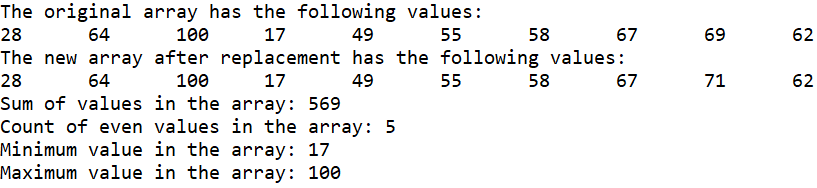
1



package com.tsl.common\_operations\_with\_array\_elements;

import java.util.Arrays;

import java.util.Random;

/\*\*

\* @author EMILIA BUTU

\* @version 1.0

\* @since 2020-07

\*

\* Student name: Tom Lever

\* Completion date: 05/17/21

\*

\* ArrayCommonOperations ecapsulates the entry point to a program to perform common operations on the elements of a

\* one-dimensional array.

\* The elements of the array are random numbers of type integer, with values between 1 and 100.

\* The calculated values are stored, and displayed at the end of the program.

\*/

public class ArrayCommonOperations {

/\*\*

\* main represents the entry point of the program.

\* main performs the common operations on the elements of a one-dimensional array.

\* @param args

\*/

public static void main(String [] args) {

//\*\*\* Task #1: Define and instantiate an array of integer numbers, with 10 elements

int[] theArrayOfIntegersWithTenElements = new int[10];

//\*\*\* Task #2: Fill in the array with integer numbers from 1 to 100

ARandomNumberGenerator theRandomNumberGenerator = new ARandomNumberGenerator();

int i;

for (i = 0; i < theArrayOfIntegersWithTenElements.length; i++) {

theArrayOfIntegersWithTenElements[i] = theRandomNumberGenerator.getARandomIntegerInclusivelyBetween(1, 100);

}

//\*\*\* Task #3: define another array, named temp, and copy the initial array in it.

//\* This allows to preserve the original array

int[] temp = theArrayOfIntegersWithTenElements.clone();

//\*\*\* Task #4: Define the variables you need to calculate the following values,

//\* and initialize them with appropriate values.

int theSumOfTheIntegersInTheArray = 0;

int theNumberOfEvenValuesInTheArray = 0;

int theMinimumValueInTheArray = 0;

int theMaximumValueInTheArray = 0;

//\*\*\* Task #5: Print the original array

System.*out*.println("The original array has the following values:");

for (i = 0; i < theArrayOfIntegersWithTenElements.length - 1; i++) {

System.*out*.print(theArrayOfIntegersWithTenElements[i] + "\t");

}

System.*out*.println(theArrayOfIntegersWithTenElements[theArrayOfIntegersWithTenElements.length - 1]);

//\*\*\* Task #6: Calculate the sum of all values

theSumOfTheIntegersInTheArray = Arrays.*stream*(theArrayOfIntegersWithTenElements).sum();

//\*\*\* Task #7: Count the number of even values

for (i = 0; i < theArrayOfIntegersWithTenElements.length; i++) {

if (MathUtilities.*isEven*(theArrayOfIntegersWithTenElements[i])) {

theNumberOfEvenValuesInTheArray++;

}

}

//\*\*\* Task #8: Calculate the minimum value in the array

theMinimumValueInTheArray = Arrays.*stream*(theArrayOfIntegersWithTenElements).min().getAsInt();

//\*\*\* Task #9: Calculate the maximum value in the array

theMaximumValueInTheArray = Arrays.*stream*(theArrayOfIntegersWithTenElements).max().getAsInt();

//\*\*\* Task #10: Replace the elements that are divisible by 3, with their value plus 2

for (i = 0; i < theArrayOfIntegersWithTenElements.length; i++) {

if (MathUtilities.*isDivisibleByThree*(theArrayOfIntegersWithTenElements[i])) {

temp[i] += 2;

}

}

//\*\*\* Task #11: Display the new array after replacement

System.*out*.println("The new array after replacement has the following values:");

for (i = 0; i < temp.length - 1; i++) {

System.*out*.print(temp[i] + "\t");

}

System.*out*.println(temp[temp.length - 1]);

//\*\*\* Task #12: Display the calculated values.

System.*out*.println("Sum of values in the array: " + theSumOfTheIntegersInTheArray);

System.*out*.println("Count of even values in the array: " + theNumberOfEvenValuesInTheArray);

System.*out*.println("Minimum value in the array: " + theMinimumValueInTheArray);

System.*out*.println("Maximum value in the array: " + theMaximumValueInTheArray);

}

}

/\*\*

\* ARandomNumberGenerator represents the structure for some random number generators.

\* @author Tom Lever

\* @version 1.0

\* @since 05/17/21

\*

\*/

class ARandomNumberGenerator {

/\*\*

\* thePopularRandomNumberGenerator is a component of ARandomNumberGenerator.

\*/

private Random thePopularRandomNumberGenerator;

/\*\*

\* ARandomNumberGenerator() is the constructor for ARandomNumberGenerator.

\*/

public ARandomNumberGenerator() {

thePopularRandomNumberGenerator = new Random();

}

/\*\*

\* getARandomIntegerInclusivelyBetween provides an integer between a lower limit and an upper limit inclusive.

\* @param theLowerLimit

\* @param theUpperLimit

\* @return

\*/

int getARandomIntegerInclusivelyBetween(int theLowerLimit, int theUpperLimit) {

return this.thePopularRandomNumberGenerator.nextInt((theUpperLimit - theLowerLimit) + 1) + theLowerLimit;

// throws IllegalArgumentException when lower limit is greater than upper limit.

}

}

/\*\*

\* MathUtilities encapsulates methods representing common mathematical operations.

\* @author Tom Lever

\* @version 1.0

\* @since 05/17/21

\*

\*/

class MathUtilities {

/\*\*

\* isEven indicates whether an integer is even or not.

\* @param theInteger

\* @return

\*/

public static boolean isEven(int theInteger) {

return (theInteger % 2 == 0);

}

/\*\*

\* isDivisibleByThree indicates whether an integer is divisible by 3 or not.

\* @param theInteger

\* @return

\*/

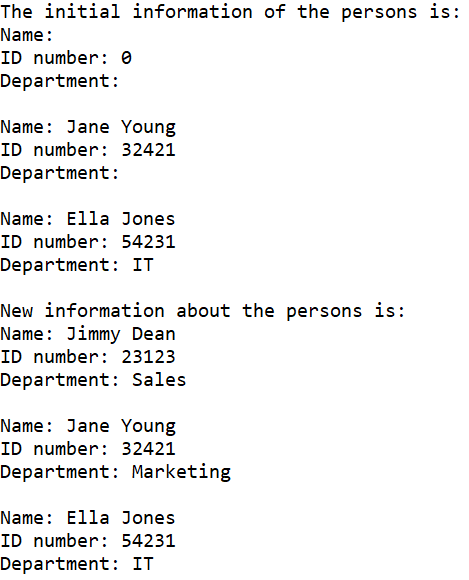
public static boolean isDivisibleByThree(int theInteger) {

return (theInteger % 3 == 0);

}

}

2



package com.tsl.person\_class;

/\*\*

\* @author EMILIA BUTU

\* version 1.0

\* since 2020-07

\*

\* Student name: Tom Lever

\* Completion date: 05/17/21

\*

\* PersonDriver.java

\*

\* This class represents the driver for class Person. The role of it is to test the methods of Person class.

\*/

public class PersonDriver {

/\*\*

\* main represents the entry point of the program.

\* main defines and instantiates three objects of type Person, displays information for the three objects, and

\* adds missing information using mutators.

\* @param args

\*/

public static void main(String[] args) {

//\*\*\* Task #1: define and instantiate three objects of type Person, using the three constructors

Person p1=new Person();

Person p2=new Person("Jane Young", 32421);

Person p3=new Person("Ella Jones",54231,"IT");

//\*\*\* Task #2: display the information of the three objects of type Person

System.*out*.println("The initial information of the persons is: ");

System.*out*.println(p1);

System.*out*.println();

System.*out*.println(p2);

System.*out*.println();

System.*out*.println(p3);

System.*out*.println();

//\*\*\* Task #3: add the missing information using mutators

p1.setName("Jimmy Dean");

p1.setIdNumber(23123);

p1.setDepartment("Sales");

p2.setDepartment("Marketing");

System.*out*.println("New information about the persons is: ");

System.*out*.println(p1);

System.*out*.println();

System.*out*.println(p2);

System.*out*.println();

System.*out*.println(p3);

System.*out*.println();

}

}

package com.tsl.person\_class;

/\*\*

\* @author EMILIA BUTU

\* version 1.0

\* since 2020-07

\*

\* Student name: Tom Lever

\* Completion date: 05/17/21

\*

\* Person represents the structure for a person.

\*/

class Person

{

//\*\*\* Task #1: define the instance variables

/\*\*

\* name is an attribute of a person.

\*/

private String name;

/\*\*

\* idNumber is an attribute of a person.

\*/

private int idNumber;

/\*\*

\* department is an attribute of a person.

\*/

private String department;

//\*\*\* Task #2: write the no-argument constructor

/\*\*

\* Person() is a zero-argument constructor for Person that initializes a person's name to "", the person's

\* idNumber to 0, and the person's department to "".

\*/

public Person() {

this.name = "";

this.idNumber = 0;

this.department = "";

}

//\*\*\* Task #3: write the constructor that passes values for the name and idNumber

/\*\*

\* Person(String theNameToUse, int theIdNumberToUse) is a two-argument constructor that initializes a person's

\* name to theNameToUse, the person's idNumber to theIdNumberToUse, and the person's department to "".

\* @param theNameToUse

\* @param theIdNumberToUse

\*/

public Person(String theNameToUse, int theIdNumberToUse) {

this.name = theNameToUse;

this.idNumber = theIdNumberToUse;

this.department = "";

}

//\*\*\* Task #4: write the constructor that initializes all three instance variables

/\*\*

\* Person(String theNameToUse, int theIdNumberToUse, String theDepartmentToUse) is a three-argument constructor

\* that initializes a person's name to theNameToUse, the person's idNumber to theIdNumberToUse, and the person's

\* department to theDepartmentToUse.

\* @param theNameToUse

\* @param theIdNumberToUse

\* @param theDepartmentToUse

\*/

public Person(String theNameToUse, int theIdNumberToUse, String theDepartmentToUse) {

this.name = theNameToUse;

this.idNumber = theIdNumberToUse;

this.department = theDepartmentToUse;

}

//\*\*\* Task #5: write accessor method for attribute name

/\*\*

\* getName provides the name of this person.

\* @return

\*/

public String getName() {

return this.name;

}

//\*\*\* Task #6: write mutator method for attribute name

/\*\*

\* setName sets the name of this person to a provided name.

\* @param theNameToUse

\*/

public void setName(String theNameToUse) {

this.name = theNameToUse;

}

//\*\*\* Task #7: write accessor method for attribute idNumber

/\*\*

\* getIdNumber provides the ID number of this person.

\* @return

\*/

public int getIdNumber() {

return this.idNumber;

}

//\*\*\* Task #8: write mutator method for attribute idNumber

/\*\*

\* setIdNumber sets the ID number of this person to a provided ID number.

\* @param theIdNumberToUse

\*/

public void setIdNumber(int theIdNumberToUse) {

this.idNumber = theIdNumberToUse;

}

//\*\*\* Task #9: write accessor method for attribute department

/\*\*

\* getDepartment provides the department of this person.

\* @return

\*/

public String getDepartment() {

return this.department;

}

//\*\*\* Task #10: write mutator method for attribute department

/\*\*

\* setDepartment sets the department of this person to a provided department.

\* @param theDepartmentToUse

\*/

public void setDepartment(String theDepartmentToUse) {

this.department = theDepartmentToUse;

}

//\*\*\* Task #11: write toString method

/\*\*

\* toString outputs information for this person.

\*/

@Override

public String toString() {

return

"Name: " + this.name + "\n" +

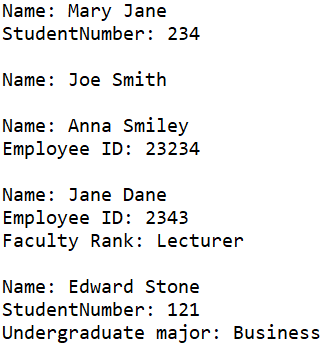
"ID number: " + this.idNumber + "\n" +

"Department: " + this.department;

}

}

3



Regarding how information for people is displayed, the program represented by the main method in the Staff class below outputs information for each person in a group of people. Considering our Java implementation, for each person, a String of information returned by the toString method associated with that person is output to the standard output stream. Every Java object is an instance of the Java Object class or a class that extends the Object class. The Object class has a public toString method. Every Java object inherits the toString method from Object or a toString method from the object’s immediate grantor that overrides the toString from Object. Information for any Java object may be acquired, and displayed, by explicitly or implicitly invoking that object’s toString method.

To display user-friendly information regarding a person, Person’s toString method overrides Object’s toString method, because Person extends Object and Person’s explicit toString method reimplements Person’s inherited toString method. To display information regarding a student, Student’s toString method overrides Person’s toString method. Student’s toString method has an expanded return statement relative to Person. To display information regarding an Undergraduate, Student’s toString method is overridden. Undergraduate’s toString method has an expanded return statement relative to Student. The same principle applies to Employee, which extends Person, and Faculty, which extends Employee.

package com.tsl.inheritance\_\_working\_with\_people;

/\*\*

\* @author EMILIA BUTU

\* version 1.0

\* since 2020-07

\*

\* Student name: Tom Lever

\* Completion date: 05/17/21

\*

\* Staff.java

\*

\* Driver for the inheritance application testing the inheritance, and

\* showing how polymorphism and dynamic binding works.

\*/

public class Staff {

/\*\*

\* main represents the entry point of the program.

\* main defines and instantiates a group of people and outputs information for each person in the group.

\* @param args

\*/

public static void main(String[] args) {

Person[] group = new Person[5];

group[0] = new Student("Mary Jane", 234);

group[1] = new Person("Joe Smith");

group[2] = new Employee("Anna Smiley", 23234);

group[3] = new Faculty("Jane Dane", 2343, "Lecturer");

group[4] = new Undergraduate("Edward Stone", 121, "Business");

for(Person p: group)

{

System.*out*.println(p);

System.*out*.println();

}

}

}

package com.tsl.inheritance\_\_working\_with\_people;

/\*\*

\* Person represents the structure for a person.

\* @author Tom Lever

\*

\*/

class Person {

/\*\*

\* name is an attribute of a person.

\*/

protected String name;

/\*\*

\* Person() is a zero-argument constructor for Person that initializes a person's name to "".

\*/

public Person() {

this.name = "";

}

/\*\*

\* Person(String name) is a one-argument constructor for Person that initializes a person's name to name.

\* @param name

\*/

public Person(String name) {

this.name = name;

}

/\*\*

\* getName provides the name of this person.

\* @return

\*/

public String getName() {

return this.name;

}

/\*\*

\* setName sets the name of this person to name.

\* @param name

\*/

public void setName(String name) {

this.name = name;

}

/\*\*

\* toString provides information for this person.

\*/

@Override

public String toString() {

return "Name: " + this.name;

}

}

package com.tsl.inheritance\_\_working\_with\_people;

/\*\*

\* Student represents the structure for a student.

\* @author Tom Lever

\*

\*/

class Student extends Person {

/\*\*

\* studentNumber is an attribute of a student.

\*/

protected int studentNumber;

/\*\*

\* Student() is a zero-argument constructor for Student that calls the one-argument constructor of Person with

\* argument "" and initializes a student's student number to 0.

\*/

public Student() {

//this.name = "";

super("");

this.studentNumber = 0;

}

/\*\*

\* Student(String name, int studentNumber) is a two-arguemtn constructor for Student that calls the one-argument

\* constructor of Person with argument name and initializes a student's student number to studentNumber.

\* @param name

\* @param studentNumber

\*/

public Student(String name, int studentNumber) {

//this.name = name;

super(name);

this.studentNumber = studentNumber;

}

/\*\*

\* getStudentNumber provides this student's student number.

\* @return

\*/

public int getStudentNumber() {

return this.studentNumber;

}

/\*\*

\* setStudentNumber sets this student's student number.

\* @param studentID

\*/

public void setStudentNumber(int studentID) {

this.studentNumber = studentID;

}

/\*\*

\* toString provides information for this student.

\*/

@Override

public String toString() {

return

"Name: " + this.name + "\n" +

"StudentNumber: " + this.studentNumber;

}

}

package com.tsl.inheritance\_\_working\_with\_people;

/\*\*

\* Employee represents the structure for an employee.

\* @author Tom

\*

\*/

class Employee extends Person {

/\*\*

\* empID is an attribute of an employee.

\*/

protected int empID;

/\*\*

\* Employee() is a zero-argument constructor for Employee that calls the one-argument constructor of Person with

\* argument "" and initializes empID to 0.

\*/

public Employee() {

//this.name = "";

super("");

this.empID = 0;

}

/\*\*

\* Employee(String name, int empID) is a two-argument constructor for Employee that calls the one-argument

\* constructor of Person with argument name and initializes empID to empID.

\* @param name

\* @param empID

\*/

public Employee(String name, int empID) {

//this.name = name;

super(name);

this.empID = empID;

}

/\*\*

\* getEmployeeID provides this employee's employee ID.

\* @return

\*/

public int getEmployeeID() {

return this.empID;

}

/\*\*

\* setEmployeeID sets this employee's employee ID.

\* @param employeeID

\*/

public void setEmployeeID(int employeeID) {

this.empID = employeeID;

}

/\*\*

\* toString provides information for this employee.

\*/

@Override

public String toString() {

return

"Name: " + this.name + "\n" +

"Employee ID: " + this.empID;

}

}

package com.tsl.inheritance\_\_working\_with\_people;

/\*\*

\* Undergraduate represents the structure for an undergraduate.

\* @author Tom

\*

\*/

public class Undergraduate extends Student {

/\*\*

\* major is an attribute of an undergraduate.

\*/

private String major;

/\*\*

\* Undergraduate() is a zero-argument constructor for Undergraduate that calls the two-argument constructor

\* of Student with arguments "" and 0 and initializes major to "".

\*/

public Undergraduate() {

//this.name = "";

//this.studentNumber = 0;

super("", 0);

this.major = "";

}

/\*\*

\* Undergraduate(String name, int studentNumber, String major) is a three-argument constructor for Undergraduate

\* that calls the two-argument constructor of Undergraduate with arguments name and studentNumber and initializes

\* major to major.

\* @param name

\* @param studentNumber

\* @param major

\*/

public Undergraduate(String name, int studentNumber, String major) {

//this.name = name;

//this.studentNumber = studentNumber;

super(name, studentNumber);

this.major = major;

}

/\*\*

\* getMajor provides this undergraduate's major.

\* @return

\*/

public String getMajor() {

return this.major;

}

/\*\*

\* setMajor sets this undergraduate's major.

\* @param major

\*/

public void setMajor(String major) {

this.major = major;

}

/\*\*

\* toString provides information for this undergraduate.

\*/

@Override

public String toString() {

return

"Name: " + this.name + "\n" +

"StudentNumber: " + this.studentNumber + "\n" +

"Undergraduate major: " + this.major;

}

}

package com.tsl.inheritance\_\_working\_with\_people;

/\*\*

\* Faculty represents the structure for a faculty member.

\* @author Tom

\*

\*/

class Faculty extends Employee {

/\*\*

\* rank is an attribute of a faculty member.

\*/

private String rank;

/\*\*

\* Faculty() is a zero-argument constructor for Faculty that calls the two-argument constructor of Employee

\* with arguments "" and 0 and initializes rank to "".

\*/

public Faculty() {

//this.name = "";

//this.empID = 0;

super("", 0);

this.rank = "";

}

/\*\*

\* Faculty(String name, int empID, String rank) is a three-argument constructor for Faculty that calls the

\* two-argument constructor of Employee with arguments name and empID and initializes rank to rank.

\* @param name

\* @param empID

\* @param rank

\*/

public Faculty(String name, int empID, String rank) {

//this.name = name;

//this.empID = empID;

super(name, empID);

this.rank = rank;

}

/\*\*

\* getRank provides this faculty member's rank.

\* @return

\*/

public String getRank() {

return this.rank;

}

/\*\*

\* setRank sets this faculty member's rank.

\* @param rank

\*/

public void setRank(String rank) {

this.rank = rank;

}

/\*\*

\* toString provides information for this faculty member.

\*/

@Override

public String toString() {

return

"Name: " + this.name + "\n" +

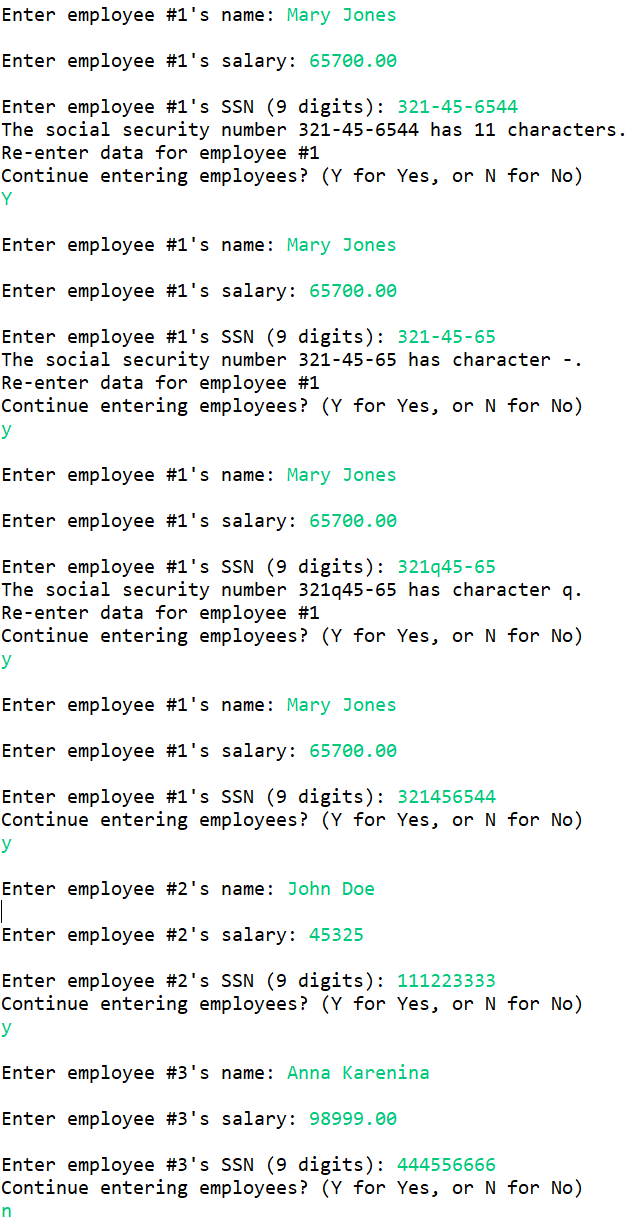
"Employee ID: " + this.empID + "\n" +

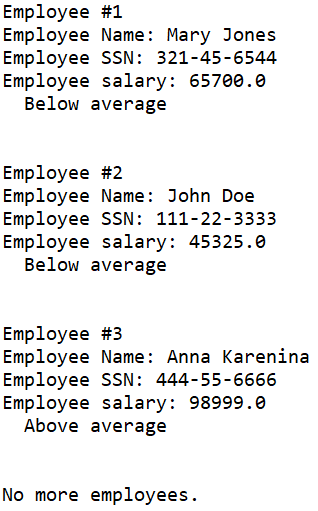
"Faculty Rank: " + this.rank;

}

}

**4**





**package** com.tsl.employee\_with\_exceptions;

**import** java.util.Scanner;

/\*\*

\* **@author** EMILIA BUTU

\* version 1.0

\* since 2020-07

\*

\* Student name: Tom Lever

\* Completion date: 05/18/21

\*

\* EmployeeExceptionDriver.java

\*

\* This class represents the driver for the EmployeeException class.

\* The driver program uses an array that can hold up to 100 employees

\* The array will be of EmployeeException type.

\* The user should be free to enter as many employees as needed (up to 100).

\* The driver class should use two exception classes to signal to the user that the SSN entered is not correct.

\* SSN needs to be entered as a 9-digit string without separators.

\*/

**public** **class** EmployeeExceptionDriver {

/\*\*

\* main represents the entry point of the program.

\* main invites a user to input name, salary, and SSN information for up to one hundred employees, and displays

\* information for all employees.

\* **@param** args

\*/

**public** **static** **void** main(String [] args) {

//\*\*\* Task #1: define the variables required for the program

EmployeeException[] e = **new** EmployeeException[100];

String empName;

String empSSN;

**double** empSalary;

**double** salarySum = 0;

**double** averageSalary;

String ignore;

//\*\*\* Task #2: define and instantiate variable of type Scanner to be able to read from keyboard

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

**char** repeat;

**int** i = 0; // Employee subscript: add 1 for employee number

//\*\*\* Task #3: create a loop in which you enter the data for employee.

**do** // Repeat if user says 'yes'

{

// Important note:

// Make a new Scanner object each time through the loop

// to avoid problems with mixing nextLine calls with

// other Scanner methods.

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

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

//\*\*\* Task #4: inside the loop, instantiate each element of the array with the constructor

// that has no arguments

e[i] = **new** EmployeeException();

//\*\*\* Task #5: read the name of the employee

System.***out***.print("Enter employee #" + (i + 1) + "'s name: ");

empName = keyboard.nextLine();

e[i].setName(empName);

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

//\*\*\* Task #6: read the salary of the employee

System.***out***.print("Enter employee #" + (i + 1) + "'s salary: ");

empSalary = keyboard.nextDouble();

e[i].setSalary(empSalary);

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

//\*\*\* Task #7: read SSN using the exceptions blocks

// check that the employee's ssn has 9 characters without separators

// check that the structure of the employee's ssn is correct

**try**

{

System.***out***.print("Enter employee #" + (i + 1) + "'s SSN (9 digits): ");

// Get rid of end of line left from nextDouble() call.

ignore = keyboard.nextLine();

empSSN = keyboard.nextLine();

**if**(empSSN.length() != 9) {

**throw** **new** SSNLengthException(empSSN, empSSN.length());

}

**else** {

**for**(**int** j = 0; j < 9; ++j) {

**if**((empSSN.charAt(j) > '9') || (empSSN.charAt(j) < '0')) {

**throw** **new** SSNCharacterException(empSSN, empSSN.charAt(j));

}

}

}

e[i].setSSN(empSSN);

++i; // Next employee

}

**catch**(SSNLengthException e1) {

System.***out***.println(e1.getMessage());

System.***out***.println("Re-enter data for employee #" + (i + 1));

}

**catch**(SSNCharacterException e2) {

System.***out***.println(e2.getMessage());

System.***out***.println("Re-enter data for employee #" + (i + 1));

}

//\*\*\* Task #8: ask the user if there are more employees to enter

System.***out***.println("Continue entering employees? (Y for Yes, or N for No)");

repeat = keyboard.next().charAt(0);

} **while**((repeat == 'y') || (repeat == 'Y'));

//\*\*\* Task #9: calculate the average salary

**for**(**int** j = 0; j < i; ++j) {

salarySum = salarySum + e[j].getSalary();

}

averageSalary = salarySum / i;

//\*\*\* Task #10: display the information about all employees with a note if their salary

// is above average, under average, or average.

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

**for**(**int** j = 0; j < i; ++j)

{

System.***out***.println("Employee #" + (j + 1));

e[j].writeOutput();

**if**(e[j].getSalary() > averageSalary)

System.***out***.println(" Above average");

**else** **if**(e[j].getSalary() < averageSalary)

System.***out***.println(" Below average");

**else**

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

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

}

System.***out***.println("No more employees.");

}

}

/\*\*

\* SSNLengthException represents the structure for objects of type SSNLengthException.

\* **@author** Tom

\*

\*/

**class** SSNLengthException **extends** Exception {

/\*\*

\* SSNLengthException() is a conventional zero-argument constructor for SSNLengthException, which calls Exception's

\* zero-argument constructor.

\*/

**public** SSNLengthException() {

**super**();

}

/\*\*

\* SSNLengthException(String theSocialSecurityNumber, int theLengthOfTheSocialSecurityNumber) is a two-argument

\* constructor for SSNLengthException, which builds an error message based on theSocialSecurityNumber and

\* theLengthOfTheSocialSecurityNumber and passes it to Exception's one-argument constructor with a message argument.

\* **@param** theSocialSecurityNumber

\* **@param** theLengthOfTheSocialSecurityNumber

\*/

**public** SSNLengthException(String theSocialSecurityNumber, **int** theLengthOfTheSocialSecurityNumber) {

**super**(

"The social security number " + theSocialSecurityNumber + " has " + theLengthOfTheSocialSecurityNumber +

" characters."

);

}

}

/\*\*

\* SSNCharacterException represents the structure for objects of type SSNCharacterException.

\* **@author** Tom

\*

\*/

**class** SSNCharacterException **extends** Exception {

/\*\*

\* SSNCharacterException() is a conventional zero-argument constructor for SSNCharacterException, which calls

\* Exception's zero-argument constructor.

\*/

**public** SSNCharacterException() {

**super**();

}

/\*\*

\* SSNCharacterException(String theSocialSecurityNumber, char theCharacterOfTheSocialSecurityNumber) is a

\* two-argument constructor for SSNCharacterException, which builds an error message based on

\* theSocialSecurityNumber and theCharacterOfTheSocialSecurityNumber and passes it to Exception's one-argument

\* constructor with a message argument.

\* **@param** theSocialSecurityNumber

\* **@param** theCharacterOfTheSocialSecurityNumber

\*/

**public** SSNCharacterException(String theSocialSecurityNumber, **char** theCharacterOfTheSocialSecurityNumber) {

**super**(

"The social security number " + theSocialSecurityNumber + " has character " +

theCharacterOfTheSocialSecurityNumber + "."

);

}

}

**package** com.tsl.employee\_with\_exceptions;

/\*\*

\* **@author** EMILIA BUTU

\* version 1.0

\* since 2020-07

\*

\* Student name: Tom Lever

\* Completion date: 05/18/21

\*

\* This class represents the blueprint for instantiating EmployeeException objects,

\* with the following attributes:

name: String

SSN: String

salary: double

\* and methods:

A constructor with no arguments that sets the attributes at default values

A constructor that passes values for all attributes

Accessor, mutator and display methods for each attribute

An equals method that has an object of type Employee as argument, and returns true if two employees have the same name, salary and sSN

\*/

**class** EmployeeException {

/\*\*

\* name is an attribute of an employee.

\*/

**private** String name;

/\*\*

\* SSN is an attribute of an employee.

\*/

**private** String SSN;

/\*\*

\* Salary is an attribute of an employee.

\*/

**private** **double** Salary;

/\*\*

\* EmployeeException is a zero-argument constructor for EmployeeException that initializes name to "", SSN to "",

\* and Salary to 0.

\*/

**public** EmployeeException() {

**this**.name = "";

**this**.SSN = "";

**this**.Salary = 0;

}

/\*\*

\* EmployeeException(String theNameToUse, String theSocialSecurityNumberToUse, double theSalaryToUse) is a three-

\* argument constructor for EmployeeException that initializes name to theNameToUse, SSN to

\* theSocialSecurityNumberToUse, and Salary to theSalaryToUse.

\* **@param** theNameToUse

\* **@param** theSocialSecurityNumberToUse

\* **@param** theSalaryToUse

\*/

**public** EmployeeException(String theNameToUse, String theSocialSecurityNumberToUse, **double** theSalaryToUse) {

**this**.name = theNameToUse;

**this**.SSN = theSocialSecurityNumberToUse;

**this**.Salary = theSalaryToUse;

}

/\*\*

\* getName provides this employee's name.

\* **@return**

\*/

**public** String getName() {

**return** **this**.name;

}

/\*\*

\* getSSN provides this employee's Social Security Number.

\* **@return**

\*/

**public** String getSSN() {

**return** **this**.SSN;

}

/\*\*

\* getSalary provides this employee's salary.

\* **@return**

\*/

**public** **double** getSalary() {

**return** **this**.Salary;

}

/\*\*

\* setName sets this employee's name.

\* **@param** theNameToUse

\*/

**public** **void** setName(String theNameToUse) {

**this**.name = theNameToUse;

}

/\*\*

\* setSSN sets this employee's Social Security Number.

\* **@param** theSocialSecurityNumberToUse

\*/

**public** **void** setSSN(String theSocialSecurityNumberToUse) {

**this**.SSN = theSocialSecurityNumberToUse;

}

/\*\*

\* setSalary sets this employee's salary.

\* **@param** theSalaryToUse

\*/

**public** **void** setSalary(**double** theSalaryToUse) {

**this**.Salary = theSalaryToUse;

}

/\*\*

\* writeOutName displays information relating to this employee's name.

\*/

**public** **void** writeOutName() {

System.***out***.println("Employee Name: " + **this**.name);

}

/\*\*

\* format provides a version of this employee's Social Security Number with hyphens.

\* **@param** theSocialSecurityNumber

\* **@return**

\*/

**public** String format(String theSocialSecurityNumber) {

**return**

**this**.SSN.charAt(0) +

**this**.SSN.charAt(1) +

**this**.SSN.charAt(2) +

"-" +

**this**.SSN.charAt(3) +

**this**.SSN.charAt(4) +

"-" +

**this**.SSN.charAt(5) +

**this**.SSN.charAt(6) +

**this**.SSN.charAt(7) +

**this**.SSN.charAt(8);

}

/\*\*

\* writeOutSSN displays information relating to this employee's Social Security Number.

\*/

**public** **void** writeOutSSN() {

System.***out***.println("Employee SSN: " + format(**this**.SSN));

}

/\*\*

\* writeOutSalary displays information relating to this employee's salary.

\*/

**public** **void** writeOutSalary() {

System.***out***.println("Employee salary: " + **this**.Salary);

}

/\*\*

\* writeOutput displays information relating to this employee.

\*/

**public** **void** writeOutput() {

System.***out***.println(

"Employee Name: " + **this**.name + "\n" +

"Employee SSN: " + format(**this**.SSN) + "\n" +

"Employee salary: " + **this**.Salary

);

}

/\*\*

\* equals provides information regarding whether all the attributes of this employee are equal to the corresponding

\* attributes of another employee or not.

\* **@param** theEmployeeExceptionToUse

\* **@return**

\*/

**public** **boolean** equals(EmployeeException theEmployeeExceptionToUse) {

**return** (

(**this**.name == theEmployeeExceptionToUse.name) &&

(**this**.SSN == theEmployeeExceptionToUse.SSN) &&

(**this**.Salary == theEmployeeExceptionToUse.Salary)

);

}

}