

# Core Java 8

## 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
15-05-2018	4.0	Yogini Naik	Updated as per the TOC of Java Full Stack

## Table of Contents

<i>Document Revision History .....</i>	<i>2</i>
<i>Table of Contents .....</i>	<i>3</i>
<i>Getting Started .....</i>	<i>5</i>
<i>Overview .....</i>	<i>5</i>
<i>Setup Checklist for Core Java.....</i>	<i>5</i>
<i>Instructions .....</i>	<i>5</i>
<i>Learning More (Bibliography if applicable) .....</i>	<i>5</i>
<i>Problem Statement/ Case Study (If applicable) .....</i>	<i>6</i>
<i>Lab 1: Working with Java and Eclipse IDE.....</i>	<i>7</i>
<i>1.2: Create Java Project .....</i>	<i>10</i>
<i>1.3: Using offline Javadoc API in Eclipse .....</i>	<i>15</i>
<i>Lab 2: Language Fundamentals, Classes and Objects .....</i>	<i>19</i>
<i>Lab 3: Exploring Basic Java Class Libraries .....</i>	<i>21</i>
<i>Lab 4: Inheritance and Polymorphism .....</i>	<i>23</i>
<i>Lab 5: Abstract classes and Interfaces .....</i>	<i>25</i>
<i>Lab 6: Exception Handling.....</i>	<i>27</i>
<i>Lab 7: Arrays and Collections.....</i>	<i>28</i>
<i>Lab 8: Multithreading .....</i>	<i>31</i>
<i>Lab 9: Files IO .....</i>	<i>33</i>
<i>Lab 10: Introduction to JUnit .....</i>	<i>34</i>
<i>10.1: Configuration of JUnit in Eclipse .....</i>	<i>34</i>
<i>10.2: Writing JUnit tests.....</i>	<i>40</i>
<i>Lab 11: Lambda Expressions and Stream API .....</i>	<i>42</i>
<i>Appendices .....</i>	<i>45</i>
<i>Appendix A: Naming Conventions .....</i>	<i>45</i>



*Appendix B: Table of Figures ..... 46*

## Getting Started

### Overview

This lab book is a guided tour for learning Core Java version 8. 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

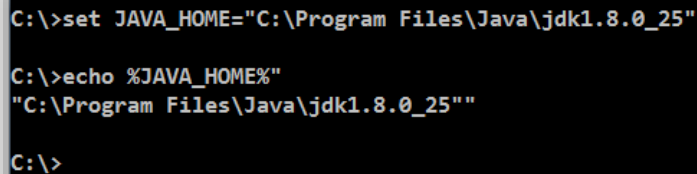
## Lab 1: Working with Java and Eclipse IDE

<b>Goals</b>	Learn and understand the process of: <ul style="list-style-type: none"><li>➤ Setting environment variables</li><li>➤ Creating a simple Java Project using Eclipse 3.0 or above</li></ul>
<b>Time</b>	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**



```
C:\>set JAVA_HOME="C:\Program Files\Java\jdk1.8.0_25"

C:\>echo %JAVA_HOME%
"C:\Program Files\Java\jdk1.8.0_25"

C:\>
```

**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 ;

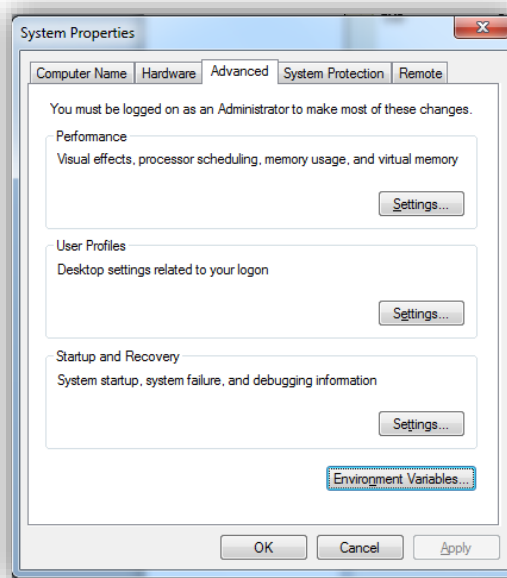
**For example:** C:\Test\myproject\Class;ant.jar



Alternatively follow the following steps for setting the environment variables

### Alternate approach:

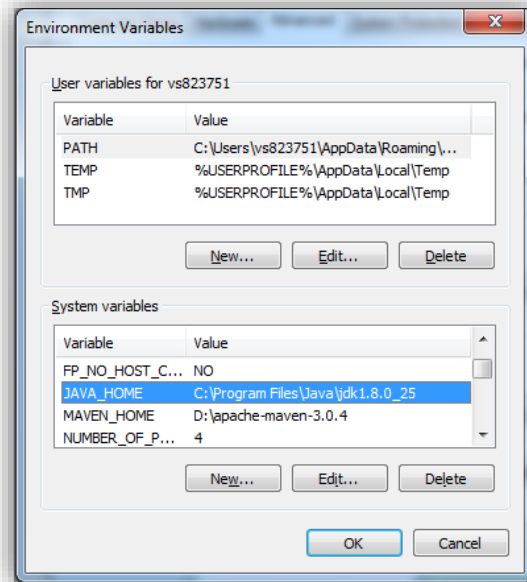
**Step 1:** Right click **My Computers**, and select **Properties**→**Environment Variables**.



**Figure 2: System Properties**

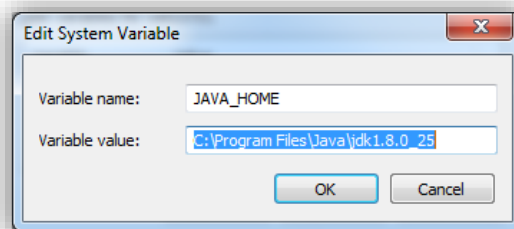
**Step 2:** Click **Environment Variables**. The Environment Variables window will be displayed.





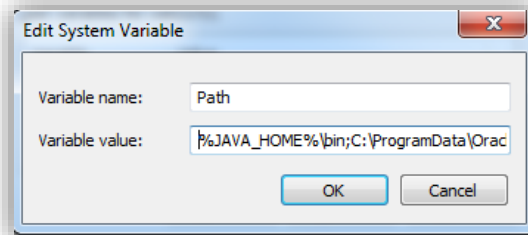
**Figure 3: Environment Variables**

**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.



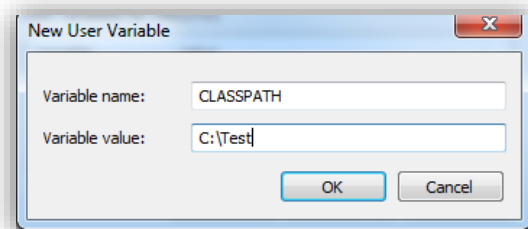
**Figure 4: Edit System Variable**

**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**

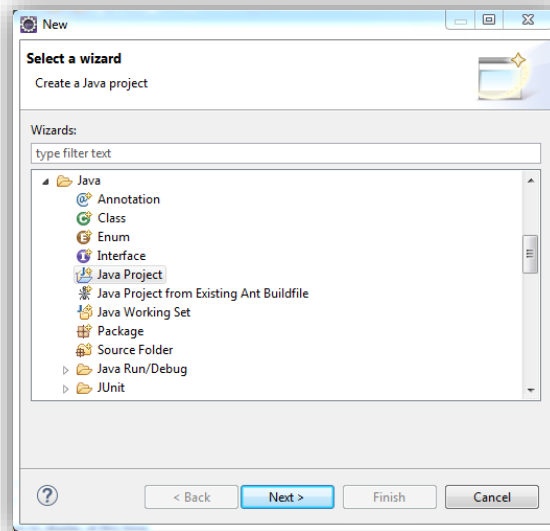
## 1.2: Create Java Project

Create a simple java project named 'MyProject'.

**Solution:**

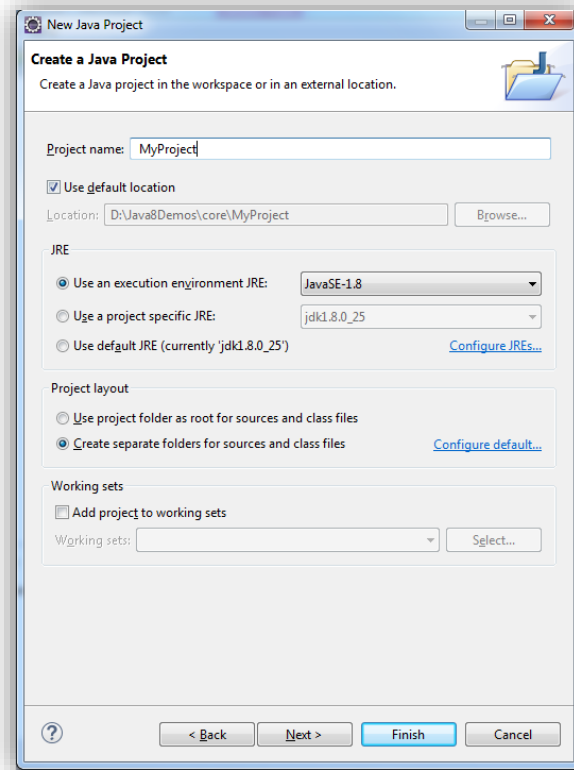
**Step 1:** Open **eclipse 4.4**(or above)

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



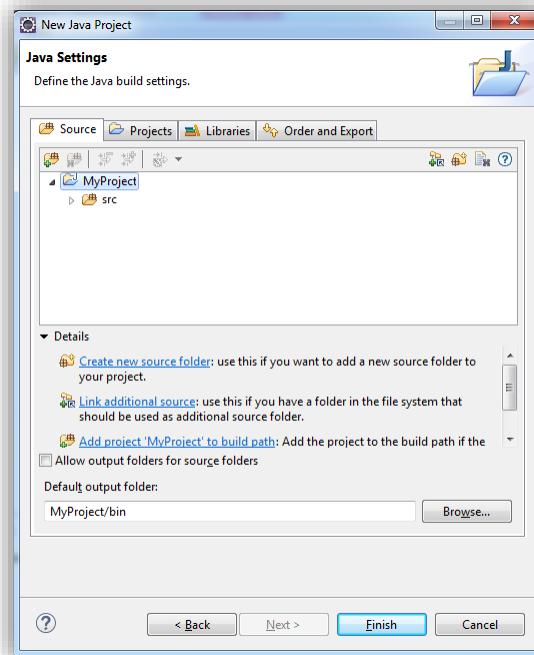
**Figure 7: Select Wizard**

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



**Figure 8: New Java Project**

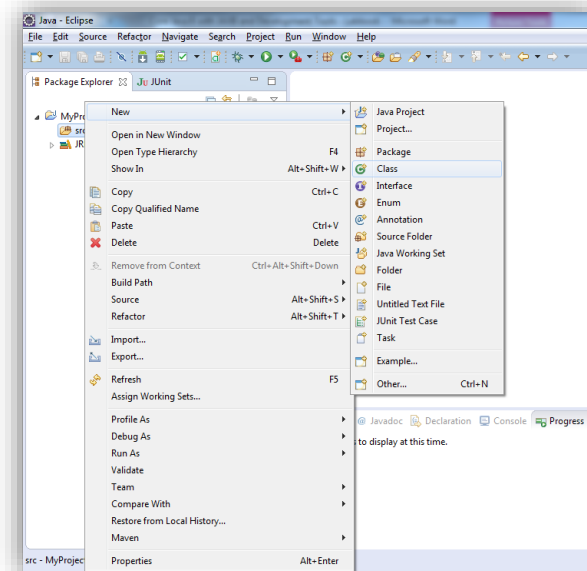
**Step 4:** Click **Next** and select build options for the project.



**Figure 9: Java Settings**

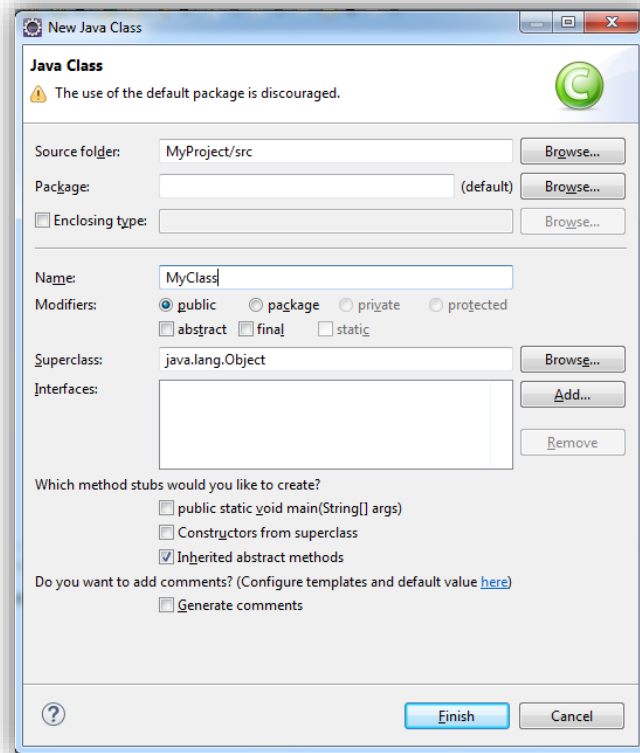
**Step 5:** Click **Finish** to complete the project creation.

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



**Figure 10: Select Resource**

**Step 7:** Provide name 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.

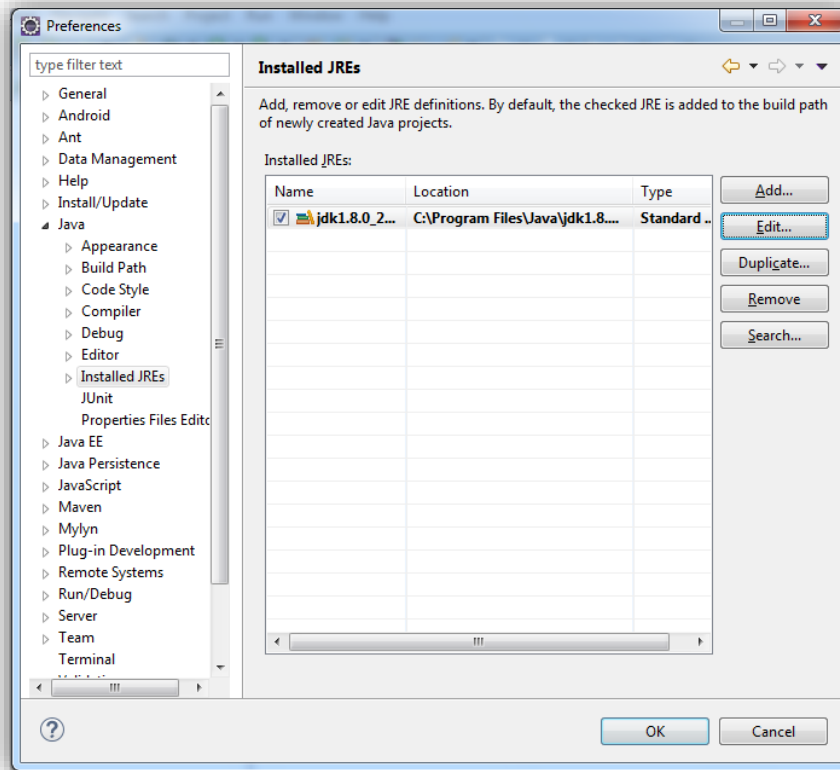
To run this class, select **Run** from toolbar, or select **Run As → Java application**. Alternatively, you can select **Run..** and you will be guided through a wizard, for the selection of class containing **main()** method.

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.

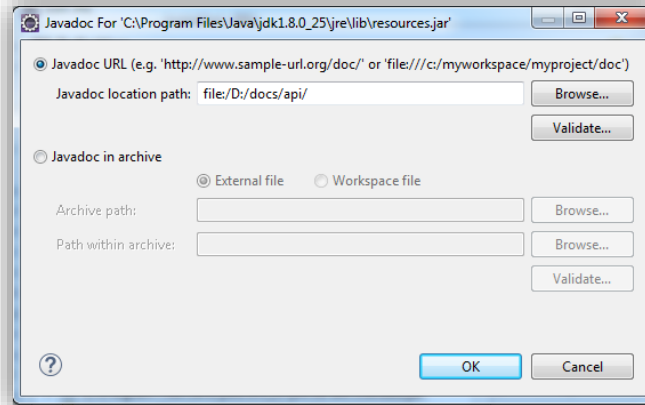


**Step3:**Select all the "JRE System libraries" using Control+A.

#### Step 4: Click "Javadoc Location"

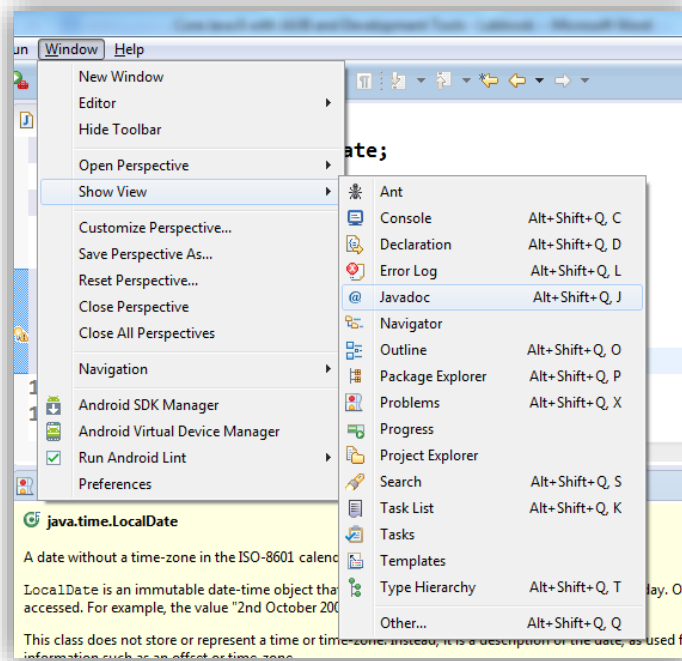
**Step 5:** Change "Javadoc location path:" from <http://download.oracle.com/javase/8/docs/api/> to "file:/E:/Java/docs/api/".



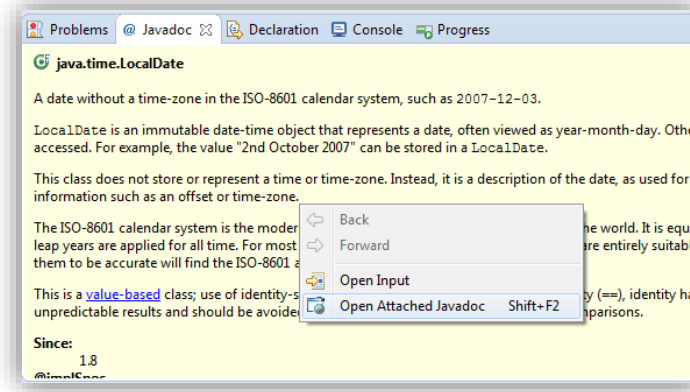


**Step 6:** Close all windows by either clicking on ok/apply.

**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

<b>Goals</b>	At the end of this lab session, you will be able to: <ul style="list-style-type: none"> <li>➤ Write a Java program that displays person details</li> <li>➤ Working with Conditional Statements</li> <li>➤ Create Classes and Objects</li> </ul>
<b>Time</b>	120 minutes

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

```

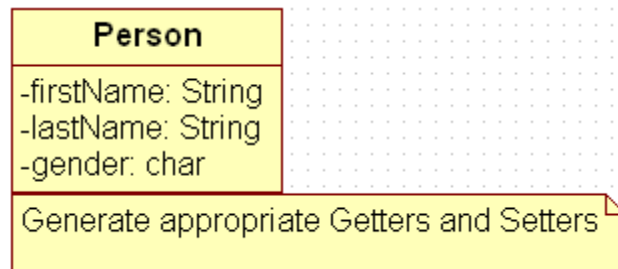
Person Details:
_____

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 create a person class.



**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:

- a) Create an object of Person class and specify person details through constructor.
- b) 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 new method to implement the same and also define method for displaying person details.

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

## Lab 3: Exploring Basic Java Class Libraries

<b>Goals</b>	At the end of this lab session, you will be able to: <ul style="list-style-type: none"><li>➤ Working with Basic Java Class Libraries</li><li>➤ Working with Strings and Date and Time API</li></ul>
<b>Time</b>	100 minutes

3.1: Create a method which can perform a particular String operation based on the user's choice. The method should accept the String object and the user's choice and return the output of the operation.

Options are

- Add the String to itself
- Replace odd positions with #
- Remove duplicate characters in the String
- Change odd characters to upper case

3.2: Create a method that accepts a String and checks if it is a positive string. A string is considered a positive string, if on moving from left to right each character in the String comes after the previous characters in the Alphabetical order. For Example: ANT is a positive String (Since T comes after N and N comes after A). The method should return true if the entered string is positive.

3.3: Create a method to accept date and print the duration in days, months and years with regards to current system date.

3.4: Revise exercise 3.3 to accept two LocalDate's and print the duration between dates in days, months and years.

3.5: Create a method to accept product purchase date and warrantee period (in terms of months and years). Print the date on which warrantee of product expires.

3.6: Create a method which accept zone id and print the current date and time with respect to given zone. (Hint: Few zones to test your code. America/New\_York, Europe/London, Asia/Tokyo, US/Pacific, Africa/Cairo, Australia/Sydney etc.)

3.7 : You are asked to create an application for registering the details of jobseeker. The requirement is:

Username should always end with \_job and there should be atleast minimum of 8 characters to the left

of \_job. Write a function to validate the same. Return true in case the validation is passed. In case of validation failure return false.

3.8 : Create a method that can accept an array of String objects and sort in alphabetical order.

## CORE JAVA 8 LAB BOOK

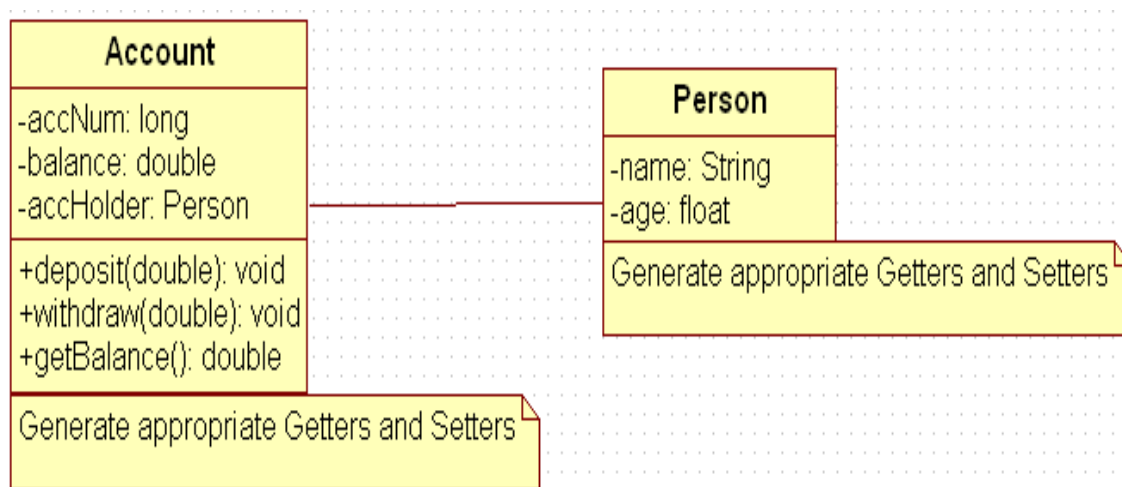
The elements in the left half should be completely in uppercase and the elements in the right half should be completely in lower case. Return the resulting array.

Note: If there are odd number of String objects, then  $(n/2)+1$  elements should be in UPPPERCASE

## Lab 4: Inheritance and Polymorphism

<b>Goals</b>	At the end of this lab session, you will be able to: <ul style="list-style-type: none"> <li>➤ Write a Java program that manipulates person details</li> <li>➤ Working with Inheritance, Polymorphism</li> </ul>
<b>Time</b>	120 minutes

4.1: Refer the case study 1 in 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**

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

#### 4.2: 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 5: Abstract classes and Interfaces

<b>Goals</b>	At the end of this lab session, you will be able to: ➤ Use of abstract classes and interfaces
<b>Time</b>	90 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 6: Exception Handling

<b>Goals</b>	At the end of this lab session, you will be able to: <ul style="list-style-type: none"><li>➤ Create and use application specific exceptions</li></ul>
<b>Time</b>	120 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 7: Arrays and Collections

<b>Goals</b>	At the end of this lab session, you will be able to:
	<ul style="list-style-type: none"> <li>➤ Use Comparator interface</li> <li>➤ Use Collections</li> <li>➤ Use Generics with Collection Classes</li> <li>➤ Use iterators to iterate through Collections</li> </ul>
<b>Time</b>	180 minutes

7.1: Create a method which accepts an integer array, reverse the numbers in the array and returns the resulting array in sorted order

Method Name:	getSorted
Method Description	Return the resulting array after reversing the numbers and sorting it
Argument	Int []
Return Type	Int
Logic	Accept an interger array ,reverse the numbers in the array ,sort it and return the resulting array Hint 1. Convert the numbers to String to reverse it 2. Use Collection APIs to sort it Ex: {12,23,96,45} Step 1: Reverse numbers {21,32,69,54}

7.2: Create a method which can perform the following operations on two String objects S1 and S2. The output of each operation should be added to an arraylist and the arraylist should be Returned (Assume S2 is of smaller size)

Examples for below statements are shown in the Logic part

1. Character in each alternate index of S1 should be replaced with S2
2. If S2 appears more than once in S1, replace the last occurrence of S2 in S1 with the reverse of S2,else return S1+S2
3. If S2 appears more than once in S1, delete the first occurrence of S2 in S1, else return S1
4. Divide S2 into two halves and add the first half to the beginning of the S1 and second half to the end of S1.

Note: If there are odd number of letters in S2, then add (n/2) +1 letters to the beginning and remaining letters to the end.(n is the number of letters in S2)

5. If S1 contains characters that is in S2 change all such characters to \*

7.3: Create a method which can remove a List from another List

Method Name :	removeElements
Method Description :	Removes the elements in one list that is present in the second list also.
Argument	List<String> list1, List<String> list2;
Return Type	List- ArrayList contains the resulting List after the removal process
Logic	Accept two List objects list1 and list2 and remove the elements from list1 that are present in list2.  This should be done in single step process without using loop. Hint: Use the List API which removes all the items in List1 which are contained in List2

7.4: Create a method which accepts an array of numbers and returns the numbers and their squares in an HashMap

Method Name	getSquares
Method Description	Accepts a list of numbers and return their squares
Argument	int[]
Return Type	Map
Logic	Iterate through the list, find the square of each number and add the elements to a map object with the number as the key and the square as the value

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

7.6: Modify the lab assignment 5.1 to accept multiple employee details and store all employee objects in a HashMap. The functionalities need to be implemented are:

- i) Add employee details to HashMap.
- ii) Accept insurance scheme from user and display employee details based on Insurance scheme
- iii) Delete an employee details from map.
- iv) 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 8: Multithreading

<b>Goals</b>	At the end of this lab session, you will be able to: <ul style="list-style-type: none"><li>➤ To process Multithreading program with Thread priority, Synchronization concept, Inter thread communication using wait(),notify()</li></ul>
<b>Time</b>	120 minutes

8.1: Write a program to do the following operations using Thread:

- Create an user defined Thread class called as “CopyDataThread .java”.
- This class will be designed to copy the content from one file “source.txt ” to another file “target.txt” and after every 10 characters copied, “10 characters are copied” message will be shown to user.(Keep delay of 5 seconds after every 10 characters read.)
- Create another class “FileProgram.java” which will create above thread.Pass required File Stream classes to CopyDataThread constructor and implement the above functionality.

8.2: Write a thread program to display timer where timer will get refresh after every 10 seconds. (Use Runnable implementation)

8.3: Write a Java Program, where one thread prints a number (Generate a random number using Math.random()) and another thread prints the factorial of that given number. Both the outputs should alternate each other.

E.g.: Number: 2  
Factorial of 2: 2  
Number: 5  
Factorial of 5: 120

8.4: Write a program using Inter Thread Communication for Electronic shop. In the billing counter first customer gives products for billing and the billing counter person should wait for him. Once he receives the products the billing person then bills the products

Expected output: Customer giving products to billing person  
Billing person bills the products.



## CORE JAVA 8 LAB BOOK



## Lab 9: Files IO

<b>Goals</b>	At the end of this lab session, you will be able to: <ul style="list-style-type: none"><li>➤ Read and write data using streams.</li><li>➤ Serialize and Deserialize objects</li></ul>
<b>Time</b>	75 minutes

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

9.2: 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.

9.3: Enhance the lab assignment 6.3 by adding functionality in service class to write employee objects into a File. Also read employee details from file and display the same in console. Analyze the output of the program.

## Lab 10: Introduction to Junit

<b>Goals</b>	At the end of this lab session, you will be able to: <ul style="list-style-type: none"><li>➤ Configuring JUnit in Eclipse</li><li>➤ Using JUnit to write TestCase for standalone Java Applications</li></ul>
<b>Time</b>	120 minutes

### 10.1: Configuration of JUnit in Eclipse

#### Step 1: Create a Java project.

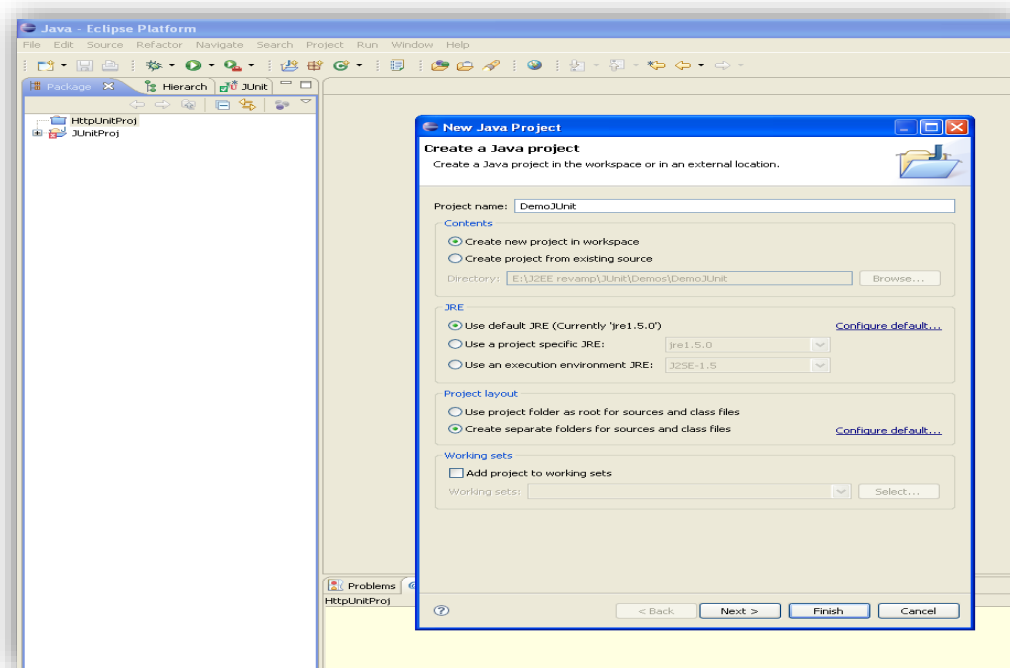
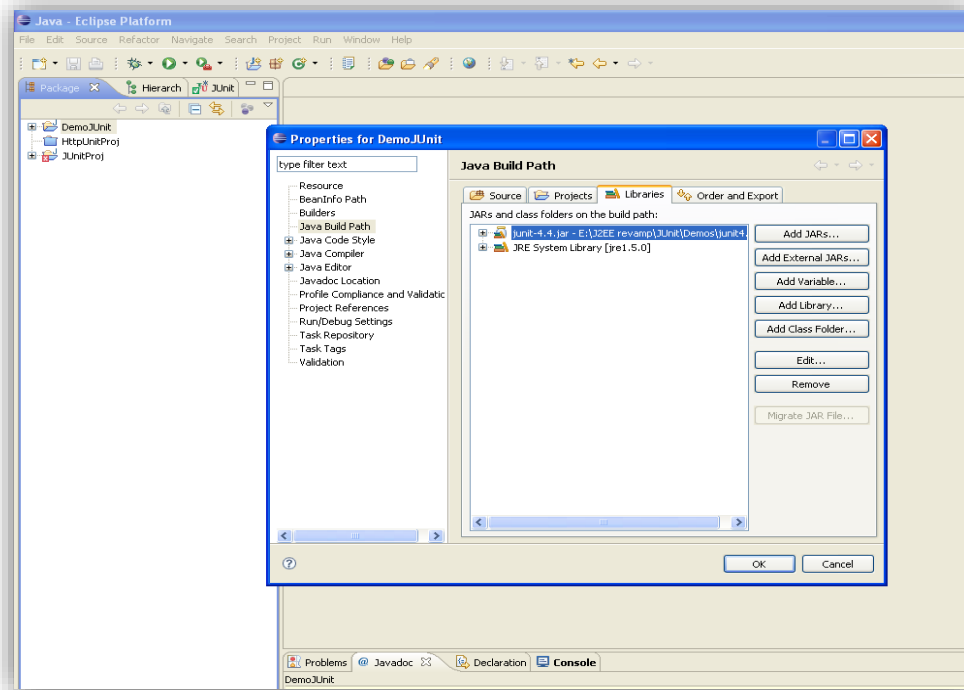


Figure 155: Creating Java Project in Eclipse

#### Step 2: Add junit4.4.jar in the build path of the project.



**Figure 16: Adding junit4.4.jar in the build path**

**Step 3:** Write the java class as follows:

```
public class Person
{
    private String firstName;
    private String lastName;

    public Person(String fname,String lname)
    {
        if(fname == null &&lname==null){
            throw new IllegalArgumentException("Both Names
Cannot be NULL");
        }
        this.firstName=fname;
        this.lastName = lname;
    }

    public String getFullName()
    {
```

```
        String first=(this.firstName != null)? this.firstName:"?";
        String last=(this.lastName != null)? this.lastName:"?";
        return first + " " + last;
    }

    public String getFirstName(){
        return this.firstName;
    }

    public String getLastName(){
        return this.lastName;
    }
    public static void main(String args[])
    {
        Person p=new Person("a","b");
        System.out.println(p.getFirstName());
    }
}
```

Example 1: Person.java

**Step 4:** Write the JUnit test class.

- Create a JUnit test case in Eclipse.

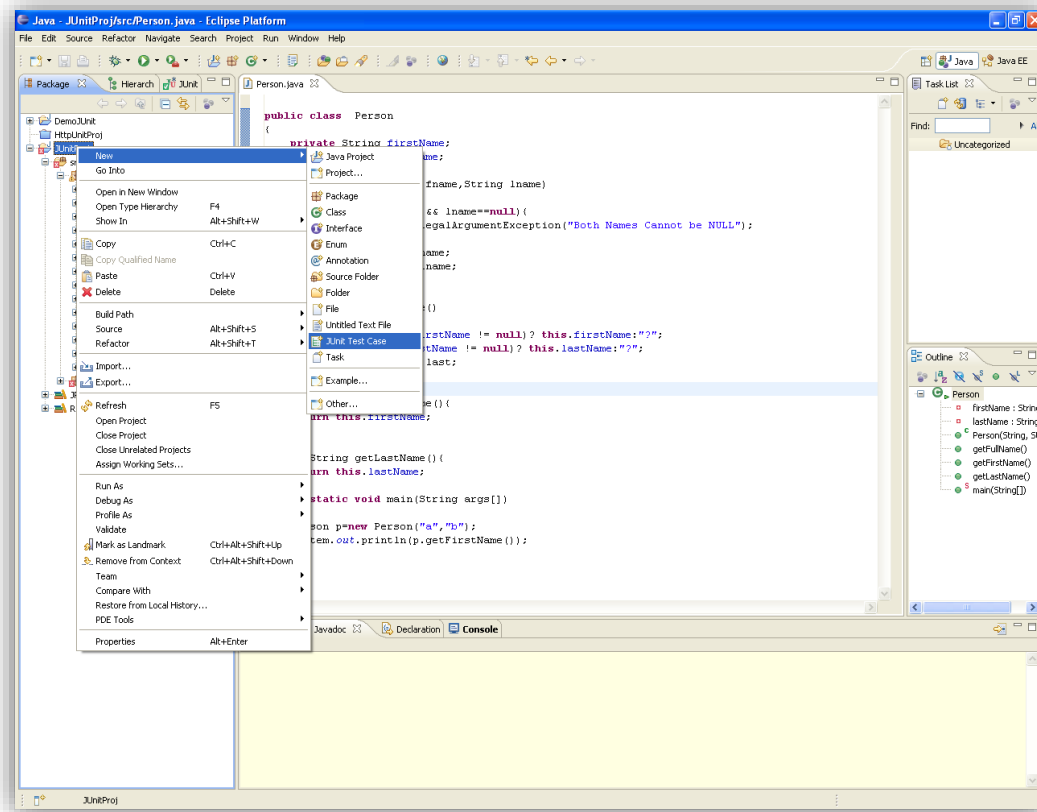
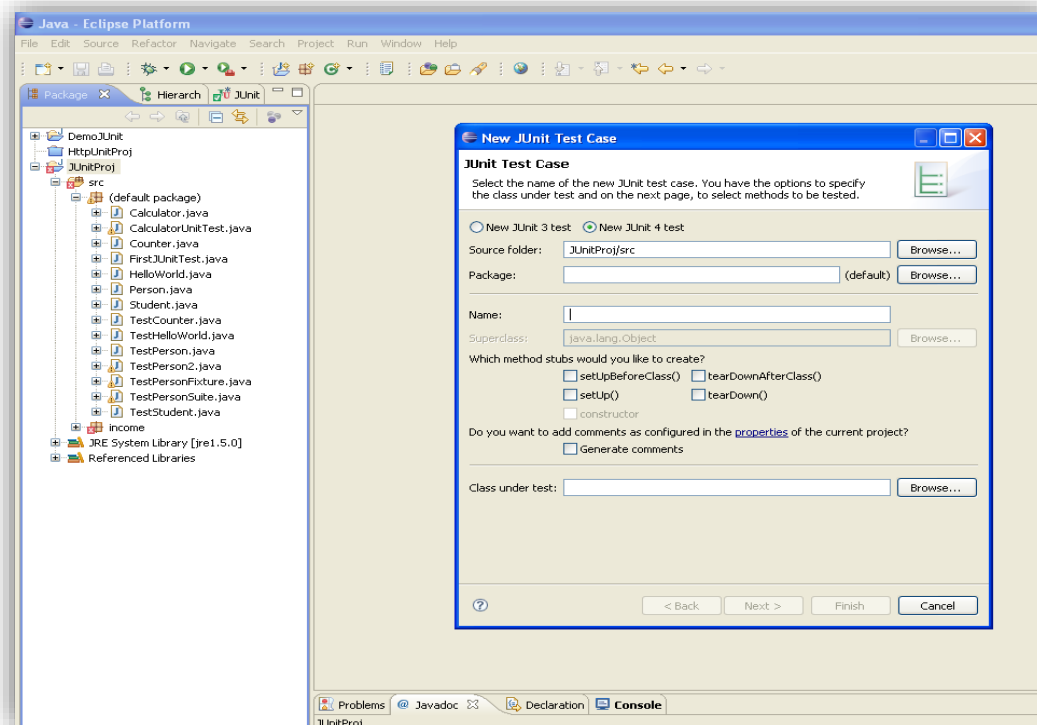


Figure 17: Adding the JUnit test case to the project

- A dialog box opens, where you need to specify the following details:
  - The Junit version that is used
  - The package name and the class name
  - The class under test
  - You can also specify the method stubs that you would like to create



**Figure 18: Specifying information for the test case**

- Write the code as follows:

```
import org.junit.*;
import org.junit.Test;
import static org.junit.Assert.*;

public class TestPerson2
{
    @Test
    public void testGetFullName()
    {
        System.out.println("from TestPerson2");
        Person per = new Person("Robert", "King");
        assertEquals("Robert King", per.getFullName());
    }

    @Test (expected=IllegalArgumentException.class)
    public void testNullsInName()
```

```
    {  
        System.out.println("from TestPerson2 testing exceptions");  
        Person per1 = new Person(null,null);  
    }  
}
```

Example 2: TestPerson2.java

**Step 5:**Run the test case.

- Right click the test case class, and select **RunAs → JUnit Test**.
- The output will be displayed as shown below:

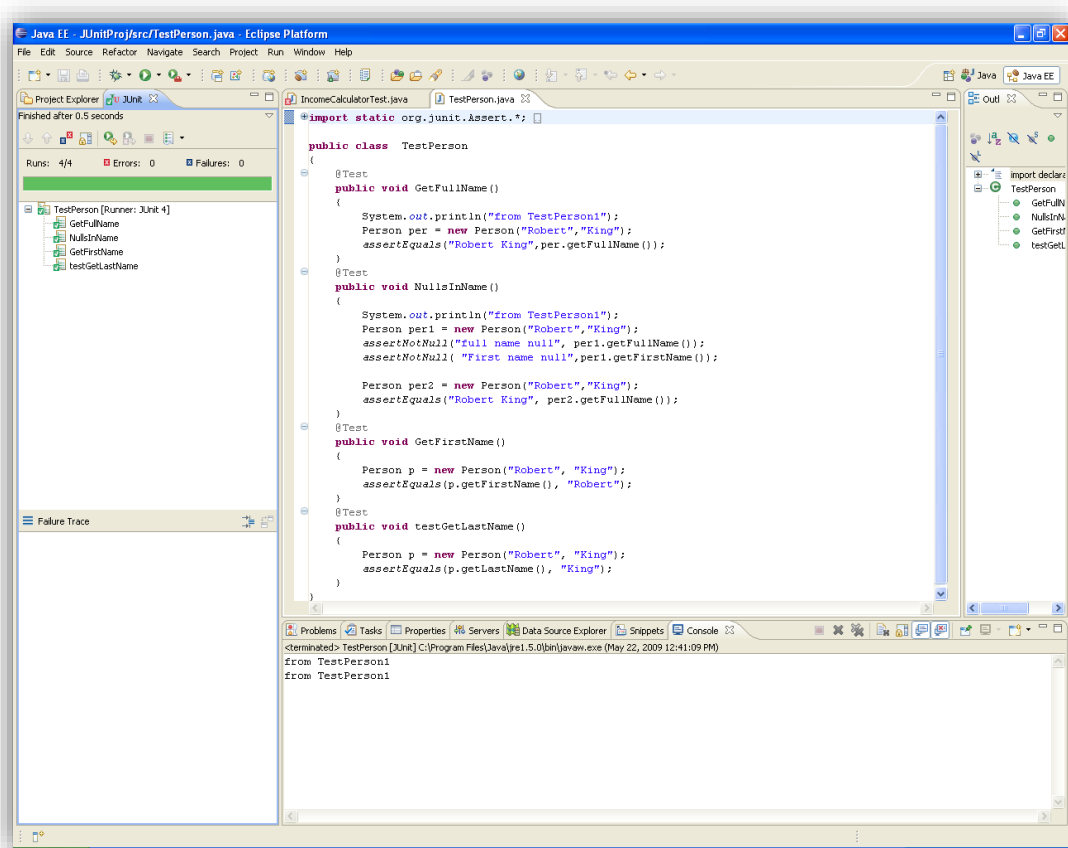


Figure 19: Output of JUnit text case execution

## 10.2: Writing JUnit tests

Consider the following Java program. Write tests for testing various methods in the class

**Solution:**

**Step 1:** Write the following Java Program **Date.java**.

```
class Date
{
    intintDay, intMonth, intYear;
    // Constructor
    Date(int intDay, int intMonth, int intYear) {
        this.intDay = intDay;
        this.intMonth = intMonth;
        this.intYear = intYear;
    }
    // setter and getter methods
    voidsetDay(int intDay)
    {
        this.intDay = intDay;
    }
    intgetDay( )
    {
        return this.intDay;
    }

    voidsetMonth(int intMonth)
    {
        this.intMonth = intMonth;
    }

    intgetMonth( )
    {
        return this.intMonth;
    }

    voidsetYear(int intYear)
    {
        this.intYear=intYear;
    }

    intgetYear( )
    {
        return this.intYear;
    }
}
```



```
}  
public String toString() //converts date obj to string.  
{  
    return "Date is "+intDay+"/"+"intMonth+"/"+"intYear;  
}  
  
} // Date class
```

Example 3: Date.java

**Step 2:** Write test class for testing all the methods of the above program and run it using the eclipse IDE.

**10.2.1:** Consider the Person class created in lab assignment 2.3. This class has some members and corresponding setter and getter methods. Write test case to check the functionality of getter methods and displaydetails method.

**10.2.2:** Consider the lab assignment 6.3 from Exception Handling Lab. Create a new class ExceptionCheck.java which handles an exception. Write a test case to verify if the exception is being handled correctly.

## Lab 11: Lambda Expressions and Stream API

<b>Goals</b>	At the end of this lab session, you will be able to: ➤ Work with lambda expressions and stream API
<b>Time</b>	180 minutes

11.1: Write a lambda expression which accepts x and y numbers and return  $x^y$ .

11.2: Write a method that uses lambda expression to format a given string, where a space is inserted between each character of string. For ex., if input is "CG", then expected output is "C G".

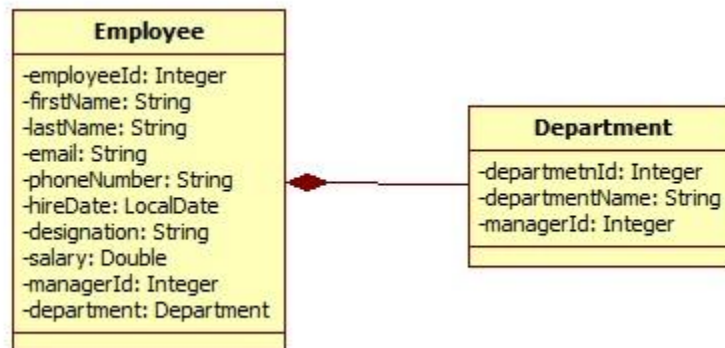
11.3: Write a method that uses lambda expression to accept username and password and return true or false. (**Hint:** Use any custom values for username and password for authentication)

11.4: Write a class with main method to demonstrate instance creation using method reference. (**Hint:** Create any simple class with attributes and getters and setters)

11.5: Write a method to calculate factorial of a number. Test this method using method reference feature.

### Case Study for Steam API:

Refer the classes given below to represent employees and their departments.



**Figure 20: Class Diagram used for Stream API**

Also refer an EmployeeRepository class which is used to create and populate employee's collection with sample data.

## CORE JAVA 8 LAB BOOK



Department.java



Employee.java



EmployeeRepository.java

Create an EmployeeService class which queries on collections provided by EmployeeRepository class for following requirements. Create separate method for each requirement. **(Note:** Each requirement stated below must be attempted by using lambda expressions/stream API. It's mandatory to solve at least 5 questions from following set. However, it is recommended to solve all questions to understand stream API thoroughly).

11.6: Find out the sum of salary of all employees.

11.7: List out department names and count of employees in each department.

11.8: Find out the senior most employee of an organization.

11.9: List employee name and duration of their service in months and days.

11.10: Find out employees without department.

11.11: Find out department without employees.

11.12: Find departments with highest count of employees.

11.13: List employee name, hire date and day of week on which employee has started.

11.14: Revise exercise 11.13 to list employee name, hire date and day of week for employee started on Friday. (Hint: Accept the day name for e.g. FRIDAY and list all employees joined on Friday)

11.15: List employee's names and name of manager to whom he/she reports. Create a report in format "employee name reports to manager name".

11.16: List employee name, salary and salary increased by 15%.

11.17: Find employees who didn't report to anyone (**Hint:** Employees without manager)

11.18: Create a method to accept first name and last name of manager to print name of all his/her subordinates.

11.19: Sort employees by their

- Employee id
- Department id
- First name



## CORE JAVA 8 LAB BOOK

## 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.

Package names 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
Figure 2: System Properties .....	8
Figure 3: Environment Variables .....	9
Figure 4: Edit System Variable .....	9
Figure 5: Edit System Variable .....	10
Figure 6: Edit User Variable .....	10
Figure 7: Select Wizard .....	11
Figure 8: New Java Project .....	12
Figure 9: Java Settings .....	13
Figure 10: Select Resource .....	14
Figure 11: Java Class .....	15
Figure 12: Sample output of Person details .....	19
Figure 13: Class Diagram of Person .....	19
Figure 14: Association of person with account class.....	18