Core Java

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
12-04-2017	3.1	Yogini and Zainab	Updated as per the BI TOC

Table of Contents

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.

2. **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	30 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

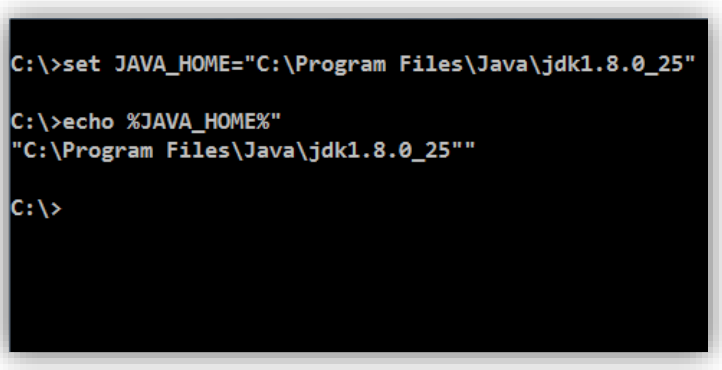


Figure 1: Java program

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**.

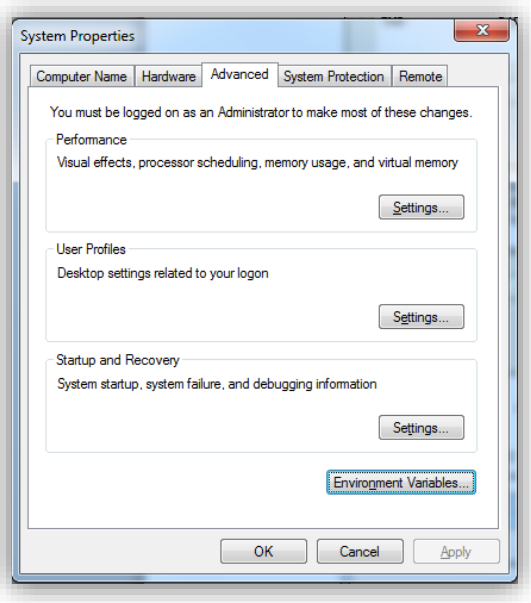


Figure 2: System Properties

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

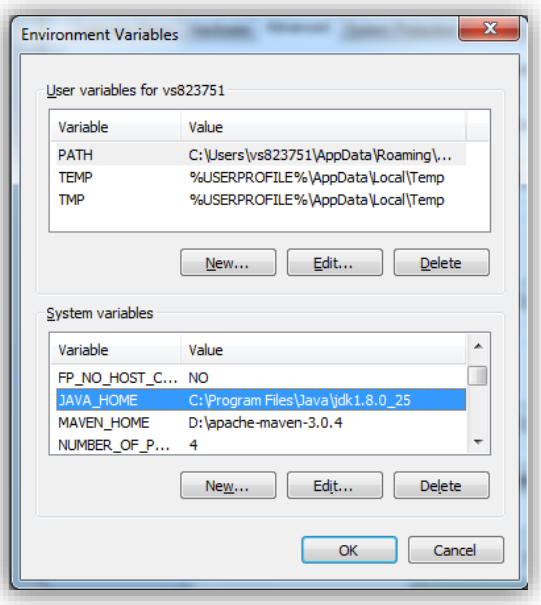


Figure 3: Environment Variables

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.

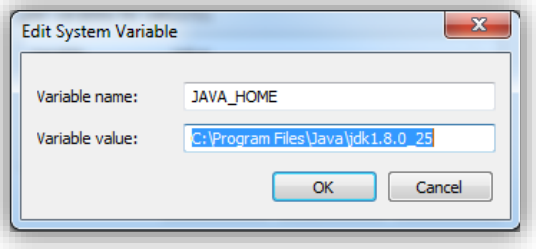


Figure 4: Edit System Variable

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

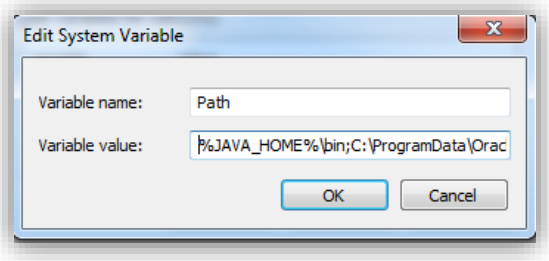


Figure 5: Edit System Variable

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

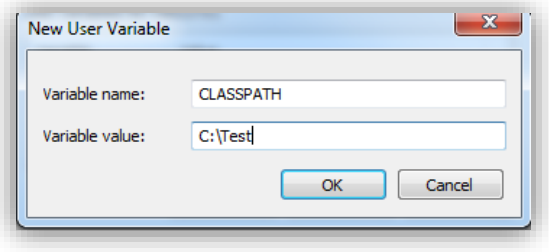


Figure 6: Edit User Variable

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**.

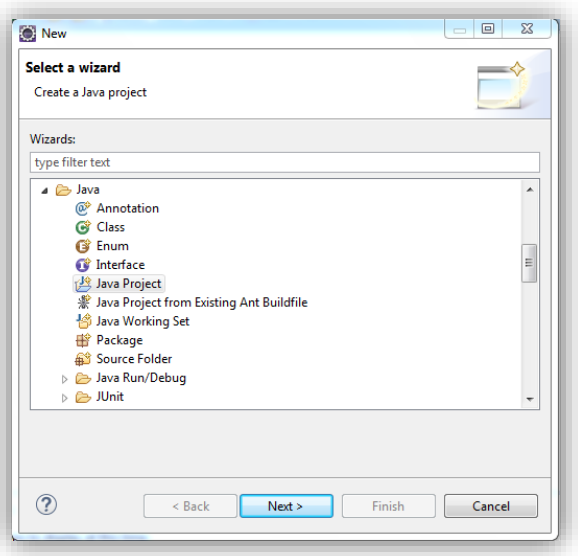


Figure 7: Select Wizard

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

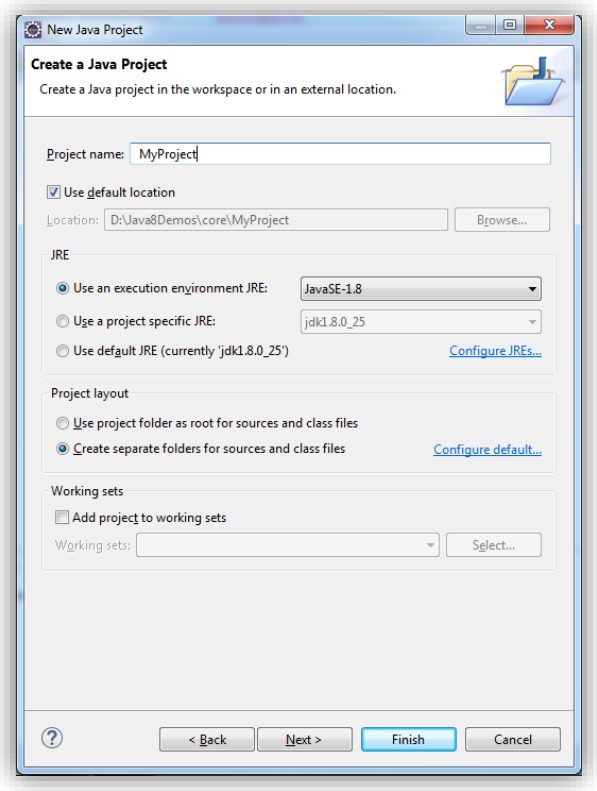


Figure 8: New Java Project

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

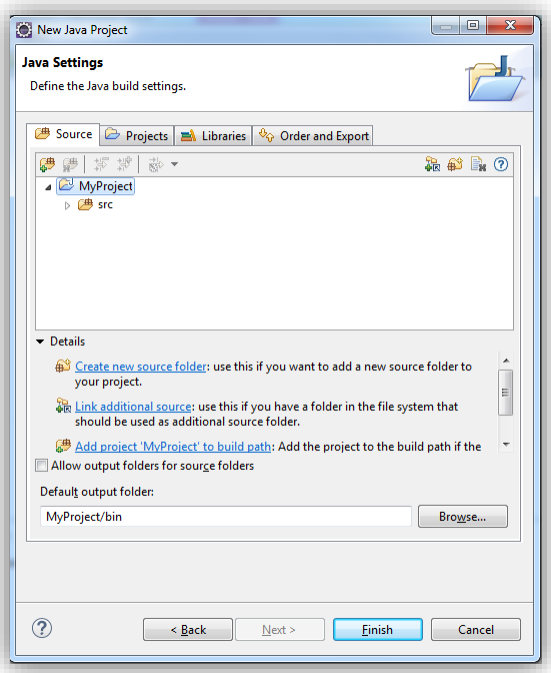


Figure 9: Java Settings

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

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

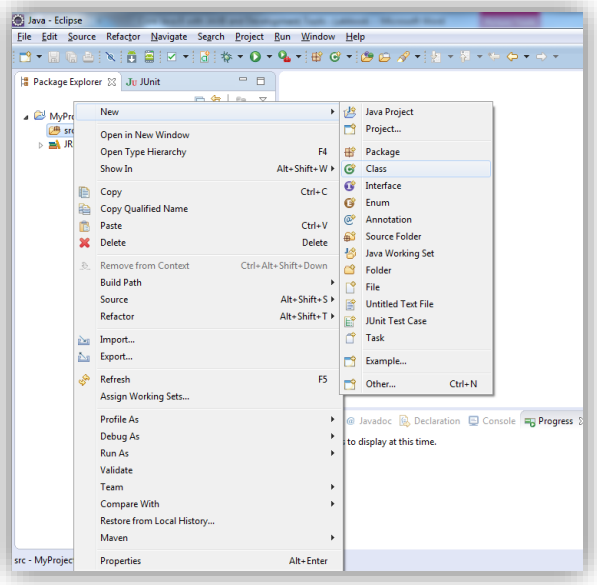


Figure 10: Select Resource

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

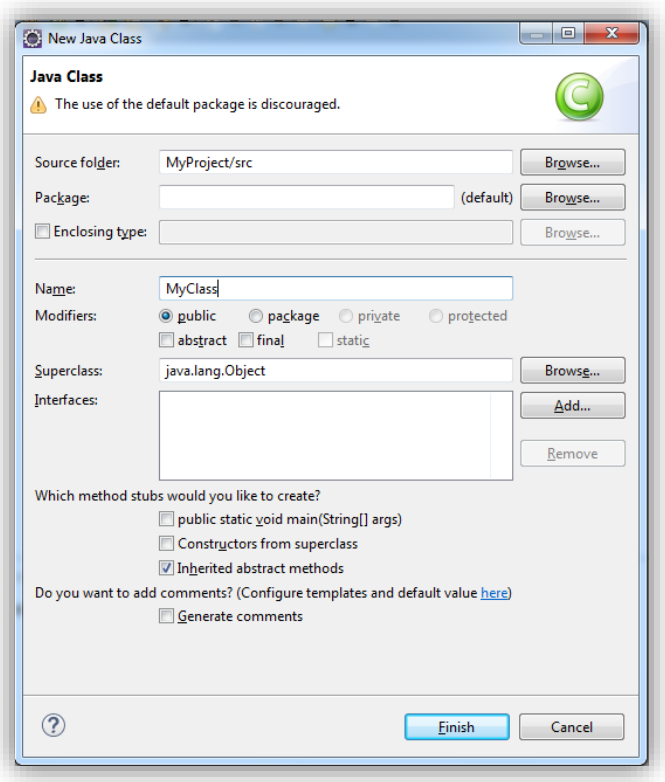


Figure 11: Java Class

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

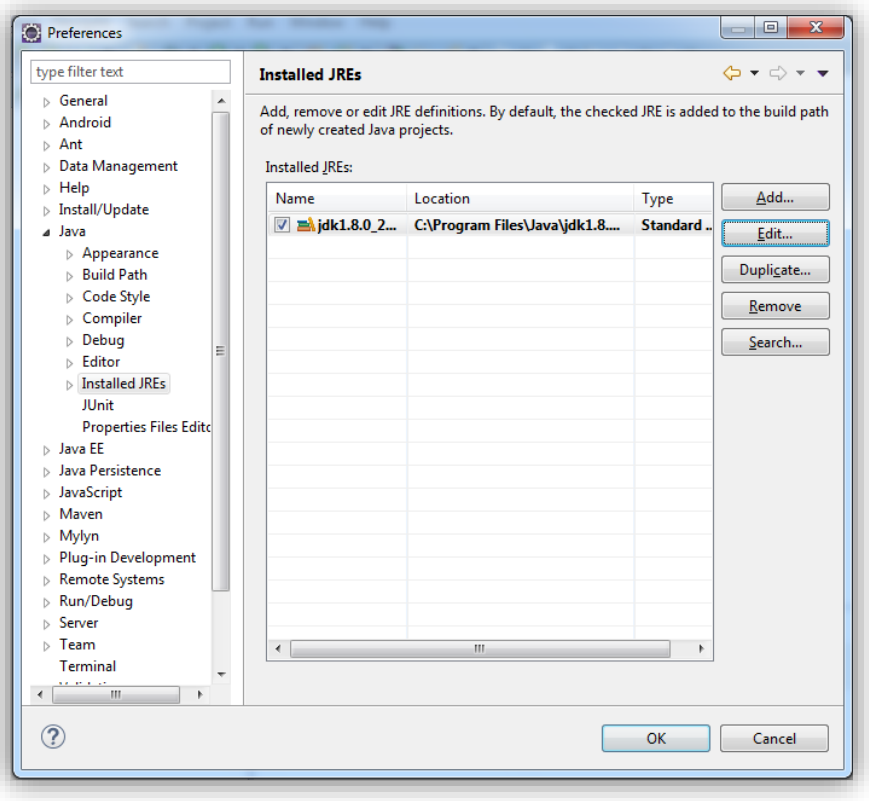
To run this class, select **Run** from toolbar, or select **Run As →Java application**. Alternatively, you can select **Run..**and you will be guided through a wizard, for the selection of class containing **main()** method.

Console window will show the output.

1.3: Using offline Javadoc API in Eclipse

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

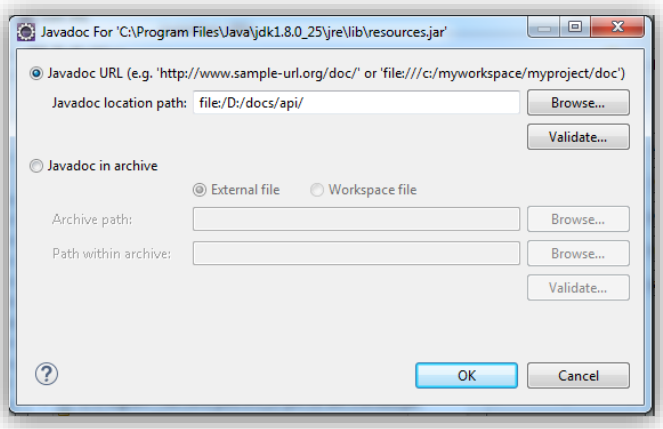
Step2: From eclipse Window → Preferences →Java → "Installed JREs" select available JRE (jdk1.8.0_25 for instance) and click Edit.



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

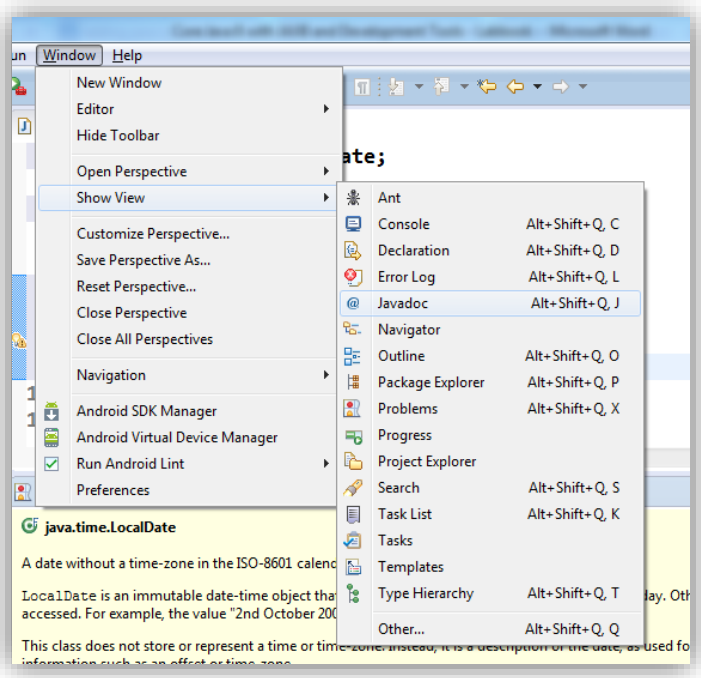
Step 4: Click "Javadoc Location"

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

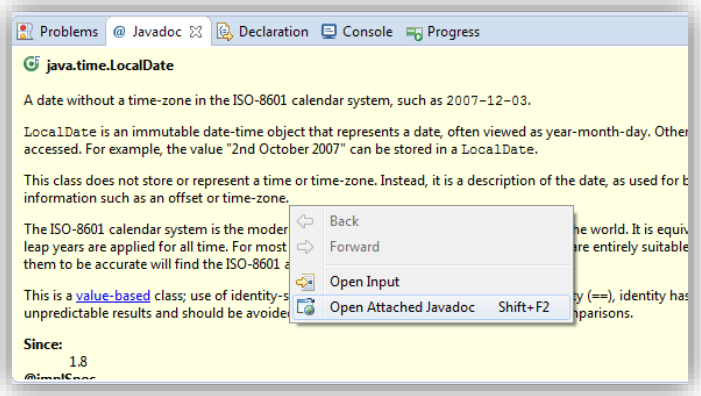


Step 6: Close all windows by either clicking on ok/apply.

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: Getting Started

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	60 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

Lab 3: Basic Language Constructs

Goals	At the end of this lab session, you will be able to: ➢ Work with Conditional Statements
Time	60 minutes

3.1: 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.

Lab 4: Classes and Objects

Goals	At the end of this lab session, you will be able to: ➢ Write a Java program that displays person details
Time	60 minutes

4.1 : Refer the class diagram given below and create a person class.

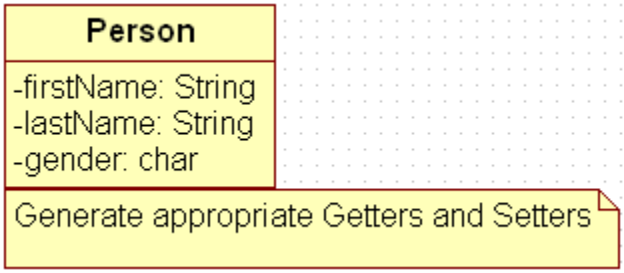


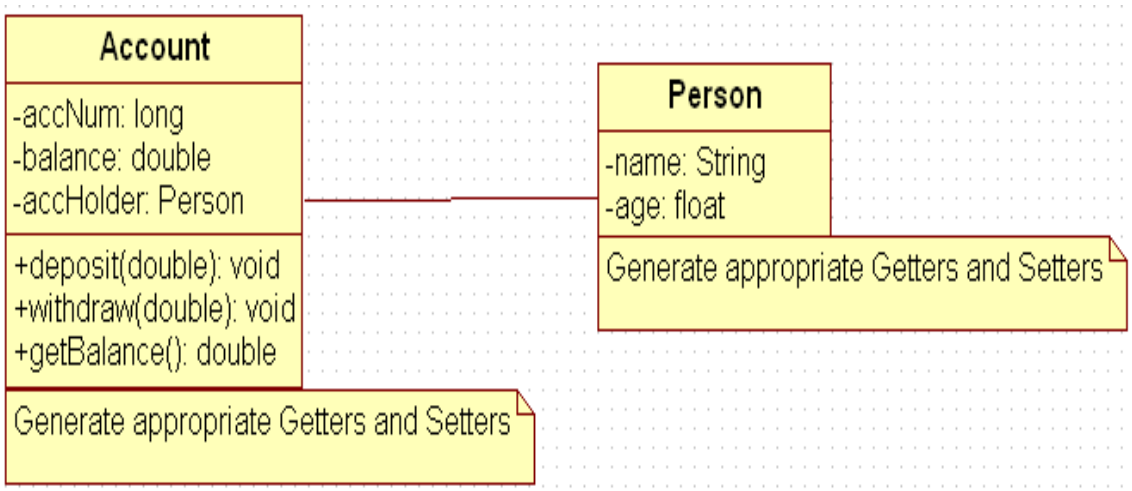
Figure 13: Class Diagram of Person

Create default and parameterized constructor for Person class.

Lab 5: Extending Classes

Goals	At the end of this lab session, you will be able to: ➢ Write a Java program that manipulates details ➢ Work with Inheritance and Polymorphism
Time	60 minutes

5.1 :create Account Class as shown below in class diagram. Ensure minimum balance of INR 500 in a bank account is available.



a) Create Account for smith with initial balance as INR 2000 and for Kathy with initial balance as 3000.(accNum should be auto generated).
b) Deposit 2000 INR to smith account.
c) Withdraw 2000 INR from Kathy account.
d) Display updated balances in both the account.
e) Generate toString() method.

5.2: Extend the functionality through Inheritance and polymorphism (Maintenance)

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

a) Savings Account
    a. Add a variable called minimum Balance and assign final modifier.
    b. Override method called withdraw (This method should check for minimum balance and allow withdraw to happen)

b) Current Account
    a. Add a variable called overdraft Limit
    b. Override method called withdraw (checks whether overdraft limit is reached and returns a boolean value accordingly)

Lab 6: Abstract classes, Interfaces and Packages

Goals	At the end of this lab session, you will be able to: ➢ Use of abstract classes and interfaces
Time	30 minutes

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

a) com.cg.eis.bean

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

b) 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.

c) 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:

i) Get employee details from user.

ii) Find the insurance scheme for an employee based on salary and designation.

iii) Display all the details of an employee.

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

5.3: Refer the problem statement 4.1. Modify account class as abstract class and declare withdraw method.

Lab 7: Exception Handling

Goals	At the end of this lab session, you will be able to: ➢ Create and use application specific exceptions
Time	50 minutes

6.1: 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.

6.2: Validate the age of a person in Lab assignment 4.2 and display proper message by using user defined exception. Age of a person should be above 15.

6.3: Modify the Lab assignment 5.1 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 8: Input Output Classes

Goals	At the end of this lab session, you will be able to: ➢ Read and write data using streams.
Time	20 minutes

8.1: Write a program to read content from file, reverse the content and write the reversed content to the file. (Use Reader and Writer APIs).

8.2: Create a file named as “numbers.txt” which should contain numbers from 0 to 10 delimited by comma. Write a program to read data from numbers.txt using Scanner class API and display only even numbers in the console.

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:

Appendix B: Table of Figures

66

