nunsys®
Tu socio tecnológico

Cuestionario

Selección de personal

CONTENIDO

INTRODUCCIÓN

El objetivo de este documento es el de ampliar y complementar la información que ya conocemos de ti a partir de tu curriculum vitae. Para ello, hemos preparado este cuestionario con el que queremos comprobar en qué medida tus conocimientos de programación se alinean con el perfil profesional que requerimos.

Las respuestas a este cuestionario nos las tendrás que redactar en un documento aparte y nos las tendrás que remitir vía email.

NOTA: Todo el código de ejemplo esta en java

CUESTIÓN 1

Indica que problemas o malas prácticas de programación hay en el siguiente código.

```
public class Customer {

    private CustomerDAO customerDAO;
    private int customerID;
    private String name;
    private String email;

    public Customer(CustomerDAO customerDAO) {
        this.customerDAO = customerDAO;
    }

    public String GetName() {
        name = name.toUpperCase();
        return name;
    }

    public String GetEmail() {
        email = email.toLowerCase();
        return email;
    }

    public void SetName(String name) {
        this.name = name;
    }

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

    public void save() {
        customerDAO.save(this);
    }
}


public interface CustomerDAO {
    Customer findById(int customerID);
    List<Customer> findAll();
    void save(Customer customer);
}
```

CUESTIÓN 2

Indica que problemas o malas prácticas de programación hay en el siguiente código.

```
import org.apache.log4j.LogManager;
import org.apache.log4j.Logger;

public class Main {
    private static final Logger logger = LogManager.getLogger(Main.class);

    public static void main(String[] args) {
        Main main = new Main();
        main.execute(args);
    }

    private void execute(String[] args) {
        try {
            validateArgs(args);
            doWork(args[0]);
        } catch (Exception e) {
            logger.error("Invalid parameters", e);
        }
    }

    private void validateArgs(String[] args) throws
                    NumberOfParametersException, ParameterFormatException {

        if (args.length == 1) {
            if (!isValidArgument(args[0]))
            {
                throw new ParameterFormatException();
            }
        } else {
            throw new NumberOfParametersException();
        }
    }

    private boolean isValidArgument(String arg) {
        return arg.length() == 5;
    }

    private void doWork(String idContract) {
        System.out.println(idContract);
    }

}
```

CUESTIÓN 3

Indica que problemas o malas prácticas de programación hay en el siguiente código.

```
public class OrderProcessor {

    public void process(Order order) {
        if (order.isValid() && this.save(order))
        {
            this.sendConfirmationMessage(order);
        }
    }

    private boolean save(Order order) {
        boolean result = false;

        // TODO Save order in database

        return result;
    }

    private void sendConfirmationMessage(Order order) {
        String name = order.getCustomer().getName();
        String email = order.getCustomer().getEmail();

        // TODO send email

    }

}
```

NOTA: Se supone que donde estan los comentario TODO, se implementará en un caso el código necesario para acceder a la base de datos y realizar las operaciones de persistencia necesarias, en el otro caso se implementarán las instrucciones necesarias para realizar el envío de email. No se incluye este código por simplicidad del ejercicio.

CUESTIÓN 4

Indica que problemas o malas prácticas de programación hay en el siguiente código.

```
public class OtherOrderProcessor {

    public void process(Order order) {
        OrderRepository repository = new OrderRepository();

        if (order.isValid() && repository.save(order))
        {
            EmailNotifier notifier = new EmailNotifier();
            notifier.sendConfirmationMessage(order);
        }
    }
}
```

CUESTIÓN 5

Indica que problemas o malas prácticas de programación hay en el siguiente código.

```
public class CustomerRepository {
    private static CustomerRepository INSTANCE = null;

    private CustomerRepository(){}

    private static void createInstance() {
        if (INSTANCE == null) {
            INSTANCE = new CustomerRepository();
        }
    }

    public static CustomerRepository getInstance() {
        if (INSTANCE == null) createInstance();
        return INSTANCE;
    }
}
```

CUESTIÓN 6

Indica que problemas o malas prácticas de programación hay en el siguiente código.

```
import junit.framework.TestCase;
import org.apache.log4j.LogManager;
import org.apache.log4j.Logger;

class OrderProcessorTest extends TestCase {
    private static final Logger logger =
LogManager.getLogger(OrderProcessorTest.class);

    void testProcess() {

        OrderProcessor orderProcessor = new OrderProcessor();

        Customer customer = new Customer();
        customer.SetEmail("customer@email.com");
        customer.SetName("Peter Smith");

        Order order = new Order();
        order.setOrderId(1);
        order.setValid(true);
        order.setCustomer(customer);


        try {
            orderProcessor.process(order);
        } catch (Exception) {
            logger.debug("Order not processed");
        }

    }

}
```

CUESTIÓN 7

Indica que problemas o malas prácticas de programación hay en el siguiente código.

```
public class Payroll {

    public Money calculatePay(Employee e) throws
InvalidEmployeeTypeException {
        // Enter in fuction CalculatePay
        switch (e.type) {
            case COMMISSIONED:
                // Calculate commissioned pay
                return calculateCommissionedPay(e);
            case HOURLY:
                // Calculate hourly pay
                return calculateHourlyPay(e);
//              case MONTHLY:
//                  // Calculate monthly pay
//                  return calculateMonthlyPay(e);
            case SALARIED:
                // Calculate salaried pay
                return calculateSalariedPay(e);
            default:
                // throw InvalidEmployeeTypeException
                throw new InvalidEmployeeTypeException(e.type);
        }
    }

    private Money calculateSalariedPay(Employee e) {
        // TODO - Calculate salaried pay
        return null;
    }

    private Money calculateHourlyPay(Employee e) {
        // TODO - Calculate hourly pay
        return null;
    }

    private Money calculateCommissionedPay(Employee e) {
        // TODO - Calculate commissioned pay
        return null;
    }

}
```

CUESTIÓN 8

Indica que problemas o malas prácticas de programación hay en el siguiente código.

```
public class Rectangle {

    private int x1;
    private int y1;
    private int x2;
    private int y2;

    public Rectangle(int x1, int y1, int x2, int y2) {
        this.x1 = x1;
        this.y1 = y1;
        this.x2 = x2;
        this.y2 = y2;
    }

    public boolean contains(int x1, int y1, int x2, int y2){
        return (this.x1 < x1 && y1 < y1 &&
                this.x1 + x2 - x1 > this.x2  &&
                y1 + y1 - y2 > this.y2);
    }

}
```

CUESTIÓN 9

Indica que problemas o malas prácticas de programación hay en el siguiente código.

```
public enum EnumResponseType {
    RESPONSE_OK,
    RESPONSE_READING_ERROR,
    RESPONSE_DEVICE_ERROR
}

public interface QRDevice {

    EnumResponseType open();
    EnumResponseType readData();
    String getData();
    EnumResponseType close();

}
```

CUESTIÓN 10

Indica que problemas o malas prácticas de programación hay en el siguiente código.

```
import java.util.List;

public class Project {

    private List<ProjectFile> projectFileList;

    public List<ProjectFile> getProjectFileList() {
        return projectFileList;
    }

    public void setProjectFileList(List<ProjectFile> projectFileList) {
        this.projectFileList = projectFileList;
    }

    public void loadAllFiles() {
        for (ProjectFile file: projectFileList) {
            file.loadFileData();
        }
    }

    public void saveAllFiles() throws InvalidOperationException {
        for (ProjectFile file: projectFileList) {
            if (!(file instanceof ReadOnlyFile)) {
                file.saveFileData();
            }
        }

    }

}
```

```
public class ProjectFile {

    private String filePath;
    private byte[] fileData;

    public String getFilePath() {
        return filePath;
    }

    public void setFilePath(String filePath) {
        this.filePath = filePath;
    }

    public byte[] getFileData() {
        return fileData;
    }

    public void setFileData(byte[] fileData) {
        this.fileData = fileData;
    }

    public void loadFileData() {
        // TODO Retrieve FileData from disk
    }

    public void saveFileData() throws InvalidOperationException {
        // TODO Write FileData to disk
    }
}


public class ReadOnlyFile extends ProjectFile {

    public void saveFileData() throws InvalidOperationException {
        throw new InvalidOperationException();
    }

}
```