[Advanced Java Concepts](https://cognizant.tekstac.com/course/view.php?id=54" \l "section-6)

* [Java In-Depth: Core Java Made EasyUdemy](https://cognizant.tekstac.com/mod/udemy/view.php?id=845)

**Example: basic icon**

Go through the learnings mentioned below and implement all the Hands On explained by the author.

* + Section 23
  + Section 18 (133 - 140)
  + Section 20, 28, 31, 38
* [Java In-Depth: Become a Complete Java Engineer!Udemy](https://cognizant.tekstac.com/mod/udemy/view.php?id=1180)

**Example: basic icon**

Go through the learnings mentioned below and implement all the Hands On explained by the author.

* + Section 20
* [Java Database Connection: JDBC and MySQLUdemy](https://cognizant.tekstac.com/mod/udemy/view.php?id=1181)

**Example: basic icon**

Go through the entire course and implement all the Hands On explained by the author.

[Add Flight using JDBCCoding exercise](https://cognizant.tekstac.com/mod/vpl/view.php?id=847)

***Zaro Flight System***wants to automate the process in their organization.  As a start up, they need to automate the flight management system. Help them to develop this application.

You are provided with a public class Flight with following private attribute :

int flightId

String source

String destination

int noOfSeats

double flightFare

Appropriate setter and getter are written.

A public 5 argument constructor with arguments – flightId, source, destination, noOfSeats and flightFare is also provided.

Create a class FlightManagementSystem which has the following method.  Use Database for manipulation.

**public  boolean addFlight(Flight flightObj)** -  This method should accept a flight object and add that flight details into the database. If flight details are added successfully, return true. Else, return false.

To connect to the database you are provided with database.properties file and DB.java file.

The flight table is already created at the backend. The structure of flight table is:

|  |  |
| --- | --- |
| Column Name | Datatype |
| flightId | int |
| source | varchar2(30) |
| destination | varchar2(30) |
| noofseats | int |
| flightfare | number(8,2) |

Create a class Main which has main method to perform the above operation.

In main method,

When **addFlight** method is invoked and if added successfully, print “Flight details added successfully” else print “Addition not done”.

**To execute on your machine, you can make the necessary changes to the values of connection url, username and password in the database.properties  file.**

**Flight.java**

public class Flight {

private int flightId;

private String source;

private String destination;

private int noOfSeats;

private double flightFare;

public int getFlightId() {

return flightId;

}

public void setFlightId(int flightId) {

this.flightId = flightId;

}

public String getSource() {

return source;

}

public void setSource(String source) {

this.source = source;

}

public String getDestination() {

return destination;

}

public void setDestination(String destination) {

this.destination = destination;

}

public int getNoOfSeats() {

return noOfSeats;

}

public void setNoOfSeats(int noOfSeats) {

this.noOfSeats = noOfSeats;

}

public double getFlightFare() {

return flightFare;

}

public void setFlightFare(double flightFare) {

this.flightFare = flightFare;

}

public Flight(int flightId, String source, String destination,

int noOfSeats, double flightFare) {

super();

this.flightId = flightId;

this.source = source;

this.destination = destination;

this.noOfSeats = noOfSeats;

this.flightFare = flightFare;

}

}

**FlightManagementSystem.java**

import java.util.\*;

import java.sql.\*;

public class FlightManagementSystem{

public boolean addFlight(Flight flightObj){

boolean f=false;

PreparedStatement p=null;

try{

p=DB.getConnection().prepareStatement("insert into flight values(?,?,?,?,?)");

p.setInt(1,flightObj.getFlightId());

p.setString(2,flightObj.getSource());

p.setString(3,flightObj.getDestination());

p.setInt(4,flightObj.getNoOfSeats());

p.setDouble(5,flightObj.getFlightFare());

if(p.executeUpdate()==1)

{

f=true;

}

else{

f=false;

}

}catch(Exception e){

}

return f;

}

}

**DB.java**

import java.io.FileInputStream;

import java.io.IOException;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

import java.util.Properties;

public class DB {

private static Connection con = null;

private static Properties props = new Properties();

//ENSURE YOU DON'T CHANGE THE BELOW CODE WHEN YOU SUBMIT

public static Connection getConnection() throws ClassNotFoundException, SQLException {

try{

FileInputStream fis = null;

fis = new FileInputStream("database.properties");

props.load(fis);

// load the Driver Class

Class.forName(props.getProperty("DB\_DRIVER\_CLASS"));

// create the connection now

con = DriverManager.getConnection(props.getProperty("DB\_URL"),props.getProperty("DB\_USERNAME"),props.getProperty("DB\_PASSWORD"));

}

catch(IOException e){

e.printStackTrace();

}

return con;

}

}

**Database.properties**

#IF NEEDED, YOU CAN MODIFY THIS PROPERTY FILE

#ENSURE YOU ARE NOT CHANGING THE NAME OF THE PROPERTY

#YOU CAN CHANGE THE VALUE OF THE PROPERTY

#LOAD THE DETAILS OF DRIVER CLASS, URL, USERNAME AND PASSWORD IN DB.java using this properties file only.

#Do not hard code the values in DB.java.

DB\_DRIVER\_CLASS=com.mysql.jdbc.Driver

DB\_URL=jdbc:mysql://localhost:3306/${sys:DB\_USERNAME}

DB\_USERNAME=${sys:DB\_USERNAME}

DB\_PASSWORD=${sys:DB\_USERNAME}

**Main.java**

import java.util.\*;

import java.sql.\*;

public class Main{

public static void main(String[] args) throws ClassNotFoundException,SQLException {

Scanner sc=new Scanner(System.in);

int flightId=sc.nextInt();

sc.nextLine();

String s=sc.nextLine();

String d=sc.nextLine();

int ns=sc.nextInt();

sc.nextLine();

double fF=sc.nextDouble();

Flight f=new Flight(flightId,s,d,ns,fF);

FlightManagementSystem fm=new FlightManagementSystem();

if(fm.addFlight(f)){

System.out.println("Flight details added successfully");

}

else{

System.out.println("Addition not done");

}

}

}

[Mall Parking SystemCoding exercise](https://cognizant.tekstac.com/mod/vpl/view.php?id=848)

**Mall Parking System**

Westfield Shopping Mall is having a common two-wheeler parking stand in its basement. This parking facility is utilized by many of their customers regularly and they charge it for 10 Rupees per hour as parking fee. To make this facility more comfortable, they wish to automate the parking fee calculation based on the In-time and Out-time of a vehicle. For each vehicle entry, they will give a token which has the current date and time printed as In-time. The Out-time of each vehicle will the greater than the In-time. Help them to calculate the total Parking Fee by getting the In-time and Out-time as inputs.

**Note:**

* The input In-time, must be in dd/MM/yyyy HH:mm format and must be lesser than the current system time for at-least 1 Minute. Otherwise, print “<In-Time> is an Invalid In-Time” and terminate.
* The input Out-time, must be in dd/MM/yyyy HH:mm format and must be greater than the In-time for at-least 1 Minute. Otherwise, print “<Out-Time> is an Invalid Out-Time” and terminate.
* Output must be the calculated total parking fee based on difference of hours in between In-time and Out-time with 10 Rupees per hour as parking fee.

**Assumption:** Consider the current system date and time as 29/10/2019 20:10

**Sample Input 1:**

In-time

20/09/2019 23:55

Out-time

21/09/2019 23:56

**Sample Output 1:**

250 Rupees

**Sample Input 2:**

In-time

25/10/2019 03:05

Out-time

25/10/2019 03:06

**Sample Output 2:**

10 Rupees

**Sample Input 3:**

In-time

29/10/2019 20:10

**Sample Output 3:**

29/10/2019 20:10 is an Invalid In-Time

**Explanation:**   Invalid In-time since it is not lesser than the current system date and time  29/10/2019 20:10 for at-least 1 Minute

**Sample Input 4:**

In-time

20/09/2019 23:55

Out-time

20/08/2019 23:55

**Sample Output 4:**

20/08/2019 23:55 is an Invalid Out-Time

**Explanation:**   Invalid Out-time since it is not greater than the In-time for at-least 1 Minute

import java.util.Scanner;

import java.text.ParseException;

import java.util.Date;

import java.text.SimpleDateFormat;

public class Main {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

SimpleDateFormat format = new SimpleDateFormat("dd/MM/yyyy HH:mm");

Date current = null;

Date d1 = null;

Date d2 = null;

String InTime = "", OutTime = "", currenttime = "29/10/2019 20:10";

try {

System.out.println("In-time");

current = format.parse(currenttime);

InTime = sc.nextLine();

d1 = format.parse(InTime);

} catch (ParseException e) {

System.out.println(InTime + " is an Invalid In-Time");

System.exit(0);

}

long diff = current.getTime() - d1.getTime();

diff = diff / 60000;

if (diff >= 1) {

System.out.println("Out-time");

OutTime = sc.nextLine();

try {

d2 = format.parse(OutTime);

} catch (ParseException e) {

System.out.println(OutTime + " is an Invalid Out-Time");

System.exit(0);

}

diff = d2.getTime() - d1.getTime();

diff = diff / 60000;

if (diff >= 1) {

System.out.println(((diff / 60 + 1) \* 10) + " Rupees");

System.exit(0);

} else {

System.out.println(OutTime + " is an Invalid Out-Time");

System.exit(0);

}

} else {

System.out.println(InTime + " is an Invalid In-Time");

System.exit(0);

}

}

}

[Validate NameCoding exercise](https://cognizant.tekstac.com/mod/vpl/view.php?id=849)

## **Validate Name**

Create a public functional interface Validate with  a method :

**public boolean validateName(String name);**

Create a **public class ValidateUtility** with the below methods :

**public static Validate validateEmployeeName()** – The lambda expression  for the validateName  method must return true if the name is valid and return false if the name is invalid.

In this case, the name is valid if it contains alphabets and space and it should contain minimum 5 characters and maximum 20 characters.

**public static Validate validateProductName()** – The lambda expression  for the validateName  method must return true if the name is valid and return false if the name is invalid.

In this case, the name is valid if the first character is an alphabet followed by 5 digits.

Write the main method in ValidateUtility  class.

             -  Get the value for employee name and product name.

             -  Invoke the validateEmployeeName method

             -  Capture the object of Validate returned by the static method.

             -  Invoke the validateName method for the employee name received as input  from the user.

             -  Display the result as shown in sample output.

             -  Next, invoke the validateProductName method

             -  Capture the object of Validate returned by the static method.

             -  Invoke the validateName method for the product name received as input  from the user.

             -  Display the result as shown in sample output.

Note  :  Implement all the static methods mentioned above using lambda expression.  Don’t create object for the interface using new keyword.

**Sample Input 1 :**

Pinky Rose

A8546

**Sample Output 1 :**

Employee name is valid

Product name is invalid

**Sample Input 2 :**

Rahul@123

X82456

**Sample Output 1 :**

Employee name is invalid

Product name is valid

public interface Validate{

public boolean validateName(String name);

}

import java.util.Scanner;

import java.util.regex.Pattern;

public class ValidateUtility {

public static void main(String args[]) {

Scanner sc = new Scanner(System.in);

String empname, prodname;

boolean i=false,j=false;

empname = sc.nextLine();

prodname = sc.nextLine();

try {

i = validateEmployeeName().validateName(empname);

if (i)

System.out.println("Employee name is valid");

else

System.out.println("Employee name is invalid");

} catch (Exception e) {

System.out.println("Employee name is invalid");

}

try {

j = validateProductName().validateName(prodname);

if (j)

System.out.println("Product name is valid");

else

System.out.println("Product name is invalid");

} catch (Exception e) {

System.out.println("Product name is invalid");

}

}

public static Validate validateEmployeeName() {

Validate val = (String x) -> (Pattern.matches("[a-zA-Z[\\s]]{5,20}",x));

return val;

}

public static Validate validateProductName() {

Validate val = (String x) -> (Pattern.matches("[a-zA-Z]{1}\\d{5}", x));

return val;

}

}