multiverse

Contents

Java **2**

Learning Plan Tasks . . . 2
#checkoutTheDocs . . . 2
1. Getting Started with Java . . . 2
    Tasks . . . 3
2. Object-Oriented Programming in Java . . . 4
    Tasks . . . 4
3. Advanced Java Concepts . . . 6
    Tasks . . . 6
4. Input/Output In Java . . . 7
    Tasks . . . 7
5. Java Best Practices . . . 8
    Tasks . . . 9
Build Something Using Java! . . . 10
Next Steps . . . 10

Name:
Date:

Java

Java is a popular programming language that was first introduced by Sun Microsystems in 1995. It was designed to be portable, secure, and easy to use, and it quickly gained popularity among developers. Today, Java is used to build a wide variety of applications, from desktop and mobile apps to large-scale enterprise systems.

This learning plan is designed to introduce you to the basics of Java programming and help you build a solid foundation in the language. Over the course of five days, you will learn about topics such as basic syntax, object-oriented programming, exception handling, file I/O, and more. By the end of the learning plan, you should have a good understanding of the key concepts and tools used in Java programming.

Java has evolved significantly since its inception, with new versions and updates being released regularly. The language is maintained by Oracle Corporation, which acquired Sun Microsystems in 2010. Today, Java is widely used in the enterprise space and is also a popular choice for building Android mobile apps.

Learning Java can be a great way to expand your programming skills and open up new career opportunities. We hope that this learning plan will provide you with a solid foundation in Java programming and help you get started on your journey to becoming a skilled Java developer.

Learning Plan Tasks

1. Java Basics
2. Objects Oriented Programming
3. Advanced Java Concepts
4. Input and Output in Java
5. Java Best Practices
6. Build Something Using Java!

#checkoutTheDocs

- **Oracle**: The Java Tutorials
- **Oracle**: Installing Java
- **Udamy**: Java Tutorial
- **Baeldung**: Java Tutorial
- **CodeAcademy**: Java Courses
- **Java Code Geeks**

1. Getting Started with Java

This section covers the following topics:

- **Introduction to Java**: Understanding what Java is, why it's popular, and its benefits.
- **Installing Java**: Installing the latest version of Java SE Development Kit (JDK) and an Integrated Development Environment (IDE), such as Eclipse or IntelliJ IDEA.
- **Writing Your First Java Program**: Creating a "Hello, World!" program in Java using your IDE.

- **Basic Java Syntax**: Understanding the syntax and structure of Java code, including variables, data types, operators, and control structures.

By completing these tasks you should have a good understanding of what Java is, how to install and configure the Java Development Kit and an IDE, and how to write basic Java programs using variables, data types, operators, and control structures.

Tasks

1. Introduction to Java
   - **Research and Read About Java**: Understand what Java is, why it's popular, and what its benefits are. You can start by reading about Java on websites like:
     - Oracle's Java Website
     - Java Tutorial for Beginners
     - Java Documentation.
   - **Watch Online Tutorials**: You can also find many online video tutorials on Java that can help you understand the basics of Java programming.
2. Installing Java
   - **Install Java SE Development Kit (JDK)**: You can download the latest version of JDK from the official Oracle website. Follow the instructions to install the JDK on your computer.
   - **Install An IDE**: An IDE is a software application that provides a comprehensive environment for Java programming. Popular Java IDEs include Eclipse, IntelliJ IDEA, and NetBeans. Download and install an IDE of your choice on your computer.
   - **Configure Your IDE**: Once you have installed your IDE, you will need to configure it to use the JDK that you installed earlier. Follow the instructions provided by your IDE to configure it.
3. Writing Your First Java Program
   - **Create a New Java Project In Your IDE**: Open your IDE and create a new Java project.
   - **Create a New Java Class**: Inside your project, create a new Java class and name it `Greeter`.
   - In the `Greeter` class, write the code for a "Hello, World!" program.
   - Save your program and run it. You should see the message "Hello, World!" printed to the console.
4. Basic Java Syntax
   - **Variables and Data Types**: Understand what variables and data types are in Java, how to declare and use them, and how to convert between data types with casting.
   - **Operators**: Understand what operators are in Java, and how to use them to perform operations on variables and values.
   - **Control Structures**: Understand what control structures are in Java, and how to use them to control the flow of your code. Learn about if-else statements, loops, and switch statements.

5. Build Something
    - **Rock, Paper, Scissors**: You now know enough to rebuild your Rock, Paper, Scissors project! Your project should include the following:
        - Randomly selects rock, paper, or scissors for each player in the game. You should explore how to use `Math.random()` and `Math.round()` to achieve this.
        - Determines the winner of the game based on the random values that were selected.
        - **BONUS**: Explore how to accept user input with a `Scanner class` and use this to play the computer.

2. Object-Oriented Programming in Java

This section covers the following topics:

- **Understanding Object-Oriented Programming**: Understanding the basic concepts of OOP, such as classes, objects, inheritance, and polymorphism.
- **Creating Classes and Objects**: Creating custom classes and instantiating objects from them.
- **Encapsulation and Access Modifiers**: Understanding encapsulation and how to use access modifiers to control access to class members.
- **Inheritance and Polymorphism**: Understanding inheritance and how to use it to create subclasses and implement polymorphism.

By completing these tasks you should have a good understanding of Object-Oriented Programming concepts, how to create custom classes and instantiate objects from them, how to use access modifiers to control access to class members, and how to use inheritance and polymorphism to create subclasses and implement polymorphism.

Tasks

1. Understanding Object-Oriented Programming
    - **Research and Read About Object-Oriented Programming**: You can start by reading about Object-Oriented Programming (OOP) concepts and be able to explain the following concepts:
        - Classes
        - Objects
        - Inheritance
        - Polymorphism
        - Explain the benefits of OOP and how it differs from other programming paradigms.
    - **Watch Online Tutorials**: You can also find many online video tutorials on OOP that can help you understand the basic concepts of OOP.
2. Creating classes and objects
    - **Create a New Class**: In your IDE, create a new class and give it a name.
    - **Define Class Members**: Inside your class, define fields and methods that represent the properties and behaviors of the class.

- **Instantiate Objects**: Create new instances of your class by instantiating objects from it.

3. Encapsulation and Access Modifiers
   - **Understand Encapsulation**: Learn what encapsulation is in Java and why it is important. Encapsulation refers to the practice of hiding the implementation details of a class from its users and providing a well-defined interface for interacting with it.
   - **Use Access Modifiers**: Use access modifiers such as public, private, and protected to control access to class members. Public members can be accessed by any code, while private members can only be accessed from within the class.
4. Inheritance and Polymorphism
   - **Understand Inheritance**: Learn what inheritance is in Java and how it allows you to create new classes that are based on existing classes. Inheritance allows you to reuse code and create a hierarchy of classes.
   - **Create Subclasses**: Create new classes that inherit from existing classes. Subclasses can add new fields and methods, or override existing ones.
   - **Implement Polymorphism**: Understand polymorphism in Java and how it allows you to use objects of different classes in the same way. Polymorphism can be achieved through method overriding or through the use of interfaces.
5. Build Something!
   - **Option #1 Scooter Application**: Recreate the Scooter Application from Bootcamp using Java.
   - **Option #2: Employee Records System**: Create a Java program that models an employee management system for a company. The systems should have the follwoing components:
     - An `Employee` class that contains properties like employee ID, name, age, job title, and salary.
     - A `Manager` class that extends the `Employee` class. The `Manager` class should have additional properties like department and a method to set the department.
     - A `SalesPerson` class that extends the `Employee` class. The `SalesPerson` class should have an additional properties like salesTarget and a method to set the salesTarget.
     - **BONUS**: An `EmployeeDatabase` that can store a collection of `Employee` objects. The `EmployeeDatabase` class should have methods like add new employees, remove employees, and retrieve employee records.
     - Create a user interface that have options to add managers and salespeople and set their respective properties. If you did the bonus task, give users an interface that allows them to interact with the `EmployeeDatabase` by adding, removing, and displaying employee records.

3. Advanced Java Concepts

This section covers the following topics:

- **Exception Handling**: Understanding exceptions and how to handle them in Java.
- **Generics**: Understanding the benefits of Generics and how to use them in Java.
- **Collections**: Understanding the collections framework in Java and how to use collections, such as `ArrayList` and `HashMap`.
- **Multithreading**: Understanding the basics of multithreading and how to use threads in Java.

By completing these tasks you should have a good understanding of how to handle exceptions in Java, how to use Generics to write type-safe code, how to use the Collections framework to work with collections of objects, and how to use Multithreading to execute tasks concurrently.

Tasks

1. Understanding Exceptions and How to Handle Them in Java
    - **Learn About Exceptions**: Learn what Exceptions are in Java and how they can be used to handle runtime errors in your programs.
    - **Understand `try-catch` Blocks**: Learn how to use `try-catch` blocks to catch and handle Exceptions in your Java code.
    - **Learn About Different Types of Exceptions**: There are many different types of Exceptions in Java, such as Checked Exceptions, Unchecked Exceptions, and Errors. Learn about each type and when to use them.
2. Generics
    - **Learn about Generics**: Learn what Generics are in Java and how they can be used to write type-safe code that works with multiple types.
    - **Understand Type Parameters**: Learn about type parameters and how they are used in Generics to create type-safe collections and methods.
    - **Use Generics in Your Code**: Practice using Generics in your Java code to create collections and methods that work with different types.
3. Collections
    - **Learn About the Collections Framework**: Learn what the Collections framework is in Java and how it provides a set of interfaces and classes for working with collections of objects.
    - **Understand Different Types of Collections**: There are many different types of collections in Java, such as `ArrayList`, `LinkedList`, `HashSet`, and `HashMap`. Learn about each type and when to use them.
    - **Use Collections In Your Code**: Practice using collections in your Java code to store and manipulate sets of objects.
4. Multithreading
    - **Learn About Multithreading**: Learn what Multithreading is in Java and how it allows you to write programs that can execute multiple threads of execution concurrently.

- **Understand Thread Creation**: Learn how to create threads in Java and how to manage their execution.
- **Practice Multithreading**: Practice writing simple Java programs that use Multithreading to execute tasks concurrently.

5. Build Something
   - **Option #1 - Chat Application**: Create a simple chat application that does the following:
     - Allows users to send messages to one another in real time while handling exceptions that may happen.
     - The application should use Generics to allow users to send and receive messages of any type and `ArrayList` and `HashMap` to store and manage the messages and users.
     - The program should use multithreading to allow users to send and receive messages simultaneously.
     - The application should have a graphical user interface (GUI) that allows users to easily send and receive messages.
   - **Option #2 - Multiplayer Game**: Create a multiplayer game where players can play against each other on the same computer. The game should implement the following features:
     - The game should handle any exceptions that may occur during gameplay, such as input/output errors, and other errors that may occur while running the game.
     - Use generics in your code to ensure type safety and to make your code more efficient.
     - Use the collections framework in Java, such as `ArrayList` and `HashMap`, to store and manipulate game data, such as player scores and game states.
     - Use threads in your code to allow for multiple players to play the game simultaneously.

4. Input/Output In Java

This section covers the following topics:

- **Reading and Writing Files**: Understanding how to read and write files in Java, including text and binary files.
- **Standard Input/Output**: Understanding standard input/output in Java and how to use it to interact with the console.
- **Networking**: Understanding the basics of networking in Java and how to create client-server applications.

By completing these tasks you should have a good understanding of how to read and write files in Java, how to use standard input/output to interact with the console, and how to create client-server applications using networking.

Tasks

1. Reading and Writing Files

- **Learn about File I/O**: Learn what File I/O is in Java and how it allows you to read and write files.
- **Understand Streams**: Learn about Input and Output Streams and how they are used for reading and writing data to files.
- **Practice reading and writing files**: Practice writing Java code to read and write files, including text and binary files.

2. Standard Input/Output

- **Learn About Standard Input/Output**: Learn what Standard Input/Output is in Java and how it allows you to interact with the console.
- **Understand `System.in` and `System.out`**: Learn how to use `System.in` and `System.out` to read input from the console and write output to the console.
- **Practice Standard I/O**: Practice writing Java code to read input from the console and write output to the console.

3. Networking

- **Learn About Networking**: Learn what Networking is in Java and how it allows you to create client-server applications.
- **Understand Sockets**: Learn how to use Sockets to create network connections between client and server applications.
- **Practice Networking**: Practice writing Java code to create a client-server application using Sockets.

4. Build Something

- **Option #1 - Chat Application**: Enhance and/or build the chat application from the last section. Build a simple client-server chat application. The application should allow multiple clients to connect to a server and communicate with each other through the server. The server should save the chat history to a text file and load the chat history from the text file when it starts up.
- **Option #2 - Multiplayer Game**: Enhance and/or build a multiplayer game from the previous section. The game should allow multiple players to connect to a server and play a game together in real-time. The server should save player data, including scores and inventory, to a file and should load player data from the file when it starts up.

5. Java Best Practices

This section covers the following topics:

- **Code Organization and Style**: Understanding how to organize and format your Java code to make it easy to read and maintain.
- **Testing**: Understanding the importance of testing your Java code and how to use testing frameworks, such as JUnit.
- **Debugging**: Understanding how to debug Java code using an IDE and other debugging tools.
- **Java Programming Tips and Tricks**: Understanding some useful tips and tricks to improve your Java programming skills and efficiency.

Name:
Date:

By completing these tasks you should have a good understanding of how to organize and style your Java code, write and run tests using JUnit, debug Java code using an IDE and other tools, and use tips and tricks to improve your Java programming skills and efficiency. Good luck!

Tasks

1. Code Organization and Style
    - **Learn About Code Organization and Style**: Learn the best practices for organizing and formatting your Java code, such as using packages and following naming conventions.
    - **Understand Code Design Patterns**: Learn about common code design patterns and how they can be used to organize and structure your code.
    - **Practice Code Organization and Style**: Refactor some of your existing Java code to adhere to best practices for code organization and style.
2. Testing
    - **Learn About Testing**: Learn the importance of testing your Java code and the different types of tests, such as unit tests and integration tests.
    - **Understand JUnit**: Learn how to use JUnit, a popular testing framework for Java, to write and run unit tests for your code.
    - **Practice Testing**: Write some JUnit tests for your existing Java code and/or projects to ensure it is working as intended.
3. Debugging
    - **Learn About Debugging**: Learn the basics of debugging Java code using an IDE and other debugging tools, such as logging.
    - **Understand Common Errors**: Learn about common errors in Java code, such as `NullPointerExceptions` and `OutOfMemoryErrors`, and how to troubleshoot them.
    - **Practice Debugging**: Use an IDE and other debugging tools to debug some of your existing Java code and fix any errors you encounter.
4. Java Programming Tips and Tricks
    - **Learn about Java Programming Tips and Tricks**: Learn some useful tips and tricks to improve your Java programming skills and efficiency, such as keyboard shortcuts and code snippets.
    - **Understand Code Optimization**: Learn about common code optimization techniques, such as using StringBuilder instead of String concatenation, to improve the performance of your code.
    - **Practice Java Programming Tips and Tricks**: Apply some of the tips and tricks you learned to optimize and improve the efficiency of your existing Java code.
5. Build Something
    - Update one of the projects you have built so far so that it:
        - Follows code organization and style principles you researched.
        - Incorporates unit testing using JUnit

Build Something Using Java!

You have already built some very powerful initial applications. After completing your Java learning plan, try to build out at least one of the applications below (or something else that you want to build!):

- **Todo List Application** - Create a simple application that allows users to create, edit, and delete todo items. You could also add features such as due dates, reminders, and categories.
- **Calculator Application** - Create a calculator application that allows users to perform basic arithmetic operations, such as addition, subtraction, multiplication, and division. You could also add advanced features, such as trigonometric functions and logarithms.
- **Address Book Application** - Create an address book application that allows users to store and manage contact information, such as names, phone numbers, and email addresses.
- **Tic Tac Toe Game** - Create a simple Tic Tac Toe game that allows two players to take turns placing X's and O's on a 3x3 grid. You could also add features such as AI opponents and different board sizes.
- **File Manager Application** - Create a file manager application that allows users to navigate and manage files and directories on their computer. You could also add features such as file search and file preview.
- **Online Quiz Application** - Create an online quiz application that allows users to take quizzes on various topics. You could also add features such as timed quizzes and leaderboard.

These are just a few project ideas that you could consider working on after completing your Java learning plan. They range from beginner to intermediate level, and they will help you test your understanding of Java programming concepts and build your portfolio.

Next Steps

After completing your Java learning plan, here are some next steps you could consider:

- **Build More Projects** - Keep building more projects to further reinforce your understanding of Java and to build up your portfolio. Try to challenge yourself with more complex projects that incorporate different aspects of Java programming.
- **Participate in Coding Challenges** - Join coding challenges on websites such as HackerRank, Codewars, or LeetCode to challenge yourself and practice your problem-solving skills.
- **Contribute to Open-Source Projects** - Contribute to open-source projects on platforms like GitHub to gain experience working on real-world projects and collaborating with other developers.
- **Learn a Framework** - Learn a Java framework such as Spring, Hibernate, or Struts to expand your knowledge and skills in Java programming.
- **Read Books on Java** - Read books on Java to deepen your understanding of the language and its best practices. Some recommended books include Effective Java by Joshua Bloch and Head First Java by Kathy Sierra and Bert Bates.