Lab: New Features in Java SE 8

**Date and Time API:**

**Exercise 1: Create a method to accept date and print the duration in days, months and years with regards to current system date.**

**Exercise 2: Revise exercise 1 to accept two LocalDates and print the duration between dates in days, months and years.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*TestDateAssignment1\_2.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package com.igate.date;

import java.time.LocalDate;

import java.time.Month;

import java.time.Period;

import java.time.format.DateTimeFormatter;

import java.util.Scanner;

public class TestDateAssignments\_1\_2

{ public static void main(String[] args)

{ LocalDate today = LocalDate.now();

System.out.println("Current Date="+today);

//Creating LocalDate by providing input arguments

LocalDate firstDay\_2013 = LocalDate.of(2013, Month.JANUARY, 1);

Period period=Period.between(firstDay\_2013, today);

System.out.println(" No Of Month : "+period.getMonths());

System.out.println(" No Of Years : "+period.getYears());

System.out.println(" No Of Days : "+period.getDays());

System.out.println(" No Of Months Other Way : "+period.toTotalMonths());

System.out.println("\*\*\*\*\*\* Enter UR Date Of Joining and Date Of Leaving\*\*\*\*\*\*\*\*\*\*\*");

try(Scanner sc=new Scanner(System.in);)

{

System.out.println("Enter DOJ DD/MM/YYYY :");

String DOJ=sc.next();

System.out.println(" Date Of Joining Is : "+DOJ);

System.out.println("Enter Date Of Resign DD/MM/YYYY :");

String DOL=sc.next();

System.out.println(" Date Of Resign Is : "+DOL);

DateTimeFormatter dtf=DateTimeFormatter.ofPattern("dd/MM/yyyy");

LocalDate dojJ=LocalDate.parse(DOJ, dtf);

LocalDate dojL=LocalDate.parse(DOL, dtf);

Period periodExep=Period.between(dojJ, dojL);

System.out.println(" His Exep In Month : "+periodExep.toTotalMonths());

System.out.println(" His Exep In Year : "+periodExep.getYears());

}

catch(Exception ee) { }

}}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

Exercise 3: 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.

\*\*\*\*\*\*\* TestDateAssignments\_3.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package com.igate.date;

import java.time.LocalDate;

import java.time.Month;

import java.time.Period;

import java.time.format.DateTimeFormatter;

import java.util.Scanner;

public class TestDateAssignments\_3

{

public static void main(String[] args)

{

try(Scanner sc=new Scanner(System.in);)

{

System.out.println("Enter Product Purchase dd-MMM-yyyy :");

String dateOfPurchase=sc.next();

System.out.println(" Date Of Product Purchase Is : "+dateOfPurchase);

System.out.print("Enter Period In Terms of Month And Year : Year : Months");

int years=sc.nextInt();

System.out.print(" : ");

int months=sc.nextInt();

System.out.println(" Warrenty In Year and Month : "+years +" : "+months);

DateTimeFormatter dateTimeFormatter1=DateTimeFormatter.ofPattern("dd-MMM-yyyy");

LocalDate dateOfPurchaseD=LocalDate.parse(dateOfPurchase, dateTimeFormatter1);

LocalDate oneYearAfterPurchaDate=dateOfPurchaseD.plusYears(years);

LocalDate lastWarrentDate=oneYearAfterPurchaDate.plusMonths(months) ;

System.out.println(" UR Purchase Date is :"+dateOfPurchaseD);

System.out.println(" Last Wanrrenty date Is : "+lastWarrentDate);

}

catch(Exception ee) { ee.printStackTrace(); }

}}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Next\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Exercise 4: 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.).**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*TestDateAssignments\_4.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package com.igate.date;

import java.time.LocalDate;

import java.time.LocalDateTime;

import java.time.Month;

import java.time.Period;

import java.time.ZoneId;

import java.time.ZoneOffset;

import java.time.format.DateTimeFormatter;

import java.util.Scanner;

import java.util.Set;

public class TestDateAssignments\_4

{ public static void main(String[] args)

{

System.out.println("\*\*\*\*\*\* Enter UR Date Of Joining and Date Of Leaving\*\*\*\*\*\*\*\*\*\*\*");

try(Scanner sc=new Scanner(System.in);)

{

Set<String> zoneIdList=ZoneId.getAvailableZoneIds();

System.out.println(zoneIdList);

int i=0;

for(String zoneName:zoneIdList)

{

System.out.println(i++ +" : "+zoneName);

}

System.out.println("Enter Zone Id :");

int zoneId= sc.nextInt();

String zoneName=null;

int j=0;

for(String zoneNameTemp:zoneIdList)

{

int index=j++;

if(zoneId==index)

{

zoneName=zoneNameTemp;

break;

}

}

System.out.println(" You Zone Name is : "+zoneName);

LocalDateTime yourZoneDate = LocalDateTime.now(ZoneId.of(zoneName));

System.out.println("Current Date in IST with Respect To UR Zone Is :"+yourZoneDate);

}

catch(Exception ee) { ee.printStackTrace(); } }}

**Exercise 5: Create a method which takes a departure ZonedDateTime and travel time (in terms of hours and minutes) taken by a flight between Mumbai and New York. Calculate and print the arrival time. (Hint: Travel time taken by non-stop flight between Mumbai and New York is 15h 20m).**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*TestDateAssignments\_5.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

/\* LocalDate today = LocalDate.now();

System.out.println("Current Date="+today);

//Creating LocalDate by providing input arguments

LocalDate firstDay\_2014 = LocalDate.of(2014, Month.JANUARY, 1);

\*/

package com.igate.date;

import java.time.LocalDate;

import java.time.LocalDateTime;

import java.time.Month;

import java.time.Period;

import java.time.ZoneId;

import java.time.ZoneOffset;

import java.time.ZonedDateTime;

import java.time.format.DateTimeFormatter;

import java.util.Collections;

import java.util.Scanner;

import java.util.Set;

import java.util.TimeZone;

import java.util.TreeSet;

public class TestDateAssignments\_5

{

public static void main(String[] args)

{

System.out.println("\*\*\*\*Zone List :\*\*\*\*\*\*\*\*");

try(Scanner sc=new Scanner(System.in);)

{

Set<String> zoneIdList=ZoneId.getAvailableZoneIds();

TreeSet<String> soretedZoneList=new TreeSet<String>(zoneIdList);

System.out.println(new TreeSet<String>(zoneIdList));

int i=0;

for(String zoneName:soretedZoneList)

{

System.out.println(i++ +" : "+zoneName);

}

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*276\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.out.println("Enter Departure Zone Id : 276 ");

int zoneDeptId= sc.nextInt();

String deptZoneName=null;

int j=0;

for(String zoneNameTemp:soretedZoneList)

{

int index=j++;

if(zoneDeptId==index)

{

deptZoneName=zoneNameTemp;

break;

}

}

System.out.println(" You Departure Zone Name is : "+deptZoneName);

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.out.println("Enter Departure Date in dd-MMM-yyyy:HH:mm format ");

String dateOfDepartureText=sc.next();

DateTimeFormatter dateFormat=DateTimeFormatter.ofPattern("dd-MMM-yyyy:HH:mm");

LocalDateTime dateOfDeparture=LocalDateTime.parse(dateOfDepartureText, dateFormat);

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*Enter Travel Time in Hours mints\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

int arrivalHours=sc.nextInt();

System.out.print(" : ");

int arrivalMinutes=sc.nextInt();

System.out.println(" Arrival Time In Hours And Minutes: "+arrivalHours +" : "+arrivalMinutes);

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.out.println("Enter Arrival Zone Id : 167 ");

int zoneArrivalId= sc.nextInt();

String arrivalZoneName=null;

j=0;

for(String zoneNameTemp:soretedZoneList)

{

int index=j++;

if(zoneArrivalId==index)

{

arrivalZoneName=zoneNameTemp;

break;

}

}

System.out.println(" You Arrival Zone Name is : "+arrivalZoneName);

LocalDateTime arrivalDateAfterHours=dateOfDeparture.plusHours(arrivalHours);

LocalDateTime arrivalDateAfterMints=arrivalDateAfterHours.plusMinutes(arrivalMinutes) ;

ZonedDateTime arrivalZoneDateTime = ZonedDateTime.of(arrivalDateAfterMints,ZoneId.of(deptZoneName));

System.out.println("Arrival Date Time " +arrivalZoneDateTime.withZoneSameInstant(ZoneId.of(arrivalZoneName)));

}

catch(Exception ee) { ee.printStackTrace();}

}

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Lambda Expressions:\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Exercise 1: Create a lambda expression which accepts x and y numbers and return xy.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*PowerOfNumber.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

@FunctionalInterface

public interface PowerOfNumber

{

//single abstract method to find max between two numbers

public int getPower(int x,int y);

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* TestPowerOfNumberDemo.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

import java.util.Scanner;

public class TestPowerOfNumberDemo

{

public static void main(String args[])

{

PowerOfNumber powerOfNumnber=(int x,int y)->{return (int)(Math.*pow*((double)x,(double)y));};

Scanner sc=new Scanner(System.*in*);

System.*out*.println( "Enter X :");

int x=sc.nextInt();

System.*out*.println( "Enter Y :");

int y=sc.nextInt();

System.*out*.println(x+ " Raise To Power "+y + " Is : "+powerOfNumnber.getPower(x, y));

}

}

**Exercise 2: Crete a method that uses lambda expression to format a given string, where a space is inserted between each character of string. For example, if input is “IGATE”, then expected output is “I G A T E”.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* SplitString.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

public interface SplitString {

public String splitSpring(String msg);

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* TestSplitStringDemo.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

import java.util.Scanner;

import java.util.function.BiFunction;

import java.util.function.Consumer;

import java.util.function.Supplier;

public class TestSplitStringDemo {

public static void main(String[] args) {

//System.out.println("Vaishali".replaceAll("\\B", " "));

try(Scanner sc=new Scanner(System.in)){;

System.out.println( "Enter Name :");

String name=sc.next();

SplitString splitString1 = (String nm) -> nm.replaceAll("\\B", " ");

System.out.println("Splitted Test is : "+splitString1.splitSpring(name));

//Another way is

Supplier<String> supplier = () -> name;

Consumer<String> consumer = (String str)-> name.replaceAll("\\B", " ");

consumer.accept(supplier.get());

}

catch(Exception ee){} }}

**Exercise 3: Create 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)**

**Exercise 4: Create a class with main method to demonstrate instance creation using method reference. (Hint: Create any simple class with attributes and getters and setters)**

**Exercise 5: Create a method to calculate factorial of a number. Test this method using method reference feature.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* IFactorial.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

public interface IFactorial {

public int calcFact1(int number);

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* FactorialImpl.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

public class FactorialImpl implements IFactorial

{

int number;

public int getNumber() {return number;}

public void setNumber(int number){this.number = number;}

@Override

public int calcFact1(int number)

{

int fact=1;

while(number!=1)

{

fact=fact\*number;

number--;

}

return fact;

}

public int calcFact2()

{

int fact=1;

while(number!=1)

{

fact=fact\*number;

number--;

}

return fact;

}

public FactorialImpl(){};

public FactorialImpl(int number){this.number=number;}

public static FactorialImpl createFactorialImpl(int number)

{ return new FactorialImpl(number); }

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*TestFactorial.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

import java.util.function.Consumer;

import java.util.function.Function;

public class TestFactorial

{

public static void main(String[] args)

{

Function<Integer,FactorialImpl> factFunction =

new Function<Integer,FactorialImpl>()

{

public FactorialImpl apply(Integer number)

{

return new FactorialImpl(number);

}

};

FactorialImpl obj=factFunction.apply(6);

System.*out*.println("Fact is "+obj.calcFact2());

//other syntax............

//Consumer<Integer> consumer2 = IFactorial :: calcFact1;

//consumer2.accept(4);

}

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*TestFactorial2.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

import java.util.function.Function;

public class TestFactorial2

{

public static void main(String[] args)

{

Function<Integer,FactorialImpl> factFunction = (Integer number)->FactorialImpl.*createFactorialImpl*(number);

FactorialImpl obj=factFunction.apply(7);

System.*out*.println("Factorial is : "+obj.calcFact2());

}

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*TestFactorial3.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

import java.util.function.Function;

public class TestFactorial3

{

public static void main(String[] args)

{

Function<Integer,FactorialImpl> factFunction = FactorialImpl::*createFactorialImpl*;

FactorialImpl obj=factFunction.apply(7);

System.*out*.println("Factorial is : "+obj.calcFact2());

}

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*TestFactorial4.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

import java.util.function.Function;

public class TestFactorial4 {

public static void main(String[] args) {

IFactorial factObj=new FactorialImpl();

Function<Integer,Integer> funcRef = factObj::calcFact1;

System.*out*.println("Factorial is :"+funcRef.apply(4));

}}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Stream API\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Case Study: Refer the classes given below to represent employees and their departments.**

****

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

****

**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).**

**Exercise 1: Find out the sum of salary of all employees.**

**Exercise 2: List out department names and count of employees in each department.**

**Exercise 3: Find out the senior most employee of an organization.**

**Exercise 4: List employee name and duration of their service in months and days.**

**Exercise 5: Find out employees without department.**

**Exercise 6: Find out department without employees.**

**Exercise 7: Find departments with highest count of employees.**

**Exercise 8: List employee name, hire date and day of week on which employee has started.**

**Exercise 9: Revise exercise 8 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)**

**Exercise 10: List employee’s names and name of manager to whom he/she reports. Create a report in format “employee name reports to manager name” .**

**Exercise 11: List employee name, salary and salary increased by 15%.**

**Exercise 12: Find employees who didn’t report to anyone (Hint: Employees without manager)**

**Exercise 13: Create a method to accept first name and last name of manager to print name of all his/her subordinates.**

**Exercise 14: Sort employees by their**

* **Employee id**
* **Department id**
* **First name**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Employee.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

import java.time.LocalDate;

public class Employee {

@Override

public String toString() {

return "\nEmployee [employeeId=" + employeeId + ", firstName="

+ firstName + ", lastName=" + lastName + ", email=" + email

+ ", phoneNumber=" + phoneNumber + ", hireDate=" + hireDate

+ ", designation=" + designation + ", salary=" + salary

+ ", managerId=" + managerId + ", department=" + department

+ "]";

}

private Integer employeeId;

private String firstName;

private String lastName;

private String email;

private String phoneNumber;

private LocalDate hireDate;

private String designation;

private double salary;

private Integer managerId;

private Department department;

public Employee(Integer employeeId, String firstName, String lastName,

String email, String phoneNumber, LocalDate hireDate,

String designation, double salary, Integer managerId,

Department department) {

super();

this.employeeId = employeeId;

this.firstName = firstName;

this.lastName = lastName;

this.email = email;

this.phoneNumber = phoneNumber;

this.hireDate = hireDate;

this.designation = designation;

this.salary = salary;

this.managerId = managerId;

this.department = department;

}

public Employee() { super(); }

public int getEmployeeId() { return employeeId; }

public void setEmployeeId(int employeeId) { this.employeeId = employeeId; }

public String getFirstName() { return firstName; }

public void setFirstName(String firstName) { this.firstName = firstName; }

public String getLastName() { return lastName; }

public void setLastName(String lastName) { this.lastName = lastName; }

public String getEmail() { return email; }

public void setEmail(String email) { this.email = email; }

public String getPhoneNumber() { return phoneNumber; }

public void setPhoneNumber(String phoneNumber) {this.phoneNumber = phoneNumber; }

public LocalDate getHireDate() { return hireDate; }

public void setHireDate(LocalDate hireDate) { this.hireDate = hireDate; }

public String getDesignation() {return designation; }

public void setDesignation(String designation) { this.designation = designation; }

public double getSalary() { return salary; }

public void setSalary(double salary) { this.salary = salary; }

public Integer getManagerId() { return managerId; }

public void setManagerId(Integer managerId) { this.managerId = managerId; }

public Department getDepartment() { return department; }

public void setDepartment(Department department) { this.department = department; }

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*EmployeeRepository.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

import java.time.LocalDate;

import java.time.Month;

import java.util.ArrayList;

import java.util.List;

public class EmployeeRepository

{

private static final List<Employee> employees;

private static final List<Department> departments;

static

{

departments = new ArrayList<Department>();

//populate departments

departments.add(new Department(10,"Administration",100));

departments.add(new Department(20,"Production",201));

departments.add(new Department(30,"Sales",124));

departments.add(new Department(40,"Finance",205));

departments.add(new Department(50,"Customer Support",100));

departments.add(new Department(60,"ITIM ",100));

//populate employees

employees= new ArrayList<Employee>();

employees.add(new Employee( 100, "Steven", "King",

"SKING", "515.123.4567", LocalDate.of(1997,Month.JUNE,17),

"President", 24000,null, departments.get(0)));

employees.add(new Employee( 101, "Neena", "Kochhar",

"NKOCHHAR", "515.123.4568", LocalDate.of(1999,Month.SEPTEMBER,21),

"Vice President", 17000,100, departments.get(0)));

employees.add(new Employee( 102, "Lex", "De Haan",

"LDEHAAN", "515.123.4569", LocalDate.of(2003,Month.JUNE,13),

"Vice President", 17000,100, departments.get(0)));

employees.add(new Employee( 201, "Michael", "Hartstein",

"MHARTSTE", "515.123.5555", LocalDate.of(1996,Month.FEBRUARY,17),

"Manager", 13000,100, departments.get(1)));

employees.add(new Employee( 202, "Pat", "Fay",

"PFAY", "603.123.6666", LocalDate.of(2007,Month.AUGUST,17),

"Deputy Manager", 6000,201, departments.get(1)));

employees.add(new Employee( 124, "Kevin", "Mourgos",

"KMOURGOS", "650.123.5234", LocalDate.of(2009,Month.NOVEMBER,16),

"Manager", 5800,100, departments.get(2)));

employees.add(new Employee( 141, "Trenna", "Rajs",

"TRAJS", "650.121.8009", LocalDate.of(1995,Month.OCTOBER,17),

"Clerk", 3500,124, departments.get(2)));

employees.add(new Employee( 205, "Shelley", "Higgins",

"SHIGGINS", "515.123.8080", LocalDate.of(2004,Month.JUNE,07),

"Manager", 12000,101, departments.get(3)));

employees.add(new Employee( 206, "William", "Gietz",

"WGIETZ", "515.123.8181", LocalDate.of(2004,Month.JUNE,07),

"Accountant", 8300,205, departments.get(3)));

employees.add(new Employee( 199, "Douglas", "Grant",

"DGRANT", "650.507.9844", LocalDate.of(2000,Month.JANUARY,13),

"Clerk", 2600,205, departments.get(3)));

employees.add(new Employee( 200, "Jennifer", "Whalen",

"JWHALEN", "515.123.4444", LocalDate.of(2015,Month.SEPTEMBER,17),

"Admin Assistant", 4400,102, null));

employees.add(new Employee( 198, "Donald", "OConnell",

"DOCONNEL", "650.507.9833", LocalDate.of(2009,Month.JUNE,21),

"Clerk", 2600,null, null));

}

public static List<Employee> getEmployees()

{ return employees; }

public static List<Department> getDepartments()

{ return departments; }

}

\*\*\*\*\*\*\*\*\*\*\*\*\*EmployeeService.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import java.time.DayOfWeek;

import java.time.LocalDate;

import java.time.LocalDateTime;

import java.time.Month;

import java.time.Period;

import java.time.chrono.ChronoPeriod;

import java.time.temporal.ChronoField;

import java.time.temporal.ChronoUnit;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Comparator;

import java.util.DoubleSummaryStatistics;

import java.util.IntSummaryStatistics;

import java.util.List;

import java.util.Map;

import java.util.Optional;

import java.util.Scanner;

import java.util.Set;

import java.util.function.Predicate;

import java.util.stream.Collector;

import java.util.stream.Collectors;

import java.util.stream.IntStream;

import java.util.stream.Stream;

public class EmployeeService

{

public static void main(String args[])

{

EmployeeRepository eso=new EmployeeRepository();

sumOfSalOfAllEmp(EmployeeRepository.getEmployees());

getDeptWiseEmpList(EmployeeRepository.getEmployees());

getSeniorEmp(EmployeeRepository.getEmployees());

getManagerDetails(EmployeeRepository.getEmployees());

getEmpDuration(EmployeeRepository.getEmployees());

getEmpWithoutDept(EmployeeRepository.getEmployees());

getDeptWithoutEmp(EmployeeRepository.getEmployees(),EmployeeRepository.getDepartments());

getDeptWithHighestEmpCount(EmployeeRepository.getEmployees(),EmployeeRepository.getDepartments());

getEmpHireDetails(EmployeeRepository.getEmployees());

//getEmpHireOnSpecificDay(EmployeeRepository.getEmployees());

getEmpSalHikeDEtails(EmployeeRepository.getEmployees());

getEmpWithoutManager(EmployeeRepository.getEmployees());

//getManagerSubordinates(EmployeeRepository.getEmployees());

sortEmpByEmpId(EmployeeRepository.getEmployees());

sortEmpByDept(EmployeeRepository.getEmployees());

sortEmpByName(EmployeeRepository.getEmployees());

}

private static void sortEmpByName(List<Employee> empList)

{

System.out.println(" Sorted Emp According to First Name ");

empList.stream().

sorted(

(employee1,employee2)->

employee1.getFirstName().compareTo(employee2.getFirstName())

).

forEach(System.out::println);

}

private static void sortEmpByDept(List<Employee> empList)

{

System.out.println(" Sorted Emp According to Dept Name");

empList.stream().

filter(employee->employee.getDepartment()!=null).

sorted((employee1,employee2)->

(employee1.getDepartment().getDepartmentName().

compareTo(employee2.getDepartment().getDepartmentName()))

).

forEach(System.out::println);

}

private static void sortEmpByEmpId(List<Employee> empList)

{

System.out.println(" Sorted Emp According to ID ");

empList.stream().

sorted((employee1,employee2)->

(employee1.getEmployeeId()-employee2.getEmployeeId())

).

forEach(System.out::println);

}

private static void getManagerSubordinates(List<Employee> empList)

{

System.out.println("\*\*\*\*Create a method to accept first name and last name");

System.out.println("of manager to print name of all his/her subordinates. ");

System.out.println(" Enter Manager Name :");

try(Scanner sc=new Scanner(System.in))

{

String mgrFName=sc.next();

String mgrLName=sc.next();

int mgrId = empList.stream().

filter(e->e.getFirstName().equals(mgrFName) && e.getLastName().equals(mgrLName)).

collect(Collectors.toList()).

get(0).

getEmployeeId();

empList.stream().

filter(employee->employee.getManagerId()!=null &&employee.getManagerId()==mgrId).

forEach(System.out::println);

}

catch(Exception e){}

}

private static void getEmpWithoutManager(List<Employee> empList)

{

System.out.println("\*\*\*\*\*List employee Without Manager\*\*\*\*\* ");

empList.stream().

filter(employee->employee.getManagerId()==null).

forEach(employee->System.out.println(employee.getEmployeeId()+ " : " +

employee.getFirstName()));

}

private static void getEmpSalHikeDEtails(List<Employee> empList)

{

System.out.println("\*\*\*\*\*List employee name, salary and salary increased by 15%.\*\*\*\*\* ");

empList.stream()

.map(employee->(employee.getFirstName()) + " : "+

employee.getSalary() + " : Hike : "+

(employee.getSalary()+(15.00f/100.00f)\*employee.getSalary())

)

.forEach(System.out::println);

}

private static void getEmpHireOnSpecificDay(List<Employee> empList)

{

System.out.println("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)");

Scanner sc=new Scanner(System.in);

System.out.println(" Enter Day ");

String dayName=sc.next().toUpperCase();

empList.stream()

.filter(employee->employee.getHireDate().getDayOfWeek().equals(DayOfWeek.valueOf(dayName)))

.map(employee->(employee.getFirstName() ) + " : "+

(employee.getHireDate() ) + " : "+

(employee.getHireDate().getDayOfWeek())

)

.forEach(System.out::println);

}

private static void getEmpHireDetails(List<Employee> empList)

{

System.out.println("\*\*\*List employee name, hire date and day of week on which employee has started.\*\*\*\* ");

empList.stream()

.map(employee->(employee.getFirstName() ) + " : "+

(employee.getHireDate() ) + " : "+

(employee.getHireDate().getDayOfWeek())

).

forEach(System.out::println);

}

private static void getDeptWithHighestEmpCount(List<Employee> empList,List<Department>

deptList)

{

System.out.println("\*\*\*\*Get All Department With Emp Count\*\*\*\*\*\*");

Map<String, List<Employee>> employeeListByDept=

empList.stream()

.filter(e->e.getDepartment()!=null).

collect

(Collectors.groupingBy(employee->employee.getDepartment().getDepartmentName()));

employeeListByDept.forEach(

(dName,eList)->System.out.println(dName+" Has "+eList.size() +" Size "));

System.out.println("\*\*\*\*\*Get Department with Highest Emp Count\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

int maxEmpsize = employeeListByDept.keySet()

.stream()

.map(deptName->employeeListByDept.get(deptName).size())

.max(Integer::compare)

.get();

employeeListByDept.keySet().stream()

.filter(deptName->employeeListByDept.get(deptName).size() == maxEmpsize)

.forEach(System.out::println);

System.out.println(" Dept With Max Emp count is : "+maxEmpsize);

//for for each below is the conventional way

List<String> deptNameList=employeeListByDept.keySet().stream()

.filter(deptName->employeeListByDept.get(deptName).size() == maxEmpsize)

.collect(Collectors.toList());

for(String dept:deptNameList)

{

System.out.println(" Dept Name: "+dept);

}

}

private static void getDeptWithoutEmp(List<Employee> empList,List<Department> deptList)

{

System.out.println("\*\*\*\*Get Department Without Employee\*\*\*\*\*\*");

for(Department tempDept:deptList)

{

boolean empOpt=

empList.stream().

filter(employee->employee.getDepartment()!=null).

filter(employee->employee.getDepartment().getDepartmentId()==tempDept.getDepartmentId()).

findFirst().isPresent();

if(empOpt==false)

{

System.out.println(tempDept.getDepartmentName() +" Department do not have any employee ");

}

}

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

}

//Find departments with highest count of employees

private static void getEmpWithoutDept(List<Employee> empList)

{

System.out.println("\*\*\*\*Find out employees without department\*\*\*\*\*\*");

List<Employee> empWithoutDeptList=

empList.stream().

filter(employee->employee.getDepartment()==null).

collect(Collectors.toList());

System.out.println(empWithoutDeptList);

}

public static void sumOfSalOfAllEmp(List<Employee> empList)

{

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Get Sum Of sal Of All Employee\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

double sumOfSalOfAllEmp=0.0;

DoubleSummaryStatistics stats = empList.

stream().mapToDouble((employee) -> employee.getSalary()).

summaryStatistics();

System.out.println(" Total Sal Is : "+stats.getSum());

}

public static void getDeptWiseEmpList(List<Employee> empList)

{

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Find Department Wise Emp List\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Stream<Employee> newEmpList=empList.stream();

Stream<Employee> newEmpDeptList=newEmpList.filter(e->e.getDepartment()!=null);

Map<String, List<Employee>> employeeListByDept=

newEmpDeptList.collect(

Collectors.groupingBy(employee->employee.getDepartment().getDepartmentName()));

employeeListByDept.forEach(

(dName,eList)->System.out.println(dName+" Has "+eList.size() +" Size ")

);

}

public static void getSeniorEmp(List<Employee> empList)

{

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Find Senior Employee\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

LocalDate today = LocalDate.now();

Stream nameExpListStream =empList.stream().

map(employee->employee.getEmployeeId()+":"

+(Period.between(employee.getHireDate(),today ).getYears())

);

String namesExpList=(String)nameExpListStream.collect(Collectors.joining(", "));

System.out.println(namesExpList);

Optional<Employee> newExpEmpList=

empList.stream().max((e,e1)->(int)(

Period.between(e.getHireDate(),today ).toTotalMonths()-

Period.between(e1.getHireDate(),today ).toTotalMonths()));

System.out.println("Senior Most Emp Is : "+newExpEmpList.get());

}

public static void getManagerDetails(List<Employee> empList){

System.out.println("\*\*\*\*\*\*\*\*\*Employee And Their Manager Details\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

for(Employee tempEmp:empList)

{

if(tempEmp.getManagerId()==null)

{

System.out.println(tempEmp.getEmployeeId() + ":"+tempEmp.getFirstName()+" : No Manager");

}

else

{

Stream<Employee> empManagerListStream= empList.stream().

filter(e->e.getEmployeeId()==tempEmp.getManagerId());

Optional<Employee> empObj=empManagerListStream.findFirst();

Employee mgr=empObj.get();

System.out.println(tempEmp.getEmployeeId() + ": Report To :"+tempEmp.getFirstName()+" : "+mgr.getFirstName());

}

}

}

public static void getEmpDuration(List<Employee> empList)

{

System.out.println("\*\*\*\*\*List employee name and duration of their service in months and days\*\*\*\*\*");

LocalDate today=LocalDate.now();

List<String> empServiceDurationList=

empList.stream().

map(employee->employee.getFirstName() +" : "+

employee.getLastName() + " : " +

Period.between(employee.getHireDate(),today).getYears() + " Years : "+

Period.between(employee.getHireDate(),today).getMonths() + " Months : And "+

Period.between(employee.getHireDate(),today).getDays() + " Days \n"

).

collect(Collectors.toList());

System.out.println(empServiceDurationList);

}

}

**JAXB:**

**Exercise 1: Refer the above Employee case study. Export data of employees in XML format, store department wise employees in separate XML file with file name as name of department.**

**\*\*\*\*\*\*\*\*\*\*\*\*Department.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package com.igate.jaxb;

import javax.xml.bind.annotation.XmlAccessType;

import javax.xml.bind.annotation.XmlAccessorType;

import javax.xml.bind.annotation.XmlElement;

import javax.xml.bind.annotation.XmlType;

@XmlAccessorType(XmlAccessType.FIELD)

//@XmlType(name="Department",propOrder={"departmentId","departmentName","managerId"})

public class Department

{

@Override

public String toString()

{

return "Department [departmentId=" + departmentId + ", departmentName="

+ departmentName + ", managerId=" + managerId + "]";

}

@XmlElement(name="departmentId",required=true)

private int departmentId;

@XmlElement(name="departmentName",required=true)

private String departmentName;

@XmlElement(name="managerId",required=true)

private int managerId;

public Department(int departmentId, String departmentName, int managerId) {

this.departmentId = departmentId;

this.departmentName = departmentName;

this.managerId = managerId;

}

public int getDepartmentId() { return departmentId; }

public void setDepartmentId(int departmentId) {this.departmentId = departmentId; }

public String getDepartmentName() {return departmentName; }

public void setDepartmentName(String departmentName) {this.departmentName = departmentName; }

public int getManagerId() { return managerId; }

public void setManagerId(int managerId) { this.managerId = managerId; }

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Employee.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package com.igate.jaxb;

import java.time.LocalDate;

import javax.xml.bind.annotation.XmlAccessType;

import javax.xml.bind.annotation.XmlAccessorType;

import javax.xml.bind.annotation.XmlAttribute;

import javax.xml.bind.annotation.XmlElement;

import javax.xml.bind.annotation.XmlType;

@XmlAccessorType(XmlAccessType.FIELD)

public class Employee {

@Override

public String toString() {

return "Employee [employeeId=" + employeeId + ", firstName="

+ firstName + ", lastName=" + lastName + ", email=" + email

+ ", phoneNumber=" + phoneNumber + ", hireDate=" + hireDate

+ ", designation=" + designation + ", salary=" + salary

+ ", managerId=" + managerId + ", department=" + department

+ "]";

}

@XmlAttribute( name="empId",required=true) private Integer employeeId;

@XmlElement(name="firstName", required=true)

private String firstName;

@XmlElement(name="lastName")private String lastName;

@XmlElement(name="email") private String email;

@XmlElement(name="phoneNumber") private String phoneNumber;

@XmlElement(name="hireDate")

private LocalDate hireDate;

@XmlElement(name="designation")

private String designation;

@XmlElement(name="salary") private double salary;

@XmlElement(name="managerId") private Integer managerId;

private Department department;

public Employee(Integer employeeId, String firstName, String lastName,

String email, String phoneNumber, LocalDate hireDate,

String designation, double salary, Integer managerId,

Department department) {

super();

this.employeeId = employeeId;

this.firstName = firstName;

this.lastName = lastName;

this.email = email;

this.phoneNumber = phoneNumber;

this.hireDate = hireDate;

this.designation = designation;

this.salary = salary;

this.managerId = managerId;

this.department = department;

}

public Employee() { super(); }

public int getEmployeeId() { return employeeId; }

public void setEmployeeId(int employeeId) { this.employeeId = employeeId; }

public String getFirstName() { return firstName; }

public void setFirstName(String firstName) { this.firstName = firstName; }

public String getLastName() { return lastName; }

public void setLastName(String lastName) { this.lastName = lastName; }

public String getEmail() { return email; }

public void setEmail(String email) { this.email = email; }

public String getPhoneNumber() { return phoneNumber; }

public void setPhoneNumber(String phoneNumber) { this.phoneNumber = phoneNumber;}

public LocalDate getHireDate() { return hireDate; }

public void setHireDate(LocalDate hireDate) { this.hireDate = hireDate; }

public String getDesignation() { return designation; }

public void setDesignation(String designation) { this.designation = designation; }

public double getSalary() { return salary; }

public void setSalary(double salary) { this.salary = salary; }

public Integer getManagerId() { return managerId; }

public void setManagerId(Integer managerId) { this.managerId = managerId; }

public Department getDepartment() { return department; }

public void setDepartment(Department department) { this.department = department; }

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Employees.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package com.igate.jaxb;

import java.util.List;

import javax.xml.bind.annotation.XmlAccessType;

import javax.xml.bind.annotation.XmlAccessorType;

import javax.xml.bind.annotation.XmlElement;

import javax.xml.bind.annotation.XmlRootElement;

import javax.xml.bind.annotation.XmlType;

@XmlAccessorType(XmlAccessType.FIELD)

@XmlRootElement(name="emps",namespace="com.emp.vinsys")

@XmlType(name="empsType",propOrder={"empList"})

public class Employees

{

@Override

public String toString()

{

return "Employess [empList=" + empList + "]";

}

@XmlElement( name="emp",required=true)private List<Employee> empList;

public List<Employee> getEmpList() { return empList; }

public void setEmpList(List<Employee> empList){ this.empList = empList; }

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* TestEmpDeptMarshaller.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package com.igate.jaxb;

import java.io.File;

import java.time.LocalDate;

import java.time.Month;

import java.util.\*;

import java.util.stream.Collectors;

import java.util.stream.Stream;

import javax.xml.bind.JAXBContext;

import javax.xml.bind.Marshaller;

public class TestEmpDeptMarshaller

{

private static final String EMP\_XML = "IGateEmp.xml";

private static final String DEPT\_XML = "IGateDept.xml";

public static void main(String[] args) throws Exception

{

Employees emps=new Employees();

List<Employee> empList=EmployeeRepository.getEmployees();

List<Department> deptList=EmployeeRepository.getDepartments();

JAXBContext context = JAXBContext.newInstance(Employees.class);

Marshaller m = context.createMarshaller();

m.setProperty(Marshaller.JAXB\_FORMATTED\_OUTPUT, Boolean.TRUE);

List<Employee> eList=null;

String deptName=null;

for(Department deptTemp:deptList)

{

deptName=deptTemp.getDepartmentName();

int deptId=deptTemp.getDepartmentId();

System.out.println(deptId+ ": "+deptName);

eList=

empList.stream().

filter((employee)->(employee.getDepartment()!=null)

&&employee.getDepartment().getDepartmentId()==deptId).

collect(Collectors.toList());

System.out.println(eList);

emps.setEmpList(eList);

m.marshal(emps, new File(deptName+".xml"));

} }}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* TestEmpMarshaller.java\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package com.igate.jaxb;

import java.io.File;

import java.time.LocalDate;

import java.time.Month;

import java.util.\*;

import javax.xml.bind.JAXBContext;

import javax.xml.bind.Marshaller;

public class TestEmpMarshaller

{

private static final String EMP\_XML = "IGateEmp.xml";

private static final String DEPT\_XML = "IGateDept.xml";

public static void main(String[] args) throws Exception

{

Employees emps=new Employees();

List<Employee> empList=EmployeeRepository.getEmployees();

emps.setEmpList(empList);

// create JAXB context and instantiate marshaller

JAXBContext context = JAXBContext.newInstance(Employees.class);

Marshaller m = context.createMarshaller();

m.setProperty(Marshaller.JAXB\_FORMATTED\_OUTPUT, Boolean.TRUE);

//m.setProperty(Marshaller.JAXB\_ENCODING,Boolean.TRUE);

// m.setProperty(Marshaller.JAXB\_SCHEMA\_LOCATION,Boolean.FALSE);

// Write to System.out

m.marshal(emps, System.out);

// Write to File

m.marshal(emps, new File(EMP\_XML));

// get variables from our xml file, created before

System.out.println();

System.out.println("Output from our XML File: ");}}

**Exercise 2: Add new 3 departments to department’s collection from an XML file.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Department.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package com.igate.jaxb.unmarshal;

import javax.xml.bind.annotation.XmlAccessType;

import javax.xml.bind.annotation.XmlAccessorType;

import javax.xml.bind.annotation.XmlElement;

import javax.xml.bind.annotation.XmlType;

@XmlAccessorType(XmlAccessType.FIELD)

public class Department

{

public Department() {

super();

// TODO Auto-generated constructor stub

}

@Override

public String toString()

{

return "\nDepartment [departmentId=" + departmentId + ", departmentName="

+ departmentName + ", managerId=" + managerId + "]";

}

@XmlElement(name="departmentId",required=true)

private int departmentId;

@XmlElement(name="departmentName",required=true)

private String departmentName;

@XmlElement(name="managerId",required=true)

private int managerId;

public Department(int departmentId, String departmentName, int managerId) {

this.departmentId = departmentId;

this.departmentName = departmentName;

this.managerId = managerId;

}

public int getDepartmentId() {return departmentId; }

public void setDepartmentId(int departmentId) { this.departmentId = departmentId; }

public String getDepartmentName() { return departmentName; }

public void setDepartmentName(String departmentName) {this.departmentName = departmentName; }

public int getManagerId() { return managerId; }

public void setManagerId(int managerId) { this.managerId = managerId; }

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Depts.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package com.igate.jaxb.unmarshal;

import java.util.List;

import javax.xml.bind.annotation.XmlAccessType;

import javax.xml.bind.annotation.XmlAccessorType;

import javax.xml.bind.annotation.XmlElement;

import javax.xml.bind.annotation.XmlRootElement;

import javax.xml.bind.annotation.XmlType;

@XmlAccessorType(XmlAccessType.FIELD)

@XmlRootElement(name="depts")

public class Depts

{

public Depts() {

super();

// TODO Auto-generated constructor stub

}

@Override

public String toString() {

return "Depts [deptList=" + deptList + "]";

}

@XmlElement(name="dept",required=true)

private List<Department> deptList;

public List<Department> getDeptList()

{ return deptList; }

public void setDeptList(List<Department> deptList)

{ this.deptList = deptList; }

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* TestDeptUnMarshaller.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package com.igate.jaxb.unmarshal;

import java.io.File;

import java.util.\*;

import javax.xml.bind.JAXBContext;

import javax.xml.bind.Unmarshaller;

public class TestDeptUnMarshaller

{

public static void main(String[] args) throws Exception

{

//List<Department> deptList=EmployeeRepository.getDepartments();

// Creating an Unmarshaller

JAXBContext jaxbCtxt = JAXBContext.newInstance(Depts.class);

Unmarshaller jaxbUnmarshaller = jaxbCtxt.createUnmarshaller();

File ff=new File("depts.xml") ;

//Converting XML to java object using JAXB unmarshal API.

Depts depts = (Depts)jaxbUnmarshaller.unmarshal(ff) ;

System.out.println(depts.getDeptList());

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Thank You\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*