ProductOrderDAO.Java

```
package com.cts.zepcpd.dao;

import java.sql.*;
import java.util.ArrayList;
import java.util.List;
import com.cts.zepcpd.exception.ProductOrderException;
import com.cts.zepcpd.util.*;
import com.cts.zepcpd.vo.ProductOrder;

public class ProductOrderDAO {


	public boolean addProductOrderDetails(List<ProductOrder> pdtOrder) throws
ProductOrderException {
		boolean recordsAdded = false;

		//Code here..
		Connection con = DBConnectionManager.getInstance().getConnection();
		PreparedStatement ps = null;
		try {
			// inserting values of list pdtOrders into database
			String query = "insert into ZEPC_MANAGE_ORDER (ORDER_ID,
PRODUCT_CODE, DATE_OF_ORDER, PRODUCT_LEVEL, DATE_OF_DELIVERY, NO_OF_PRODUCTS,
NO_OF_KMS_FOR_DELIVERY,MANAGER_APPROVAL, PRODUCT_COST, GST_TAX, DELIVERY_COST,
TOTAL_ORDER_COST, FINAL_STATUS_OF_ORDER)values(?,?,?,?,?,?,?,?,?,?,?,?,?)";
			for (ProductOrder e : pdtOrder) {
				ps = con.prepareStatement(query);
				ps.setString(1, e.getOrderId());
				ps.setString(2, e.getProductCode());
				ps.setDate(3,
ApplicationUtil.convertUtilToSqlDate(e.getDateOfOrder()));
				ps.setString(4, e.getProductLevel());
				ps.setDate(5,
ApplicationUtil.convertUtilToSqlDate(e.getDateOfDelivery()));
				ps.setInt(6, e.getNoOfProducts());
				ps.setDouble(7, e.getNoOfKmsForDelivery());
				ps.setString(8, e.getManagerApproval());
				ps.setDouble(9, e.getProductCost());
				ps.setDouble(10, e.getGstTax());
				ps.setDouble(11, e.getDeliveryCost());
				ps.setDouble(12, e.getTotalOrderCost());
				ps.setString(13, e.getFinalStatusOfOrder());
				int i = ps.executeUpdate();
				if (i > 0) {
					recordsAdded = true;
				} else {
					recordsAdded = false;
				}
			}

		} catch (SQLException e) {
			try {
				con.rollback();
			} catch (Exception e1) {
				e.printStackTrace();
			}
		} catch (Exception e) {
```

```
                e.printStackTrace();
                // throw new ProductOrderException("Database Value Insertion
Failed",
                // e.getCause());

            } finally {
                try {
                    ps.close();
                    con.close();
                } catch (Exception e) {
                    e.printStackTrace();
                    // throw new ProductOrderException("Database Value
Insertion Failed",
                    // e.getCause());
                }
            }



            return recordsAdded;
        }

        public List<ProductOrder> getAllProductOrderDetails() throws
ProductOrderException {
            List<ProductOrder> pdtOrder = new ArrayList<ProductOrder>();

            //Code here..
            // Retrieval of all records from database
            String query = "select * from ZEPC_MANAGE_ORDER";
            try (Connection con =
DBConnectionManager.getInstance().getConnection();
                    Statement st = con.createStatement();
                    ResultSet rs = st.executeQuery(query);) {
                while (rs.next()) {
                    // storing retrieved records in object
                    ProductOrder obj = new ProductOrder();
                    obj.setOrderId(rs.getString(1));
                    obj.setProductCode(rs.getString(2));
                    obj.setDateOfOrder(new
java.util.Date(rs.getDate(3).getTime()));
                    obj.setProductLevel(rs.getString(4));
                    obj.setDateOfDelivery(new
java.util.Date(rs.getDate(5).getTime()));
                    obj.setNoOfProducts(rs.getInt(6));
                    obj.setNoOfKmsForDelivery(rs.getDouble(7));
                    obj.setManagerApproval(rs.getString(8));
                    obj.setProductCost(rs.getDouble(9));
                    obj.setGstTax(rs.getDouble(10));
                    obj.setDeliveryCost(rs.getDouble(11));
                    obj.setTotalOrderCost(rs.getDouble(12));
                    obj.setFinalStatusOfOrder(rs.getString(13));
                    // adding ProductOrders object into arraylist
                    pdtOrder.add(obj);
                }

            } catch (Exception e) {
                e.printStackTrace();
                // throw new StudentAdmissionException("Database Value
Retrieval Failed",
```

```
                        // e.getCause());
                }


                return pdtOrder;

        }
}
```

ProductOrderException.Java

```
package com.cts.zepcpd.exception;

public class ProductOrderException extends Exception {

        private static final long serialVersionUID = -1105431869622052445L;

        /**
         * @param message
         * @param cause
         */
        public ProductOrderException(String message, Throwable cause) {
                super(message, cause);
        }
}
```

MainApp.Java

```
package com.cts.zepcpd.main;

import com.cts.zepcpd.service.*;
import com.cts.zepcpd.skeletonvalidator.SkeletonValidator;
import com.cts.zepcpd.util.*;

public class MainApp {

        private MainApp() {
        }


        public static void main(String[] args) {
                //Don't delete this code
                //Skeletonvalidaton starts
                new SkeletonValidator();
                //Skeletonvalidation ends
                //Write your code here..
                try {
                        ProductOrderService service = new ProductOrderService();

        System.out.println(service.addProductOrderDetails("inputFeed.txt"));
                        System.out.println(service.searchProductOrder("R005"));
                        } catch (Exception e) {
                        e.printStackTrace();


                        }
```

```
            // List<StudentAdmission> studentAdmissionList =
            //
service.buildStudentAdmissionsList(ApplicationUtil.readFile("inputFeed.txt"));
            /*
            * for(StudentAdmission e:studentAdmissionList) {
System.out.println(e); }
            */
            }


      }
```

ProductOrderService.Java

```
package com.cts.zepcpd.service;

import java.util.ArrayList;
import java.util.List;

import com.cts.zepcpd.dao.*;
import com.cts.zepcpd.exception.ProductOrderException;
import com.cts.zepcpd.util.*;
import com.cts.zepcpd.vo.ProductOrder;

public class ProductOrderService {


      /**
       * @param productOrderRecords
       * @return List<ProductOrder>
       */
      public static List<ProductOrder> buildProductOrdersList(List<String>
productOrderRecords) {
            List<ProductOrder> productOrderList = new ArrayList<ProductOrder>();

            //Code here..
            for (String e : productOrderRecords) {
                  String res[] = e.split(",");
                  String orderId = res[0];
                  String productCode = res[1];
                  String dateOfOrder = res[2];   //DateType
                  String productLevel = res[3];
                  String dateOfDelivery = res[4];   //DateType
                  String  noOfProducts= res[5];
                  String noOfKmsForDelivery = res[6];
                  String managerApproval = res[7];
                  ProductOrder obj = new ProductOrder();
                  obj.setOrderId(orderId);
                  obj.setProductCode(productCode);

                  // converting String to java.uti.Date

      obj.setDateOfOrder(ApplicationUtil.convertStringToDate(dateOfOrder));
                  obj.setProductLevel(productLevel);

                  // converting String to java.uti.Date

      obj.setDateOfDelivery(ApplicationUtil.convertStringToDate(dateOfDelivery));
                  obj.setNoOfProducts(Integer.parseInt(noOfProducts));
```

```
	obj.setNoOfKmsForDelivery(Double.parseDouble(noOfKmsForDelivery));
			obj.setManagerApproval(managerApproval);
			double[] productOrderCosts =
					calculateTotalOrderCost(
Integer.parseInt(noOfProducts),

	Double.parseDouble(noOfKmsForDelivery),productLevel );

			obj.setProductCost(productOrderCosts[0]);
			obj.setGstTax(productOrderCosts[1]);
			obj.setDeliveryCost(productOrderCosts[2]);
			obj.setTotalOrderCost(productOrderCosts[3]);
			obj.setFinalStatusOfOrder("OrderSuccessfull");
			productOrderList.add(obj);
			}
			return productOrderList ;
			}



	public boolean addProductOrderDetails(String inputFeed) throws
ProductOrderException {
		List<ProductOrder> productOrderList = ProductOrderService

	.buildProductOrdersList(ApplicationUtil.readFile(inputFeed));
				ProductOrderDAO stdDao = new ProductOrderDAO();
				return stdDao.addProductOrderDetails(productOrderList);
		//Code here..

		 //TODO change this return value
	}

	public static double[] calculateTotalOrderCost(int noOfProducts, double
noOfKmsForDelivery, String productLevel) {
		double[] productOrderCosts = new double[4];

			if ("Level01".equals(productLevel)) {
				productOrderCosts[0] = 500;
				productOrderCosts[1] = 2;
				productOrderCosts[2] = 8;
			} else if ("Level02".equals(productLevel)) {
				productOrderCosts[0] = 600;
				productOrderCosts[1] = 3;
				productOrderCosts[2] = 10;
			} else if ("Level03".equals(productLevel)) {
				productOrderCosts[0] = 800;
				productOrderCosts[1] = 5;
				productOrderCosts[2] = 11;
			} else if ("Level04".equals(productLevel)) {
				productOrderCosts[0] = 1200;
				productOrderCosts[1] = 7;
				productOrderCosts[2] = 13;
			} else if ("Level05".equals(productLevel)) {
				productOrderCosts[0] = 1750;
				productOrderCosts[1] = 8;
				productOrderCosts[2] = 14;
			} else if ("Level06".equals(productLevel)) {
				productOrderCosts[0] = 2500;
```

```
                        productOrderCosts[1] = 9;
                        productOrderCosts[2] = 14;
                }
                productOrderCosts[0] =
Math.round(productOrderCosts[0]*noOfProducts*100.0)/100.0;
                productOrderCosts[1] =  Math.round((productOrderCosts[1] *
productOrderCosts[0])/100*100.0)/100.0;
                productOrderCosts[2] =
Math.round(productOrderCosts[2]*noOfKmsForDelivery*100.0)/100.0;
                productOrderCosts[3] =
productOrderCosts[0]+productOrderCosts[1]+productOrderCosts[2];



                return productOrderCosts;

        }

        public boolean searchProductOrder(String orderId) throws
ProductOrderException {
                boolean status = false;

                //Code here..
                ProductOrderDAO ptdDao = new ProductOrderDAO();
                List<ProductOrder> ptdOrders = ptdDao.getAllProductOrderDetails();
                for (ProductOrder e : ptdOrders ) {
                        if (e.getOrderId().equals(orderId)) {
                                status = true;
                                System.out.println(e);
                                break;
                        }
                        else {
                                System.out.println("Order Request not Found");
                        }
                }

                return status;
        }
}
```

ApplicationUtil.Java

```
package com.cts.zepcpd.util;


import java.io.*;
import java.text.*;
import java.util.*;


import com.cts.zepcpd.exception.ProductOrderException;

public class ApplicationUtil {
```

```java
	/**
	 * @param fileName
	 * @return List<String>
	 * @throws ProductOrderException
	 */
	public static List<String> readFile(String fileName) throws
ProductOrderException {
		List<String> productOrderList = new ArrayList<String>();
		//Code Here
		FileReader fr = null;
		BufferedReader br = null;
		try {
			fr = new FileReader(fileName);
			br = new BufferedReader(fr);
			String line = null;
			while ((line = br.readLine()) != null) {
				String[] res = line.split(",");
				String managerApproval = res[7];
				Date dtOfOrder = convertStringToDate(res[2]);
				Date dtOfDelivery = convertStringToDate(res[4]);
				if (checkIfValidOrder(dtOfOrder,dtOfDelivery,
				managerApproval)) {
					productOrderList.add(line);
				}
			}
		} catch (Exception e) {
				e.printStackTrace();
		}

		return productOrderList;
	}



	/**
	 * @param util
	 *            Date
	 * @return sql Date
	 */
	public static java.sql.Date convertUtilToSqlDate(java.util.Date uDate) {
		java.sql.Date sDate = new java.sql.Date(uDate.getTime());

		//Code here..

		return sDate;
	}

	/**
	 * @param inDate
	 * @return Date
	 */
	public static Date convertStringToDate(String inDate) {
		//Code here..
		try {
			SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd",
Locale.ENGLISH);
			return format.parse(inDate);
		} catch (Exception e) {
			e.printStackTrace();
```

```
                return null;
            }


        }

        public static boolean checkIfValidOrder(Date dtOfOrder, Date dtOfDelivery,
String manager) {
            boolean orderValidity = false;

            //Code here..
            if ("Approved".equals(manager)&& ((dtOfDelivery.getTime() -
dtOfOrder.getTime()) / (1000 * 60 * 60 * 24)) % 365 >=7) {
                    orderValidity = true;
            }
            return orderValidity;

        }
}
```

DBConnectionManager.Java

* Don't change this code

*/

```
package com.cts.zepcpd.util;


import java.io.FileInputStream;

import java.io.FileNotFoundException;

import java.io.IOException;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

import java.util.Properties;


import com.cts.zepcpd.exception.ProductOrderException;



public class DBConnectionManager {


                private static DBConnectionManager instance;

                public static final String PROPERTY_FILE = "database.properties";
```

```
public static final String DRIVER = "drivername";
public static final String URL = "url";
public static final String USER_NAME = "username";
public static final String PASSWORD = "password";

private static Connection connection = null;
private static Properties props = null;

/**
 * @throws ProductOrderException
 */
private DBConnectionManager() throws ProductOrderException {
	loadProperties();
	try {
		Class.forName(props.getProperty(DRIVER));
		this.connection =
DriverManager.getConnection(props.getProperty(URL), props.getProperty(USER_NAME),
					props.getProperty(PASSWORD));
	} catch (ClassNotFoundException ex) {
		throw new ProductOrderException("Could not find Driver
class ", ex.getCause());
	} catch (SQLException e) {
		throw new ProductOrderException("Database Connection
Creation Failed", e.getCause());
	}
}

/**
 * @return Connection
 */
public Connection getConnection() {
	return connection;
```

```
                }

                /**
                 * @return DBConnectionManager
                 * @throws ProductOrderException
                 */
                public static DBConnectionManager getInstance() throws
ProductOrderException {

    // Code here
                        instance = new DBConnectionManager();
                        return instance;

                }

                /**
                 * @throws ProductOrderException
                 */
                private void loadProperties() throws ProductOrderException {
                        FileInputStream inputStream = null;
                        try {
                                inputStream = new FileInputStream(PROPERTY_FILE);
                                props = new Properties();
                                props.load(inputStream);
                        } catch (FileNotFoundException e) {
                                throw new ProductOrderException("Database Property File
Not Found", e.getCause());
                        } catch (IOException e) {
                                throw new ProductOrderException("Exception during
property file I/O", e.getCause());
                        } finally {
                                if (inputStream != null) {
```

```
                                                try {
                                                        inputStream.close();
                                                } catch (IOException e) {
                                                        throw new
ProductOrderException("Exception during property file I/O", e.getCause());
                                                }
                                        }
                                }
                        }
}
```

ProductOrder.Java

```
* Don't change this code
 */
package com.cts.zepcpd.vo;

import java.util.Date;

public class ProductOrder {

                        String orderId;
                        String productCode;
                        Date dateOfOrder;
                        String productLevel;
                        Date dateOfDelivery;
                        int noOfProducts;
                        double noOfKmsForDelivery;
                        String managerApproval;
                        double productCost;
                        double gstTax;
                        double deliveryCost;
                        double totalOrderCost;
```

```
	String finalStatusOfOrder;


	public ProductOrder() {
		super();
	}


	public ProductOrder(String orderId, String productCode, Date dateOfOrder,
String productLevel, Date dateOfDelivery,
			int noOfProducts, double noOfKmsForDelivery, String
managerApproval, double productCost, double gstTax,
			double deliveryCost, double totalOrderCost, String
finalStatusOfOrder) {
		super();
		this.orderId = orderId;
		this.productCode = productCode;
		this.dateOfOrder = dateOfOrder;
		this.productLevel = productLevel;
		this.dateOfDelivery = dateOfDelivery;
		this.noOfProducts = noOfProducts;
		this.noOfKmsForDelivery = noOfKmsForDelivery;
		this.managerApproval = managerApproval;
		this.productCost = productCost;
		this.gstTax = gstTax;
		this.deliveryCost = deliveryCost;
		this.totalOrderCost = totalOrderCost;
		this.finalStatusOfOrder = finalStatusOfOrder;
	}


	public String getOrderId() {
		return orderId;
	}
```

```
public void setOrderId(String orderId) {
	this.orderId = orderId;
}

public String getProductCode() {
	return productCode;
}

public void setProductCode(String productCode) {
	this.productCode = productCode;
}

public Date getDateOfOrder() {
	return dateOfOrder;
}

public void setDateOfOrder(Date dateOfOrder) {
	this.dateOfOrder = dateOfOrder;
}

public String getProductLevel() {
	return productLevel;
}
```

```
public void setProductLevel(String productLevel) {
	this.productLevel = productLevel;
}

public Date getDateOfDelivery() {
	return dateOfDelivery;
}

public void setDateOfDelivery(Date dateOfDelivery) {
	this.dateOfDelivery = dateOfDelivery;
}

public int getNoOfProducts() {
	return noOfProducts;
}

public void setNoOfProducts(int noOfProducts) {
	this.noOfProducts = noOfProducts;
}

public double getNoOfKmsForDelivery() {
	return noOfKmsForDelivery;
}
```

```
public void setNoOfKmsForDelivery(double noOfKmsForDelivery) {
        this.noOfKmsForDelivery = noOfKmsForDelivery;
}


public String getManagerApproval() {
        return managerApproval;
}


public void setManagerApproval(String managerApproval) {
        this.managerApproval = managerApproval;
}


public double getProductCost() {
        return productCost;
}


public void setProductCost(double productCost) {
        this.productCost = productCost;
}


public double getGstTax() {
        return gstTax;
}


public void setGstTax(double gstTax) {
```

```
        this.gstTax = gstTax;
}

public double getDeliveryCost() {
        return deliveryCost;
}

public void setDeliveryCost(double deliveryCost) {
        this.deliveryCost = deliveryCost;
}

public double getTotalOrderCost() {
        return totalOrderCost;
}

public void setTotalOrderCost(double totalOrderCost) {
        this.totalOrderCost = totalOrderCost;
}

public String getFinalStatusOfOrder() {
        return finalStatusOfOrder;
}

public void setFinalStatusOfOrder(String finalStatusOfOrder) {
        this.finalStatusOfOrder = finalStatusOfOrder;
```

```
		}



		@Override

		public String toString() {

			return "ZepcManagementOrder Details: [orderId=" + orderId + ",
productCode=" + productCode + ", dateOfOrder="

					+ dateOfOrder + ", productLevel=" + productLevel +
", dateOfDelivery=" + dateOfDelivery + ", noOfProducts="

					+ noOfProducts + ", noOfKmsForDelivery=" +
noOfKmsForDelivery + ", managerApproval=" + managerApproval

					+ ", productCost=" + productCost + ", gstTax=" +
gstTax + ", deliveryCost=" + deliveryCost + ", totalOrderCost=" + totalOrderCost

					+ ", finalStatusOfOrder=" + finalStatusOfOrder + "]";

		}


}
```

SkeletonValidator.Java

```
package com.cts.zepcpd.skeletonvalidator;


import java.lang.reflect.Array;

import java.lang.reflect.Method;

import java.util.logging.Level;

import java.util.logging.Logger;


/**

 * @author t-aarti3

 *	This class is used to verify if the Code Skeleton is intact and not

 *	modified by participants thereby ensuring smooth auto evaluation

 * */
```

```
public class SkeletonValidator {
                public SkeletonValidator() {
                        validateClassName("com.cts.zepcpd.util.DBConnectionManager");
                        validateClassName("com.cts.zepcpd.util.ApplicationUtil");
                        validateClassName("com.cts.zepcpd.service.ProductOrderService");
                        validateClassName("com.cts.zepcpd.dao.ProductOrderDAO");
                        validateClassName("com.cts.zepcpd.vo.ProductOrder");

                validateClassName("com.cts.zepcpd.exception.ProductOrderException");



                        validateMethodSignature(

                "addProductOrderDetails:boolean,getAllProductOrderDetails:List",
                                        "com.cts.zepcpd.dao.ProductOrderDAO");
                        validateMethodSignature(

                "buildProductOrdersList:List,addProductOrderDetails:boolean,calculateTotal
OrderCost:double[],searchProductOrder:boolean",
                                        "com.cts.zepcpd.service.ProductOrderService");
                        validateMethodSignature(

                "readFile:List,convertUtilToSqlDate:Date,convertStringToDate:Date,checkIfV
alidOrder:boolean",
                                        "com.cts.zepcpd.util.ApplicationUtil");
                        validateMethodSignature(

                "getConnection:Connection,getInstance:DBConnectionManager",
                                        "com.cts.zepcpd.util.DBConnectionManager");



                }
```

```
		private static final Logger LOG = Logger.getLogger("SkeletonValidator");
		protected final boolean validateClassName(String className) {

			boolean iscorrect = false;
			try {
				Class.forName(className);
				iscorrect = true;
				LOG.info("Class Name " + className + " is correct");

			} catch (ClassNotFoundException e) {
				LOG.log(Level.SEVERE, "You have changed either the " +
"class name/package. Use the correct package "
						+ "and class name as provided in the
skeleton");

			} catch (Exception e) {
				LOG.log(Level.SEVERE,
						"There is an error in validating the " + "Class
Name. Please manually verify that the "
							+ "Class name is same as
skeleton before uploading");
			}
			return iscorrect;
		}

		protected final void validateMethodSignature(String methodWithExcptn,
String className) {
			Class cls = null;
			try {

				String[] actualmethods = methodWithExcptn.split(",");
				boolean errorFlag = false;
				String[] methodSignature;
```

```
String methodName = null;
String returnType = null;

for (String singleMethod : actualmethods) {
	boolean foundMethod = false;
	methodSignature = singleMethod.split(":");

	methodName = methodSignature[0];
	returnType = methodSignature[1];
	cls = Class.forName(className);
	Method[] methods = cls.getMethods();
	for (Method findMethod : methods) {
		if
(methodName.equals(findMethod.getName())) {
			foundMethod = true;
			if
(!(findMethod.getReturnType().getSimpleName().equals(returnType))) {
				errorFlag = true;
				LOG.log(Level.SEVERE, " You
have changed the " + "return type in '" + methodName
					+ "'
method. Please stick to the " + "skeleton provided");

			} else {
				LOG.info("Method signature
of " + methodName + " is valid");
			}

		}
	}
	if (!foundMethod) {
		errorFlag = true;
```

```
                                                    LOG.log(Level.SEVERE, " Unable to find the
given public method " + methodName
                                                                + ". Do not change the " +
"given public method name. " + "Verify it with the skeleton");
                                                }


                                            }
                                            if (!errorFlag) {
                                                LOG.info("Method signature is valid");
                                            }


                                        } catch (Exception e) {
                                            LOG.log(Level.SEVERE,
                                                    " There is an error in validating the " +
"method structure. Please manually verify that the "
                                                            + "Method signature is
same as the skeleton before uploading");
                                        }
                                    }


}
```