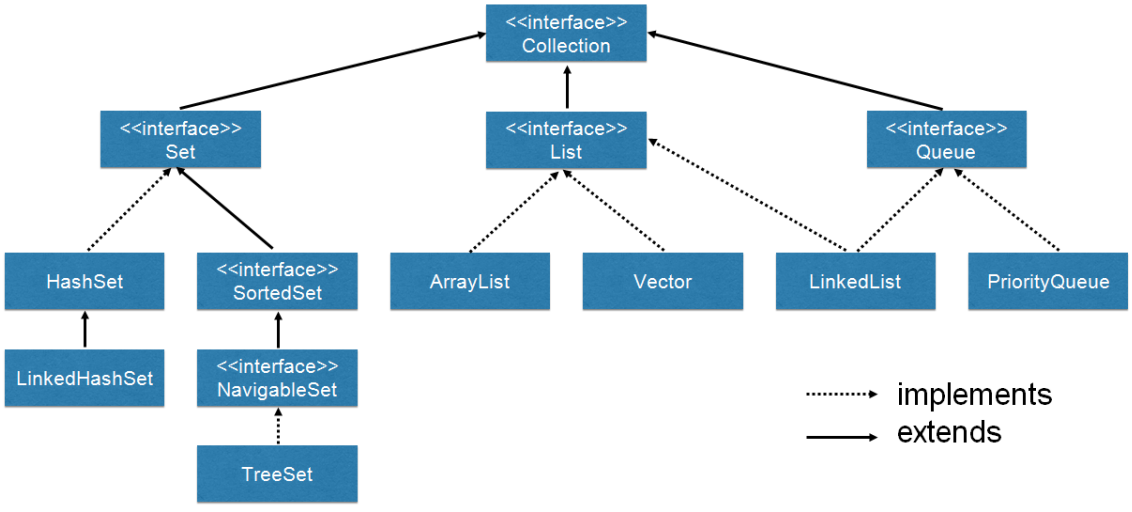
Collections - Lists

Section 3–1

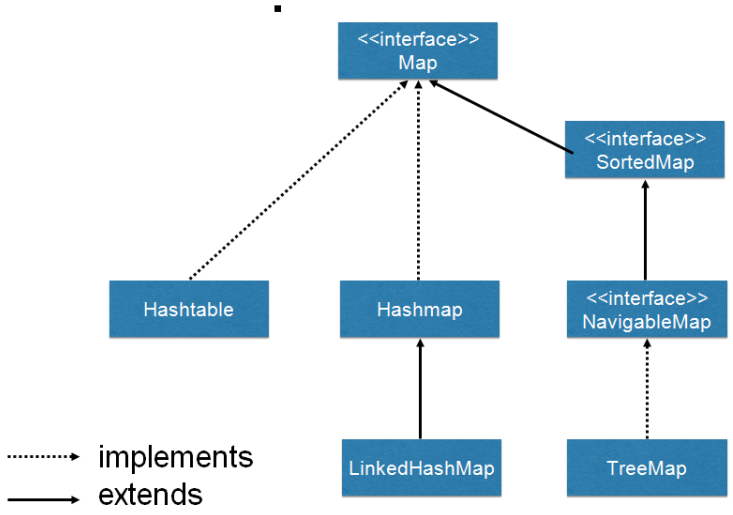
Collections

Collections

- **In Java, a collection is a single "container" object that contains zero or more individual objects**
  - Informally, you can think of an array as a collection
- **But Java formally introduced something called the *Collections Framework* in JDK 1.2**
  - Important places to find them include `java.util.Collection` and `java.util.Map`
- **The Collections Framework defines many interfaces and classes that help you manage collections of data**
  - This keeps you from having to build classes from the ground up
- **Collections include:**



- **Maps are also considered an important type of collection and include:**



Advantages of Using the Collection Framework

- **Good programmers could and have created their own collection classes**
  - But that takes time
  - They have to be tested
  - And they are usually only robust enough for the situation at hand
- **Java's Collection Framework classes implement as set of common interfaces**
  - This means that although each collection may manage their items differently, the way you interact with them feels familiar
- **Java's Collection Framework classes have been code to:**
  - provide fast access to data
  - efficiently manage the data
- **It is better to spend a few hours or days learning about the Collections Framework and have the tools available at your disposal than to spend similar amounts of time trying to code your own collections**

Section 3–2

Using an `ArrayList`

ArrayList

- **Java's `ArrayList` class manages a resizable array**
  - It maintains the order of the elements you add to the `ArrayList`
- **You will find it in the `java.util` package**

Example

```
import java.util.ArrayList;
```

- **But unlike a built-in array, the size of `ArrayList` isn't fixed**
  - It can grow or shrink over time as you add and remove elements
- **Because it is a generic class, you must specify the type of data that it manages**

Example

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

Adding Items to an `ArrayList`

- The `add()` method is used to add an element to the end of the collection

Example

```
import java.util.ArrayList;

public class ArrayListApp {

  public static void main(String[] args) {

    ArrayList<String> kids = new ArrayList<String>();

    kids.add("Natalie");
    kids.add("Brittany");
    kids.add("Zachary");

    System.out.println(kids);
  }
}
```

OUTPUT

```
["Natalie", "Brittany", "Zachary"]
```

Accessing an Item in an `ArrayList`

- The `get()` method is used to access an element by its position

- NOTE: To use the `get()` method in a loop, you can use the `size()` methods to return the number of elements in the

Example

```
import java.util.ArrayList;

public class ArrayListApp {

  public static void main(String[] args) {

    ArrayList<String> kids = new ArrayList<String>();

    kids.add("Natalie");
    kids.add("Brittany");
    kids.add("Zachary");

    for (int i = 0; i < kids.size(); i++) {
      System.out.println((i + 1) + " : " + kids.get(i));
    }
  }
}
```

OUTPUT

```
1 : Natalie
2 : Brittany
3 : Zachary
```

Updating an Item in an `ArrayList`

- **The `set()` method is used to update an element**
  - Specify its position and it's new value

Example

```
import java.util.ArrayList;

public class ArrayListApp {
  public static void main(String[] args) {

    ArrayList<String> kids = new ArrayList<String>();

    kids.add("Natalie");
    kids.add("Brittany");
    kids.add("Zachary");

    kids.set(2, "Zach");

    for (int i = 0; i < kids.size(); i++) {
      System.out.println((i + 1) + " : " + kids.get(i));
    }
  }
}
```

OUTPUT

```
1 : Natalie
2 : Brittany
3 : Zach
```

Removing an Item in an `ArrayList`

- **The `remove()` method is used to remove an element based on its position**
  - Once the item is removed, all elements below it in the list shift forward in the list and the size is reduced by one

Example

```
import java.util.ArrayList;

public class ArrayListApp {
  public static void main(String[] args) {

    ArrayList<String> kids = new ArrayList<String>();

    kids.add("Natalie");
    kids.add("Brittany");
    kids.add("Zachary");

    kids.remove(1);

    System.out.println("After Brittany removed: ");
    for (int i = 0; i < kids.size(); i++) {
      System.out.println((i + 1) + " : " + kids.get(i));
    }

    kids.add("Brittany");

    System.out.println("After Brittany re-added: ");
    for (int i = 0; i < kids.size(); i++) {
      System.out.println((i + 1) + " : " + kids.get(i));
    }
  }
}
```

OUTPUT

```
1 : Natalie
2 : Zach
3 : Brittany
```

Other `ArrayList` Methods

- **There are many other array list methods, including:**
  - `Collections.sort()` - sorting an `ArrayList`
    - It can sort lists alphabetically or numerically
  - `clear()` - clearing all items in an `ArrayList`

Example: Manage a Collection of Data using `ArrayList`

Example

`Product.java`

```
public class Product {
   private int id;
   private String name;
   private float price;

   public Product(int id, String name, float price) {
      this.id = id;
      this.name = name;
      this.price = price;
   }

   public int getId() {
      return this.id;
   }

   public String getName() {
      return this.name;
   }

   public float getPrice() {
      return this.price;
   }

}
```

Example

Store.java

```
import java.util.ArrayList;
import java.util.Scanner;

public class StoreApp {

  static void main(String[] args) {

      ArrayList<Product> inventory = getInventory();

      Scanner scanner = new Scanner(System.in);

      System.out.println("We carry the following inventory: ");

      for (int i = 0; i < inventory.size(); i++) {
         Product p = inventory.get(i);
         System.out.printf("id: %d %s - Price: $%.2f",
            p.getId(), p.getName(), p.getPrice());
      }
   }

   public ArrayList<Product> getInventory() {

      ArrayList<Product> inventory = new ArrayList<Product>();

      // this method loads product objects into inventory
      // and its details are not shown

      return inventory;
   }

}
```

Exercises

In the following exercise you will create an application to manage and search the product inventory of a store using java collections.

EXERCISE 1

Create a new Java application named SearchInventory. You will code the application we saw in the previous pages. The application displays the inventory that our store carries.

You will need to code the `getInventory()` method and create an initial inventory of at least 5 producs. An `ArrayList`'s size can change and will continue to grow as long as you have the energy to place products in the list.

Test the application.

BONUS: Replace the code that loaded the `ArrayList` with code that reads data from a file named `inventory.csv`. Create a file containing products that resembles the following.

```
4567|10' 2x4  (grade B)|9.99
1234|Hammer|19.49
2345|Box of nails|9.29
```

Read the file a line at a time. Split the string where you find the pipe (`|`) character and use the parts to create a `Product` object. Add the object to the `ArrayList`. The list will be able to accommodate however many products you add to the file.

BONUS: Sort the products by name before you display them.

Hint: `https://www.bezkoder.com/java-sort-arraylist-of-objects/`

BONUS: Replace the user interface of the program with a menu driven one. Provide a loop and prompt the user using a style resembling the following:

```
What do you want to do?
  1- List all products
  2- Lookup a product by its id
  3- Find all products within a price range
  4- Add a new product
  5- Quit the application
Enter command:
```

Commit and push your code!

Section 3–3

CodeWars

CodeWars - Roman Numerals

- **Roman Numerals Converter**
  - Convert an integer to a Roman Numeral String
  - https://www.codewars.com/kata/51b62bf6a9c58071c600001b/java

Module 4

Collections - Maps

Section 4–1

Using a `HashMap`

HashMap

- **A `HashMap` stores items in key/value pairs**
- **To access an element, you use the key (like a subscript)**
  - It returns the value associated with that key
- **You will find it in the `java.util` package**

Example

```
import java.util.HashMap;

HashMap<String, String> statesAndCapitals
                         = new HashMap<String, String>();
```

- **The `HashMap` class is a generic class**
  - They key can be of one data type
  - The value can be of another data type

Example

```
// The key in this example is a String (state) and
// the value is a String (capital)

HashMap<String, String> statesAndCapitals
                                = new HashMap<String, String>();
```

Adding Items to a `HashMap`

- **There can add an item to a `HashMap` using the `put()` method**
  - The keys in a `HashMap` must be unique

Example

```
import java.util.HashMap;

public class Main {
  public static void main(String[] args) {

    HashMap<String, String> statesAndCapitals
                              = new HashMap<String, String>();

    statesAndCapitals.put("CT", "Hartford");
    statesAndCapitals.put("CA", "Sacramento");
    statesAndCapitals.put("WA", "Olympia");
    statesAndCapitals.put("TX", "Austin");
    statesAndCapitals.put("FL", "Tallahassee");

    System.out.println(statesAndCapitals);
  }
}
```

OUTPUT

```
{CT=Hartford, TX=Austin, FL=Tallahassee, WA=Olympia,
CA=Sacramento}
```

- **Notice the values aren't necessarily displayed in the order they were added**
  - This is because the `HashMap` uses a "hash function" to convert the key into a position in an underlying collection for fast lookup
  - This display order is impacted by this hash function

Looking up Items in a `HashMap`

- **You can look up an item in a `HashMap` using the `get()` method**
  - Specify its key
- **If it finds the key in the `HashMap`, it returns the corresponding value**
  - If it doesn't find the key, it returns `null`

Example

```
import java.util.HashMap;

public class Main {
  public static void main(String[] args) {

    HashMap<String, String> statesAndCapitals
                              = new HashMap<String, String>();

    statesAndCapitals.put("CT", "Hartford");
    statesAndCapitals.put("CA", "Sacramento");
    statesAndCapitals.put("WA", "Olympia");
    statesAndCapitals.put("TX", "Austin");
    statesAndCapitals.put("FL", "Tallahassee");

    System.out.print("The capital of Texas is: ");
    System.out.println(statesAndCapitals.get("TX"));
  }
}
```

OUTPUT

```
The capital of Texas is: Austin
```

- **Failed lookups return `null`**

  Example

```
System.out.print("The capital of Oklahoma is: ");
System.out.println(statesAndCapitals.get("OK"));
```

OUTPUT

```
The capital of Oklahoma is: null
```

- **If you are concerned about failed lookups, you should check the returned value before trying to use it**

  Example

```
System.out.print("The capital of Oklahoma is: ");

String value = statesAndCapitals.get("OK");
if (value != null) {
   System.out.println(value);
}
else {
   System.out.println("OK is not in the states map");
};
```

OUTPUT

```
OK is not in the states map
```

Removing an Item from a `HashMap`

- **You can remove one item from a `HashMap` by passing the key to the `remove()` method**
  - If it finds the key in the `HashMap`, it does not fail
- **You can also remove all items in a `HashMap` by calling `clear()`**

Example

```
import java.util.HashMap;

public class Main {
  public static void main(String[] args) {

    HashMap<String, String> statesAndCapitals
                               = new HashMap<String, String>();

    statesAndCapitals.put("CT", "Hartford");
    statesAndCapitals.put("CA", "Sacramento");
    statesAndCapitals.put("WA", "Olympia");
    statesAndCapitals.put("TX", "Austin");
    statesAndCapitals.put("FL", "Tallahassee");

    statesAndCapitals.remove("WA");

    System.out.println(statesAndCapitals);
  }
}
```

OUTPUT

```
{CT=Hartford, TX=Austin, FL=Tallahassee, CA=Sacramento}
```

Iterating Through Items in a `HashMap`

- **You can iterate through the items in a `HashMap` using different techniques**
  - If you are only interested in the values, call the `values()` methods on the `HashMap` and iterate through the results

Example

```
HashMap<String, String> statesAndCapitals
                              = new HashMap<String, String>();

statesAndCapitals.put("CT", "Hartford");
statesAndCapitals.put("CA", "Sacramento");
statesAndCapitals.put("WA", "Olympia");
statesAndCapitals.put("TX", "Austin");
statesAndCapitals.put("FL", "Tallahassee");

for (String value : statesAndCapitals.values()) {
   System.out.println(value);
}
```

OUTPUT

```
Hartford
Austin
Tallahassee
Olympia
Sacramento
```

  - If you are only interested in the values, call the `keySet()` methods on the `HashMap` and iterate through the results

Example

```
for (String key : statesAndCapitals.keySet()) {
   System.out.println(key + ": " + statesAndCapitals.get(key));
}
```

OUTPUT

```
CT: Hartford
TX: Austin
FL: Tallahassee
WA: Olympia
CA: Sacramento
```

- **Although we've shown iterating through a `HashMap` in these pages, more often it is used to look values up on demand**

Example: Using a `HashMap` for Lookup

Example

`Product.java`

```
public class Product {
   private int id;
   private String name;
   private float price;

   public Product(int id, String name, float price) {
      this.id = id;
      this.name = name;
      this.price = price;
   }

   public int getId() {
      return this.id;
   }

   public String getName() {
      return this.name;
   }

   public float getPrice() {
      return this.price;
   }
}
```

Example

Store.java

```
import java.util.HashMap;
import java.util.Scanner;

public class StoreApp {

   // the key is the product id, the value is a product object
   static HashMap<int, Product> inventory =
                                    new HashMap<int, Product>();

   static void main(String[] args) {

      // this method loads product objects into inventory
      loadInventory();

      Scanner scanner = new Scanner(System.in);

      System.out.print("What item # are you interested in? ");
      int id = scanner.nextInt();

      Product matchedProduct = inventory.get(id);
      if (matchedProduct == null) {
         System.out.println("We don't carry that product");
         return;
      }

      System.out.printf("We carry %s and the price is $%.2f",
         matchedProduct.getName(), matchedProduct.getPrice());
   }
}
```

Exercises

In this exercise you will create a product inventory application similar to the previous one that you created. This time you will manage the inventory with a `Map` instead of with an `ArrayList`.

EXERCISE 1

Create a new Java application SearchInventoryMap. Write the code for the application that was demonstrated in the previous pages.

Add a `loadInventory()` method to load all of the products from the `Inventory.csv` file. Create a product from each line and add it to a Map. Use the product name as the map key so that users can search for products by name.

Test the application.

BONUS: Write code to let the user look up more than one product. After the program displays the results of the search, ask the user "Do you want to search again?". Keep repeating the search as long as they answer yes to the question.

Commit and push your code!

Section 4–2

CodeWars

CodeWars - Cat Years / Dog Years

- **Cat Years and Dog Years**
  - Calculate how old a pet is in…
    - Human years
    - Cat years
    - Dog Years
  - https://www.codewars.com/kata/5a6663e9fd56cb5ab800008b/java