Fidelity LEAP

Technology Immersion Program

Refresher Workshop: HTML/CSS, OO, TDD, and Java Basics

Introduction

Course Description

How is this course valuable to a Full Stack Engineer (FSE)?

- This is an overview, level-setting course
- The goal is for all associates to have an understanding of the concepts and terminology used throughout this LEAP training program
- Understanding the “big picture” is what this course offers
- Pay close attention to the terms, ideas, and best practices discussed, as other LEAP associates have had issues with some of these concepts. This course can help you avoid their challenges.
- All LEAP training is to help you be “desk ready, day one.” This course supports that goal.

Course Outline

Course Objectives

In this course, we will:

- Learn the basics of HTML (HyperText Markup Language) and CSS (Cascading Style Sheets)
- Understand Object-Oriented Programming (OOP or OO) concepts and explore UML (Unified Modeling Language) class diagrams to help problem solve
- Discuss Test-Driven Development (TDD) and the value of testing for a full-stack developer
- Explore Java basics before exploring Java in a large financial institution

Key Deliverables

Course Notes

There is no Skills Assessment for this course.

Fidelity LEAP

Technology Immersion Program

Refresher Workshop: HTML/CSS, OO, TDD, and Java Basics

Chapter 1: HTML and CSS Overview

Chapter Objectives

In this chapter, we will explore a high-level overview of standardizing a presentation with HTML including:

- HTML essentials
- Basic styling
- CSS

Chapter Concepts

HyperText Markup Language (HTML)

- HTML is the standard markup language for creating web pages
- HTML allows you to define the structure and content of your web page
- This course is high-level, and we will discuss concepts of building HTML and CSS web pages
- Additional detail will be discussed in later instructor-led courses

HTML Core Concepts: Elements

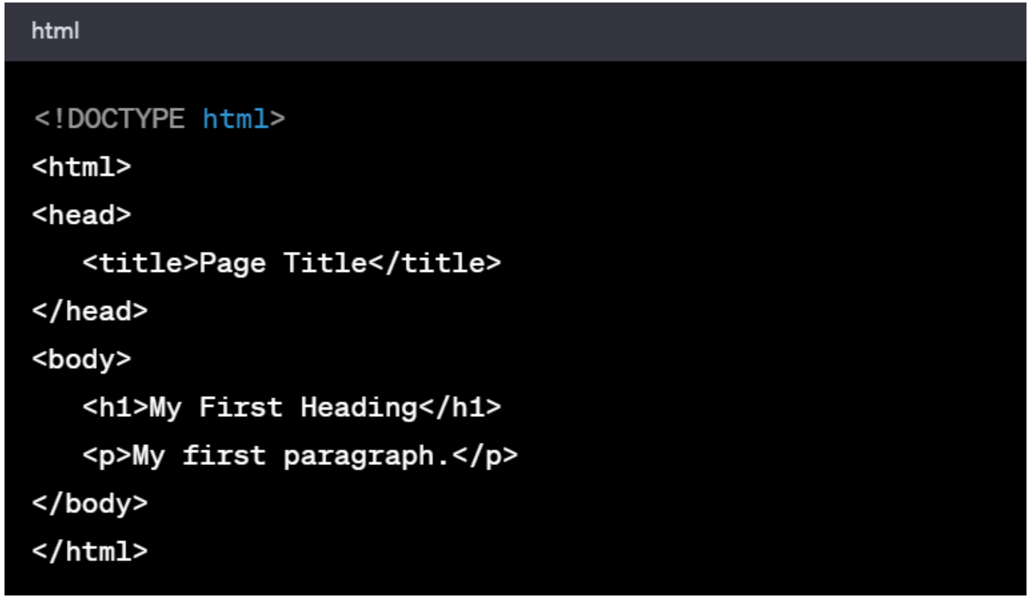
- Elements and tags: HTML is composed of elements, which are surrounded by tags
- For example, `<h1>Hello World</h1>`
  - `<h1>` is an opening tag
  - `</h1>` is the closing tag
  - `Hello World` is the content
- An `h1` tag makes content within it large and bold, and it includes an end of line
- Heading tags go from `h1` to `h6`, becoming progressively smaller

HTML Core Concepts: Attributes

- Attributes provide additional information about an element
- E.g., `<a href="https://www.example.com">Click here</a>`
- Here, `href` is an attribute that specifies the link’s URL

Simple HTML Page

- Simple text makes an HTML page



html

```html
<!DOCTYPE html>
<html>
<head>
   <title>Page Title</title>
</head>
<body>
   <h1>My First Heading</h1>
   <p>My first paragraph.</p>
</body>
</html>
```

HTML Common Elements

- Headings: `<h1>`, `<h2>`, … `<h6>`
- Paragraph: `<p>`
- Links: `<a>`
- Lists: `<ul>` (unordered), `<ol>` (ordered), and `<li>` (list item)
- Images: `<img>`
- Forms: `<form>`, `<input>`, `<textarea>`, `<button>`, etc.
- For more elements: https://textbooks.cs.ksu.edu/cis526/a-html/07-common-elements/

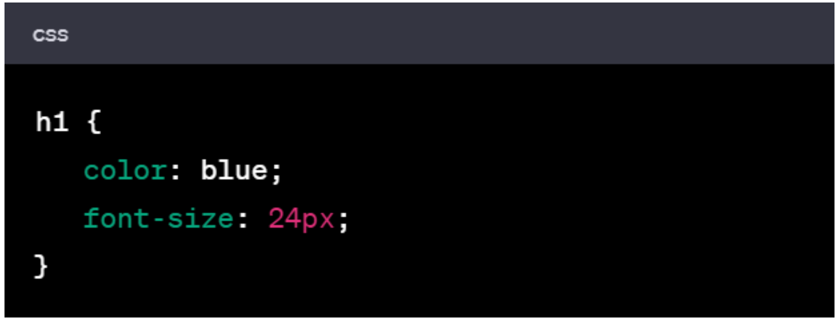
Semantic Elements

- Semantic elements are **required**, and more will be discussed in later courses
- Be sure to use semantic elements
- These are newer elements introduced to make the web more descriptive and accessible
  - Examples include `<main>`, `<header>`, `<footer>`, `<article>`, `<section>`, `<nav>`, and `<aside>`
  - Another example would be:
    `<footer>copyright Fidelity Investments &copy;</footer>`
- Be sure to include semantic tags in your solutions

Chapter Concepts

Cascading Style Sheets (CSS)

- CSS describes how HTML elements should be displayed
  - It deals with the look and feel of a web page
- **Selectors and Properties**: CSS rules are defined by selecting an HTML element and applying properties to it



```
css
```

```css
h1 {
    color: blue;
    font-size: 24px;
}
```

- Here, **`h1`** is the selector, and **`color`** and **`font-size`** are properties

CSS Classes and IDs

- **Class**: can be used on multiple elements
  - E.g., `.myClass { color: red; }` and in HTML as `<p class="myClass">...`
- **ID**: unique and used on a single element
  - E.g., `#myID { font-weight: bold; }` and in HTML as `<div id="myID">...`
- For more detail: https://www.w3schools.com/css/css_selectors.asp
- Detailed discussions of CSS are included in upcoming courses

Optional Instructor Demo

DEMO

- Optionally, if there is time, your instructor will show a short demo of how to make a web page using Visual Studio Code
- Later courses will have many hands-on exercises
  - This offering explores what will be covered in great detail in the future

Exercise 1.1: Applying CSS Styling

HANDS-ON EXERCISE

30 min

- Using Visual Studio Code, make a web page showing your favorite festival
- Name that page `index.html`
- As an example, Diwali or the American 4th of July, but even better, choose a local festival that is important to you
- Work in teams
- Everyone at your table, or breakout room, is on your team
- Make sure everyone creates a simple web page about their favorite festival
- Be prepared to share your work with your classmates

Chapter Concepts

Chapter Summary

In this chapter, we have explored a high-level overview of standardizing a presentation with HTML including:

- HTML essentials
- Basic styling
- CSS

Fidelity LEAP

Technology Immersion Program

Refresher Workshop: HTML/CSS, OO, TDD, and Java Basics

Chapter 2: Object-Oriented Programming (OOP) Overview

Chapter Objectives

In this chapter, we will explore:

- A high-level overview of Object-Oriented Programming (OOP)
  - What is OOP?
  - Unified Modeling Language (UML) class diagrams

Chapter Concepts

Object-Oriented Programming (OOP) or (OO)

- Object-Oriented Programming (OOP) is a programming paradigm based on the concept of "objects"
- These objects can contain data in the form of fields (often known as attributes or properties), and code, in the form of procedures (often known as methods)
- It's a way to structure and design your code, making it more modular, reusable, and often easier to understand
- https://www.geeksforgeeks.org/introduction-of-object-oriented-programming/

Classes and Start of Objects

- **Classes**: a blueprint or template for creating objects
- For instance, if you have a class `Dog`, then you can have objects like `GoldenRetriever`, `Bulldog`, etc., which are instances of the `Dog` class
- Example: `Account`, `SavingAccount`, `CheckingAccount` as classes
- `MyAccount`, `mikesAccount`, `charlesAccount`, `bradsAccount` each are different and separate in memory
  - They are objects
- Classes are blueprints to build "buildings", or objects

Core Concepts of OOP: Objects

- **Objects**: the primary building blocks of OOP
- An object is a self-contained unit that represents a real-world entity, combining data (attributes) and the operations (methods) that can use or modify that data
- Example: Account could be an object with **accountNumber**, **ownerName**, **balanceOfAccount**, and ability to **deposit** an amount of money, **withdraw**, and get current balance
- As an object, my account is completely separate from your account
- An `Account` class might be used as a plan or blueprint, but each account (object) is separate in memory

Objects

- Objects are an instance of the class
- Objects have data and the ability to manipulate that data
- The same class is used to make an object
  - For example, my retirement account is separate from your account

Inheritance

- **Inheritance**: a mechanism where a new class can inherit properties and methods from an existing class
- This promotes code reuse and establishes a natural hierarchy between classes
- For instance, a `GoldenRetriever` class might inherit from the general `Dog` class
- Example `SavingAccount` and `CheckingAccount` might inherit from a base class of `Account`

Polymorphism

- **Polymorphism**: it allows one interface to be used for a general class of actions
- The specific action is determined by the exact nature of the situation
- For instance, you might have a method `makeSound()` for a `Dog` class
- The exact sound might differ if the dog is a `GoldenRetriever` or a `Bulldog`, but you can still use `makeSound()` for any dog

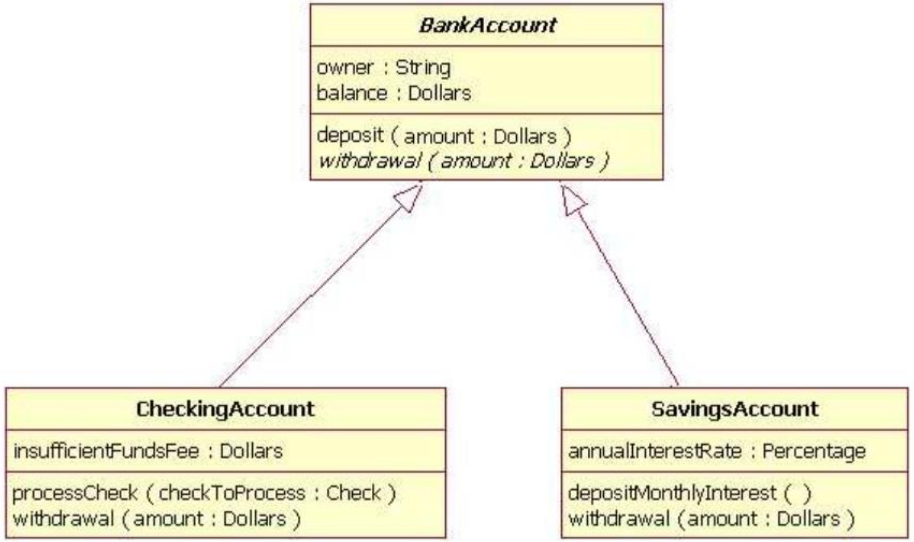
Encapsulation

- **Encapsulation**: bundling the data (attributes) and the methods that operate on the data into a single unit or class, and restricting the direct access to some of the object's components
- This is a fundamental principle that enforces data hiding and modularity
- Generally, make member variables **private**
- Much more on this topic in later courses

Chapter Concepts

UML (Unified Modeling Language) Class Diagrams

- Unified Modeling Language (UML) is a standardized way to visualize the design of a system
- One of the most common UML diagrams is the class diagram, which shows the static structure of a system, illustrating its classes, attributes, methods, and relationships among objects
- Creating even a simple class diagram before coding can be the key to success
- https://developer.ibm.com/articles/the-class-diagram/


BankAccount
owner : String
balance : Dollars
deposit (amount : Dollars)
withdrawal (amount : Dollars)
CheckingAccount
insufficientFundsFee : Dollars
processCheck (checkToProcess : Check)
withdrawal (amount : Dollars)
SavingsAccount
annualInterestRate : Percentage
depositMonthlyInterest ( )
withdrawal (amount : Dollars)

Model Before Coding

- “Think first, code second”
- https://www.codecademy.com/resources/blog/how-to-think-like-a-programmer/
- Writing out simple UML class diagrams before coding to understand the problem is common in OOP

Optional Instructor Demo

DEMO

- If time permits, your instructor will make a simple `BankAccount` class in the Java programming language, as seen on a previous slide
- Associates will make and test classes in later courses

Optional Exercise 2.1: Simple Java Class

HANDS-ON EXERCISE

30 min

- Using Eclipse, create a new Java project
- Create a `BankAccount` class similar to the UML shown in earlier slides
- In later courses, you will create Java classes, so it is not critical to make them today

Chapter Concepts

Chapter Summary

In this chapter, we have explored:

- A high-level overview of Object-Oriented Programming (OOP)
  - What is OOP?
  - Unified Modeling Language (UML) class diagrams

Fidelity LEAP

Technology Immersion Program

Refresher Workshop: HTML/CSS, OO, TDD, and Java Basics

Chapter 3: Test-Driven Development (TDD) Overview

Chapter Objectives

In this chapter, we will explore:

- A high-level overview of Test-Driven Development (TDD)
  - What is TDD?
  - TDD in a large financial firm

Chapter Concepts

What Is Test-Driven Development (TDD)?

- **Test-Driven Development (TDD)** is a software development approach where you write automated tests before writing the actual code (functionality)
- It relies on the repetition of a very short development cycle

Write the Test First

- You first write a failing test case that defines a desired improvement or new function
- Then produce code to pass that test
- Finally, refactor or improve the code to acceptable standards
- https://www.guru99.com/test-driven-development.html
- https://www.youtube.com/watch?v=GvAzrC6-spQ&ab_channel=CraftSoftwarewithVictorRentea

Benefits of TDD in a Financial Firm

- **Early bug detection**: discovering issues early in the development cycle is less costly than finding them later, especially in a financial context
- **Documentation**: tests act as documentation
  - New team members can understand how a system is supposed to function simply by looking at the tests
- **Confidence**: when changes are made to a system (such as regulatory updates), having a suite of tests can provide confidence that nothing has been inadvertently broken
- **Improved design**: TDD often results in a modular and flexible codebase, making it easier to adapt to changing requirements in the dynamic world of finance

Chapter Concepts

Why Is TDD Important for a Financial Firm?

- **Accuracy is paramount**: even minor errors can result in significant financial losses in the financial domain
- TDD can catch these errors early
- Testing units of work is likely to keep bugs from being introduced into a code base
- It's very hard to fix bugs in production code
  - Better to address before

Testing Is Important

- **Regulatory compliance**: many financial transactions must adhere to regulatory standards
  - Tests ensure that the software meets these requirements consistently
- **Complex systems**: financial systems can be intricate, with numerous interconnected components
  - TDD can help in ensuring that a change in one part doesn't break another
- "Don't Break the System"
  - https://rachelcarmena.github.io/2019/04/13/refactoring.html

Write a Failing Test

- **Write a failing test**: before writing any functional code, write a test that checks for the expected outcome of the feature
- This test should fail because the feature hasn't been implemented yet

Write the Minimum Code to Pass the Test

- **Write the code**: write the minimum amount of code required to make the test pass
- **Run the test**: check if your code passes the test
  - If it doesn't, modify the code (not the test) and run the test again
- **Refactor**: once the test passes, you can refactor your code to improve its structure, efficiency, or readability without changing its external behavior
  - This step is crucial to keep the codebase clean and maintainable
- **Repeat**: continue this process about every two minutes for each new feature or functionality
  - Red, Green, Refactor
- https://javaniceday.com/what-is-red-green-refactor-in-software-development/

Optional Instructor Demo: TDD

DEMO

- If time permits, your instructor will demonstrate the Red, Green, Refactor pattern of TDD
- While more testing might be done later, testing can help a developer write better production code
- TDD will be explored in later courses throughout the program

Chapter Concepts

Chapter Summary

In this chapter, we have explored:

- A high-level overview of Test-Driven Development (TDD)
  - What is TDD?
  - TDD in a large financial firm

Fidelity LEAP
Technology Immersion Program

Refresher Workshop: HTML/CSS, OO, TDD, and Java Basics

Chapter 4: Java Programming Overview

Chapter Overview

In this chapter, we will explore:

- A high-level overview of Java
  - Java overview
  - Java classes

Chapter Concepts

Overview of Java

- Java is one of the predominant languages used in large financial organizations, owing to its robustness, platform independence, and enterprise-friendly features
- Java is a high-level, object-oriented programming language
  – It was developed by Sun Microsystems in the 1990s and has since become one of the most popular programming languages in the world
- Java is now owned and maintained by Oracle
- https://www.ibm.com/topics/java
- https://docs.oracle.com/en/database/oracle/oracle-database/21/jjdev/Java-overview.html

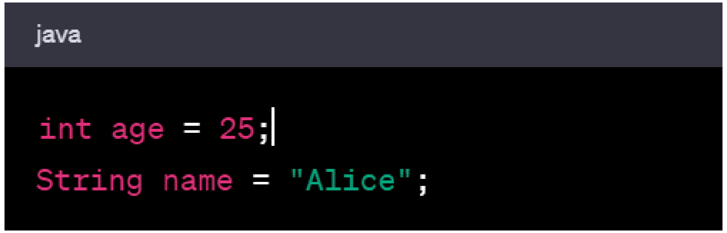
Why Java?

- Java programming in a large financial organization is about leveraging the language’s strengths and its ecosystem’s strengths to build robust, secure, and efficient applications tailored to the intricate and high-stakes world of finance
- Java is stable, proven, and common in financial institutions

Basic Java Programming Elements

- **Variables**: store data for processing
  - Java is statically-typed, which means you must declare the type of a variable before using it
- Java requires a datatype before a variable is declared
- Member variables of a class should have a permission level, usually private

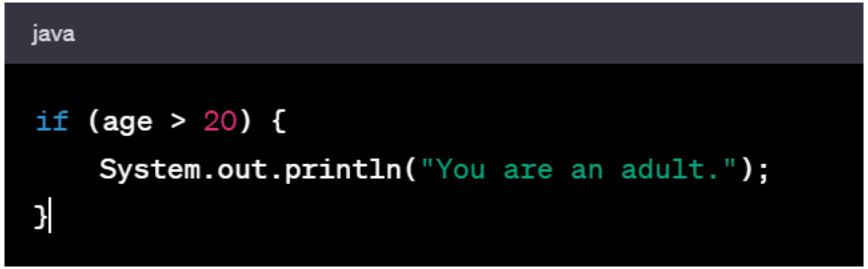
```
private int age = 25;
private String fullName = "Charles Smartman";
```



```java
int age = 25;
String name = "Alice";
```

Java Control Structures

- Java uses C programming control structures for `if` statements and loops



java

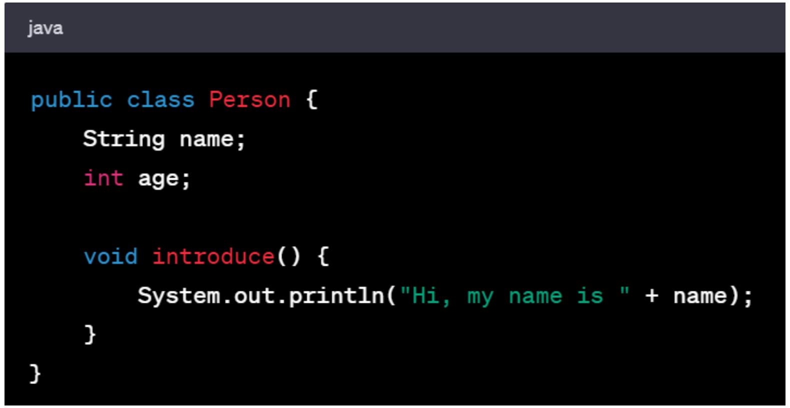
```java
if (age > 20) {
    System.out.println("You are an adult.");
}
```

Chapter Concepts

Java Classes

- **Classes**: blueprints for objects
  - A class bundles methods and attributes (variables) that operate on the data
- While technically legal, member variables should be private and methods public
  - More on this in future courses

```
public class Person{

   private String name;

   private int age;

   public void introduce(){

         System.out.println("This is better");

   }

}
```



java

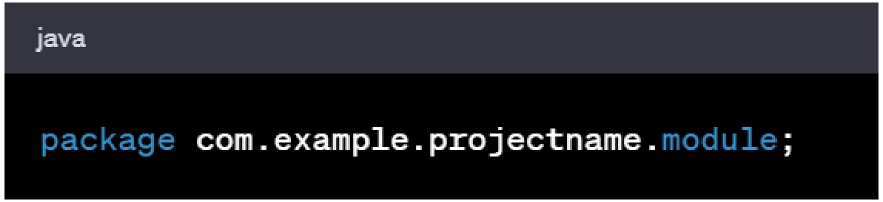
```java
public class Person {
    String name;
    int age;

    void introduce() {
        System.out.println("Hi, my name is " + name);
    }
}
```

Java Naming Conventions

- Classes start in uppercase
- Variables start in lowercase
- New words are in uppercase; e.g., `thisIsCommonInJava`
- Many Java frameworks and other developers rely on adherence to the Java naming conventions
- https://www.oracle.com/java/technologies/javase/codeconventions-namingconventions.html

Java Packages

- Java packages are an organizational system for code that resembles folders
- Use lowercase letters
- Use domain names in reverse as a prefix to ensure uniqueness

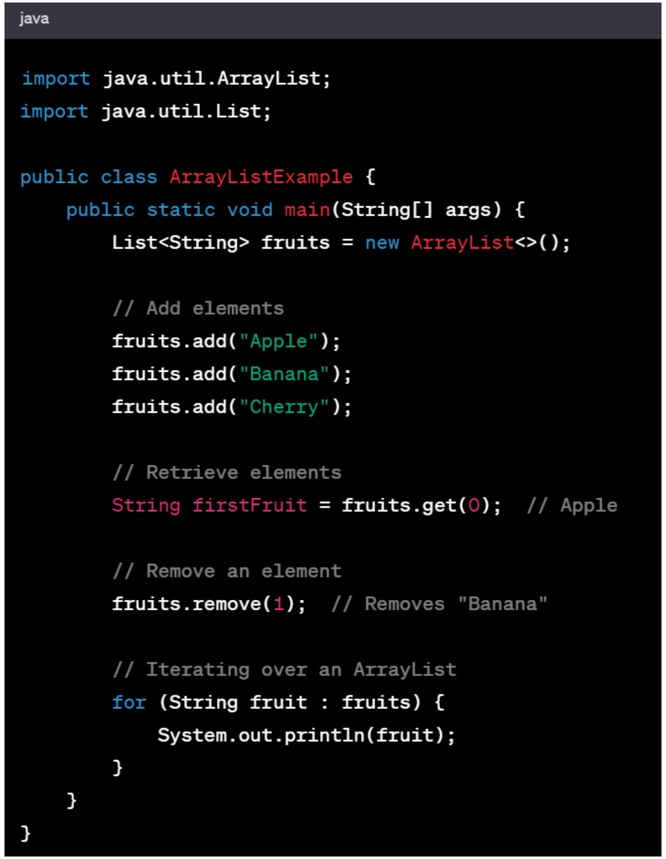


java

```java
package com.example.projectname.module;
```

List and ArrayList

- Discussed in much more detail in later courses, collections of objects can be kept in a list
- Here is a short example exposing you to Java collections
- As expected, `add` is used to add objects to a collection
- Using its index `remove`, removes items from a collection
- `public static void main(String[] args)` is how a command line application can be started
- `for` is a looping structure


java

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

public class ArrayListExample {
    public static void main(String[] args) {
        List<String> fruits = new ArrayList<>();

        // Add elements
        fruits.add("Apple");
        fruits.add("Banana");
        fruits.add("Cherry");

        // Retrieve elements
        String firstFruit = fruits.get(0);   // Apple

        // Remove an element
        fruits.remove(1);  // Removes "Banana"

        // Iterating over an ArrayList
        for (String fruit : fruits) {
            System.out.println(fruit);
        }
    }
}
```

Optional Instructor Demo: Java Programming

DEMO

- If time permits, your instructor will demonstrate a simple Java program
- The Java code on the previous slides is a good starting place
- If time permits, associates can create and run a similar Java program

Chapter Concepts

Chapter Summary

In this chapter, we have explored:

- A high-level overview of Java
  - Java overview
  - Java classes

Fidelity LEAP

Technology Immersion Program

Refresher Workshop: HTML/CSS, OO, TDD, and Java Basics

Course Summary

Course Summary

In this course, we have:

- Learned the basics of HTML (HyperText Markup Language) and CSS (Cascading Style Sheets)
- Understood Object-Oriented Programming (OOP or OO) concepts and explored UML (Unified Modeling Language) class diagrams to help problem solve
- Discussed Test-Driven Development (TDD) and the value of testing for a full-stack developer
- Explored Java basics before exploring Java in a large financial institution