Core Java 8 and Development Tools

Lab Book

Document Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Revision No. | Author | **Summary of Changes** |
| 17-11-2013 | 1.0 | Rathnajothi P | As of updated module content, designed lab book |
| 28-05-2015 | 2.0 | Vinod Satpute | Updated to include new features of Java SE 8, Junit 4 and JAXB 2.0 |
| 25-05-2016 | 3.0 | Tanmaya K Acharya | Updated as per the integrated ELT TOC |
|  |  |  |  |

Table of Contents

[Document Revision History 2](#_Toc451941074)

[Table of Contents 3](#_Toc451941075)

[Getting Started 5](#_Toc451941076)

[Overview 5](#_Toc451941077)

[Setup Checklist for Core Java 5](#_Toc451941078)

[Instructions 5](#_Toc451941079)

[Learning More (Bibliography if applicable) 5](#_Toc451941080)

[Problem Statement/ Case Study (If applicable) 6](#_Toc451941081)

[Lab 1: Working with Java and Eclipse IDE 7](#_Toc451941082)

[1.1: Setting Environment Variable](#_Toc451941083)………………………………………………………………………………………….7

[1.2: Create Java Project 10](#_Toc451941083)

[1.3: Using offline Javadoc API in Eclipse 14](#_Toc451941084)

[Lab 2: Language Fundamentals, Classes and Objects 18](#_Toc451941085)

[Lab 3:Inheritance and Polymorphism 20](#_Toc451941087)

[Lab 4: Abstract classes and Interfaces 22](#_Toc451941088)

[Lab 5: Exception Handling 24](#_Toc451941089)

[Lab 6: Collections 25](#_Toc451941090)

[Lab 7: Property Files and JDBC 4.0 27](#_Toc451941095)

[Lab 8: Introduction to Layered Architecture 2](#_Toc451941096)8

[Appendices 30](#_Toc451941106)

[Appendix A: Naming Conventions](#_Toc451941107) 30

[Appendix B: Table of Figures 3](#_Toc451941108)1

Getting Started

## Overview

This lab book is a guided tour for learning Core Java version 8 and development tools. It comprises of assignments to be done. Refer the demos and work out the assignments given by referring the case studies which will expose you to work with Java applications.

## Setup Checklist for Core Java

Here is what is expected on your machine in order to work with lab assignment.

Minimum System Requirements

* Intel Pentium 90 or higher (P166 recommended)
* Microsoft Windows 7 or higher.
* Memory: (1GB or more recommended)
* Internet Explorer 9.0 or higher or Google Chrome 43 or higher
* Connectivity to Oracle database

Please ensure that the following is done:

* A text editor like Notepad or Eclipse is installed.
* JDK 1.8 or above is installed. (This path is henceforth referred as <java\_home>)

## Instructions

* + - For all Naming conventions, refer Appendix A. All lab assignments should adhere to naming conventions.
    - Create a directory by your name in drive <drive>. In this directory, create a subdirectory java\_assignments. For each lab exercise create a directory as lab <lab number>.

## Learning More (Bibliography if applicable)

* <https://docs.oracle.com/javase/8/docs/>
* Java, The Complete Reference; by Herbert Schildt
* Thinking in Java; by Bruce Eckel
* Beginning Java 8 Fundamentals by KishoriSharan

Problem Statement/ Case Study (If applicable)

1. **Bank Account Management System:**

* Funds Bank needs an application to feed new Account Holder information. AccountHolder will be a person. There are two types of accounts such as SavingsAccount, CurrentAccount.

1. **Employee Medical Insurance Scheme:**

* By default, all employees in an organization will be assigned with a medical insurance scheme based on the salary range and designation of the employee. Refer the below given table to find the eligible insurance scheme specific to an employee.

|  |  |  |
| --- | --- | --- |
| **Salary** | **Designation** | **Insurance scheme** |
| >5000 and < 20000 | System Associate | Scheme C |
| >=20000 and <40000 | Programmer | Scheme B |
| >=40000 | Manager | Scheme A |
| <5000 | Clerk | No Scheme |

Lab 1: Working with Java and Eclipse IDE

|  |  |
| --- | --- |
| **Goals** | Learn and understand the process of:   * + Setting environment variables   + Creating a simple Java Project using Eclipse 3.0 or above |
| **Time** | 45 minutes |

1.1: Setting environment variables from CommandLineSolution:

**Step 1:** Set **JAVA\_HOME** to Jdk1.8 using the following command:

* **Set JAVA\_HOME=**C:\Program Files\Java\jdk1.8.0\_25



Figure 1: Java program

**Step 2:** Set PATH environment variable:

* **Set PATH=%PATH%;%JAVA\_HOME%\bin;**

**Step 3:** Set your current working directory and set classpath.

* Set CLASSPATH=.

**Note:** Classpath searches for the classes required to execute the command. Hence it must be set to the directory containing the class files or the names of the jars delimited by ;

**For example:** C:\Test\myproject\Class;ant.jar

**** Alternatively follow the following steps for setting the environment variables

**Alternate approach:**

**Step 1:** Right click **My Computers**, and select **Properties**🡪**Environment Variables**.



Figure 2: System Properties

**Step 2:** Click **Environment Variables**. The Environment Variables window will bedisplayed.



Figure 3: Environment Variables

**Step 3:** Click **JAVA\_HOME** System Variable if it already exists, or create a new one and set the path of JDK1.8 as shown in the figure.



Figure 4: Edit System Variable

**Step 4:** Click **PATH** System Variable and set it as ***%PATH%;%JAVA\_HOME%\bin***.



Figure 5: Edit System Variable

**Step 5:** Set **CLASSPATH** to your working directory in the **User Variables** tab.



Figure 6: Edit User Variable

## 1.2: Create Java Project

Create a simple java project named ‘MyProject’.

Solution:

**Step 1:** Open **eclipse 4.4**(or above)

**Step 2:** Select **File**🡪**New**🡪**Project** 🡪**Java project**.



Figure 7: Select Wizard

**Step 3:**Click **Next** and provide name for the project.



Figure 8: New Java Project

**Step 4:** Click **Next** and select build options for the project.



Figure 9: Java Settings

**Step 5:** Click **Finish** to complete the project creation.

**Step 6:** Right-click **myproject**, and select resource type that has to be created.



Figure 10: Select Resource

**Step 7:** Providename and other details for the class, and click **Finish**.



Figure 11: Java Class

This will open **MyClass.java** in the editor, with ready skeleton for the class, default constructor, **main()** method, and necessary **javadoc** comments.

To run this class, select **Run** from toolbar, or select **Run As** 🡪**Java application**. Alternatively, you can select **Run..**and you will be guided through a wizard, for the selection of class containing **main()** method.

Console window will show the output.

## 1.3: Using offline Javadoc API in Eclipse

**Step 1:** Open **eclipse 4.4**(or above)

**Step2:**From eclipse Window 🡪 Preferences 🡪Java 🡪 "Installed JREs" select available JRE (jdk1.8.0\_25 for instance) and click Edit.



**Step3:**Select all the "JRE System libraries" using Control+A.

**Step 4:** Click "Javadoc Location"

**Step 5:**Change "Javadoc location path:" from <http://download.oracle.com/javase/8/docs/api/> to "file:/E:/Java/docs/api/".



**Step 6**: Close all windows by either clicking on ok/apply.

**Step 7**: Open the Javadoc view from Window 🡪 Show View 🡪 Javadoc.



**Note:** Henceforth whenever you select any class or method in Editor Window, it Javadoc view will display the reference documentation.



If you want to open the Java documentation for specified resource as html page, right click in the Javadoc view 🡪 Open Attached Javadoc.

Lab 2: Language Fundamentals, Classes and Objects

|  |  |
| --- | --- |
| **Goals** | At the end of this lab session, you will be able to:   * + Write a Java program that displays person details   + Working with Conditional Statements   + Create Classes and Objects |
| **Time** | 120 minutes |

2.1 Write a java program to print person details in the format as shown below:

Person Details:

\_\_\_\_\_\_\_\_\_\_\_\_

First Name: Divya

Last Name: Bharathi

Gender: F

Age: 20

Weight: 85.55

Figure 12: Sample output of Person details

2.2: Write a program to accept a number from user as a command line argument and check whether the given number is positive or negative number.

2.3: Refer the class diagram given below and createa personclass.



Figure 13: Class Diagram of Person

Create default and parameterized constructor for Person class.

Also Create “PersonMain.java” program and write code for following operations:

1. Create an object of Person class and specify person details through constructor.
2. Display the details in the format given in Lab assignment 2.1

2.4: Modify Lab assignment 2.3 to accept phone number of a person. Create a newmethod to implement the same and also define method for displaying persondetails.

2.5: Modify the above program, to accept only ‘M’ or ‘F’ as gender field values. Use Enumeration for implementing the same.

Lab 3: Inheritance and Polymorphism

|  |  |
| --- | --- |
| **Goals** | At the end of this lab session, you will be able to:   * + Write a Java program that manipulates person details   + Working with Inheritance, Polymorphism |
| **Time** | 120 minutes |

3.1: Refer the case study 1in Page No: 5 and create Account Class as shown below in class diagram. Ensure minimum balance of INR 500 in a bank account is available.



Figure 14: Association of person with account class

1. Create Account for smith with initial balance as INR 2000 and for Kathy with initial balance as 3000.(accNum should be auto generated).
2. Deposit 2000 INR to smith account.
3. Withdraw 2000 INR from Kathy account.
4. Display updated balances in both the account.
5. Generate toString() method.

3.2: Extend the functionality through Inheritanceand polymorphism (Maintenance)

Inherit two classes Savings Account and Current Account from account class. Implement the following in the respective classes.

1. Savings Account
2. Add a variable called minimum Balance and assign final modifier.
3. Override method called withdraw (This method should check for minimum balance and allow withdraw to happen)
4. Current Account
5. Add a variable called overdraft Limit
6. Overridemethod called withdraw (checks whether overdraft limit is reached and returns a boolean value accordingly)

Lab 4: Abstract classes and Interfaces

|  |  |
| --- | --- |
| **Goals** | At the end of this lab session, you will be able to:   * Use of abstract classes and interfaces |
| **Time** | 90 minutes |

4.1: Refer the case study 2 in page no: 5 and create an application for that requirement by creating packages and classes as given below:

1. **com.cg.eis.bean**

In this package, create “Employee” class with different attributes such as id, name, salary, designation, insuranceScheme.

1. **com.cg.eis.service**

This package will contain code for services offered in Employee Insurance System. The service class will have one EmployeeService Interface and its corresponding implementation class.

1. **com.cg.eis.pl**

This package will contain code for getting input from user, produce expected output to the user and invoke services offered by the system.

The services offered by this application currently are:

1. Get employee details from user.
2. Find the insurance scheme for an employee based on salary and designation.
3. Display all the details of an employee.

4.2: Use overrides annotation for the overridden methods available in a derived class of an interface of all the assignments.

4.3: Refer the problem statement 3.1. Modify account class as abstract class and declare withdraw method.

Lab 5: Exception Handling

|  |  |
| --- | --- |
| **Goals** | At the end of this lab session, you will be able to:   * Create and use application specific exceptions |
| **Time** | 120 minutes |

5.1: Modify the Lab assignment 2.3 to validate the full name of an employee. Create and throw a user defined exception if firstName and lastName is blank.

5.2: Validate the age of a person in Lab assignment 3.2 and display proper message by using user defined exception. Age of a person should be above 15.

5.3: Modify the Lab assignment 4.1 to handle exceptions. Create an Exception class named as “EmployeeException”(User defined Exception) in a package named as “com.cg.eis.exception” and throw an exception if salary of an employee is below than 3000. Use Exception Handling mechanism to handle exception properly.

Lab 6: Collections

|  |  |
| --- | --- |
| **Goals** | At the end of this lab session, you will be able to:   * Use Comparator interface * Use Collections * Use Generics with Collection Classes * Use iterators to iterate through Collections |
| **Time** | 180 minutes |

6.1: Write a program to store product names in a string array and sort strings available in an array.

6.2: Modify the above program to store product names in anArrayList, sort strings available in an arrayList and display the names using for-each loop.

6.3: Modify the lab assignment 4.1 to accept multiple employee details and store all employee objects in a HashMap. The functionalities need to be implemented are:

1. Add employee details to HashMap.
2. Accept insurance scheme from user and display employee details based on Insurance scheme
3. Delete an employee details from map.
4. Sort the employee details based on salary and display it.

**Note:** Use generics and Comparable/comparator interface.

**Sample code Snippet of EmployeeServiceImpl class:**

|  |
| --- |
| public class EmployeeServiceImpl {  HashMap<String,Employee> list = new HashMap<String,Employee>();    public void addEmployee(Employee emp) {  //code to add employee  }  public boolean deleteEmployee(int id) {  // code to delete a employee whose id is passed as parameter  }  ………………..  } |

Lab 7: Property Files and JDBC 4.0

|  |  |
| --- | --- |
| **Goals** | At the end of this lab session, you will be able to:   * + Understand how property files can be used. * Use JDBC for connecting to the Database through DriverManager and DataSource |
| **Time** | 240 minutes |

7.1: Write a program to store a person details in a properties file named as “PersonProps.properties” and also do the following tasks:

1. Read data from properties file, load the data into Properties object and display the data in the console.
2. Read data from properties file(using getProperties method) and print data in the console.

7.2: Extend the assignment 6.3 by persisting data into database instead of hashmap and display/delete data from database. Use DriverManager for connecting to the database.

Lab 8: Introduction to Layered Architecture

|  |  |
| --- | --- |
| **Goals** | At the end of this lab session, you will be able to:   * + Develop an complete Java application in layered architecture |
| **Time** | 300 minutes |

8.1: Develop a Mobile Purchase system for a Mobile Sales shop. This application is a part of the system. Consider customer is doing full payment by cash, so payment details are not in the scope of our system.Assume mobile details are available in the table (TableName: mobiles). Each mobile detail have unique id and many quantity is available for each mobile. In this system,administratorshould be able to do the following process:

1. Insert the customer and purchase details into database
   * Before inserting into database, do check that the quantity of the mobile should be greater than 0, else display error message.
2. Update the mobile quantity in mobiles table, once mobile is purchased by a customer.
3. View details of all mobiles available in the shop.
4. Delete a mobile details based on mobile id.
5. Search mobiles based on price range.
6. Write a test case for insert and search mobile service functionalities.

When a customer purchased a mobile, the customer and purchase details have to be inserted to the database through system. Perform the following validations while accepting customer details:

* **Customer name:** Valid value should contain maximum 20 alphabets. Out of 20 Characters, first character should be in UPPERCASE.
* **MailId:** should be valid mail id.
* **Phone number:** Valid value should contain 10 digits exactly.
* **MobileId:** Valid value should contain only 4 digits and it should be one of the mobileid available in mobiles table.
* **PurchaseId:** Generate automatically using sequence.
* **Purchasedate:** Should be the current system date.

**Note**:

1. Use layered architecture while implementing application
2. Handle all exceptions as a user defined exception.
3. Use Datasource for connecting to the database.
4. Read database details from properties file.
5. Use RegEx for performing validations.
6. Adhere to the coding standards and follow best practices.
7. Application should provide the menu options for the above requirements.

Assume mobile details are already available in the database.

**Table Script to be used:**

CREATE TABLE mobiles (mobileid NUMBER PRIMARY KEY, name VARCHAR2 (20), price NUMBER(10,2),quantity VARCHAR2(20));

INSERT INTO mobiles VALUES(1001,’Nokia Lumia 520’,8000,20);

INSERT INTO mobiles VALUES(1002,’Samsung Galaxy IV’,38000,40);

INSERT INTO mobiles VALUES(1003,’Sony xperia C’,15000,30);

//TO DO – INSERT few more mobile details.

CREATE TABLE purchasedetails(purchaseid NUMBER, cname vARCHAR2(20), mailid VARCHAR2(30),phoneno VARCHAR2(20), purchasedate DATE, mobileid references mobiles(mobileid));

Appendices

## Appendix A: Naming Conventions

**Package** names are written in all lower case to avoid conflict with the names of classes or interfaces.Companies use their reversed Internet domain name to begin their package names—for example, com.cg.mypackage for a package named mypackage created by a programmer at cg.com.

Packages in the Java language itself begin with **java. Or javax**.

**Classes and interfaces** The first letter should be capitalized, and if several words are linked together to form the name, the first letter of the inner words should be uppercase (a format that's sometimes called "camelCase").

For classes, the names should typically be nouns. For example:

***Dog***

***Account***

***PrintWriter***

For interfaces, the names should typically be adjectives like

***Runnable***

***Serializable***

**Methods** The first letter should be lowercase, and then normal camelCase rules should be used. In addition, the names should typically be verb-noun pairs. For example:

***getBalance***

***doCalculation***

***setCustomerName***

**Variables** Like methods, the camelCase format should be used, starting with a lowercase letter. Sun recommends short, meaningful names, which sounds good to us. Some examples:

***buttonWidth***

***accountBalance***

***myString***

**Constants** Java constants are created by marking variables static and final. They should be named using uppercase letters with underscore characters as separators:

***MIN\_HEIGHT***

## Appendix B: Table of Figures

[Figure 1: Java program 7](#_Toc372543055)

[Figure 2: System Properties 8](#_Toc372543056)

[Figure 3: Environment Variables 9](#_Toc372543057)

[Figure 4: Edit System Variable 9](#_Toc372543058)

[Figure 5: Edit System Variable 10](#_Toc372543059)

[Figure 6: Edit User Variable 10](#_Toc372543060)

[Figure 7: Select Wizard 11](#_Toc372543061)

[Figure 8: New Java Project 12](#_Toc372543062)

[Figure 9: Java Settings 13](#_Toc372543063)

[Figure 10: Select Resource 13](#_Toc372543064)

[Figure 11: Java Class 14](#_Toc372543065)

[Figure 12: Sample output of Person details 17](#_Toc372543066)

[Figure 13: Class Diagram of Person 17](#_Toc372543067)

[Figure 14: Association of person with account class](#_Toc372543068) 18