**Proyecto Curso: Programación Orientada a Objetos (UPC)**

**Administración Hotelera**

**Integrantes:**

Carlos Ríos Rosales (u201100198)

Yolvi Jhoseth Jimenez Anampa (u201216326)

Jonatán Aarón Pizarro Rojas (u201100617)

Gino Yampiere Adrianzen Espinoza (u200921172)

# Presentación

La Programación Orientada a Objetos es un paradigma de programación que usa objetos y sus interacciones, para diseñar aplicaciones y programas informáticos.

Está basado en diferentes técnicas, incluyendo herencia, abstracción, polimorfismo y encapsulamiento; así como también involucra las buenas prácticas de programación, la que genera que se pueda reutilizar código y finalmente esto se traduce en menor esfuerzo, menor código y tiempo de desarrollo.

# Índice

Contenido

[Presentación 2](#_Toc326072073)

[Índice 3](#_Toc326072074)

[Introducción 4](#_Toc326072075)

[Caso Propuesto 4](#_Toc326072076)

[Historia de Usuario 4](#_Toc326072077)

[Pruebas Unitarias 5](#_Toc326072078)

[Administración de Cuenta Test: AdmCuentaTest. 5](#_Toc326072079)

[Administración de Número de Habitaciones Test: AdmNumHabitacionesTest 7](#_Toc326072079)

[Codigo Fuente 7](#_Toc326072084)

[Clases 7](#_Toc326072085)

[Conclusiones 8](#_Toc326072086)

# Introducción

En este caso práctico aplicaremos los conceptos de la Programación Orientada a Objetos para la definición y diseño de un proyecto software, utilizando JAVA como lenguaje de programación.

Este proyecto debe reflejar en el desarrollo las historias de usuario solicitadas con sus respectivos Tests y documentación, los cuales deberán ser subidos al GitHub.

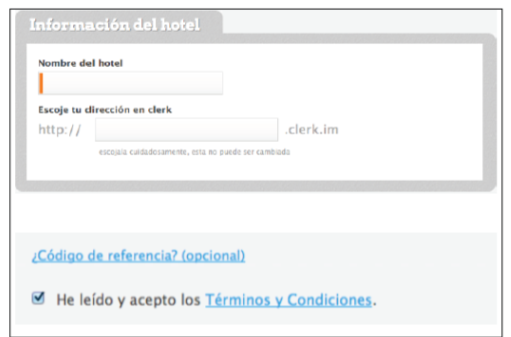
# Caso Propuesto

En el presente trabajo nos toca desarrollar un caso práctico de Administración Hotelera. Dicho trabajo deberá ser dividido en dos entregables.

# Historia de Usuario

Para la primera entrega la historia de usuario a presentar es: **Configurar Cuenta en el Sistema.**





|  |  |
| --- | --- |
| **Criterios de Aceptación** | |
| ***Cuando*** | ***Espero*** |
| * Habiendo ingresado mi información personal (correo electrónico y contraseña) y la información del hotel (nombre y dirección que tendré en el sistema hotelero), además de haber leído y aceptado los Términos y Condiciones. | * Ver la pantalla de proceso finalizado indicando que mi cuenta ha sido creada exitosamente e indicando que tengo 24 horas para confirmar el e-mail recibido del sistema hotelero. Se indica además que se tiene el plan gratuito hasta que se cambie a un plan con costo. |
| * No ingreso los campos solicitados o aceptado los Términos y condiciones. | * Un mensaje de error por cada campo no ingresado. |
| * Cuando ingreso el nombre del hotel | * Espero hasta 5 recomendaciones sobre el nombre. Una vez seleccionado o ingresado el nombre, la dirección web del hotel se genera anteponiéndose http://<<nombre>>.clerk.im |
| **Test de Aceptación** | |
| * Si el correo electrónico es incorrecto se indica que mensaje indicando “Debe ingresar correo electrónico valido”. * Si el correo electrónico no se ingresa se debe mostrar un mensaje indicando “Debe ingresar correo electrónico”. * Si no se ingresa la contraseña dos veces se indicará el mensaje “Debe ingresar contraseña”. * Si las dos contraseñas no coinciden se indicará el mensaje “Las contraseñas no coinciden”. * Si no se ingresa el nombre se indicará el siguiente mensaje “Debe ingresar nombre de hotel”. * Si ingresa un nombre de hotel se deberá mostrar una lista de nombres sugeridos (hasta 5) validos y la dirección web debe ser http://<nombre>.clerk.im | |

# Pruebas Unitarias

**Administración de Cuenta: AdmCuentaTest**

@Test

public void siPasoCampoCorreoVacioMeDebeDarError() {

assertFalse(admCuentas.verificaCadena(null, "Correo Electrónico"));

assertFalse(admCuentas.verificaCadena(null, "Contraseña"));

assertFalse(admCuentas.verificaCadena(null, "Nombre de Hotel"));

}

@Test

public void siPasoCadenaNoVaciaMeDebeDarVerdad() {

assertTrue(admCuentas.verificaCadena("verdad@verdad.com", "Correo"));

assertTrue(admCuentas.verificaCadena("Verdad001", "Clave"));

assertTrue(admCuentas.verificaCadena("La Verdad", "Hotel"));

System.out.println("Correcto. El campo no esta vacío");

}

@Test

public void siPasoUnCorreoErradoMeDebeDarError() {

assertFalse(admCuentas.verificaCorreo("jaime@mail.com."));

}

@Test

public void siPasoUnCorreoCorrectoMeDebeDarVerdad() {

assertTrue(admCuentas.verificaCorreo("crios@ddperu.com.pe"));

}

@Test

public void siComparoDosCadenasClavesDiferentesMeDebeDarError() {

assertFalse(admCuentas.comparaClaves("master001", "Master001"));

}

@Test

public void siComparoDosCadenasClavesIgualesMeDebeDarVerdad() {

assertTrue(admCuentas.comparaClaves("master001", "master001"));

System.out.println("Las Claves son iguales.");

}

@Test

public void siPasoElNombreDelHotelMeDebeDarSuDireccionWeb() {

String cadenaGenerada = admCuentas.generaDirClerk("El Holandes Errante");

String cadenaComparar = "http://elholandeserrante.clerk.im";

assertEquals(cadenaGenerada, cadenaComparar);

System.out.println("La Dirección Web es: " + cadenaGenerada);

}

@Test

public void siNoPasoElCheckMeDebeDarError() {

assertFalse(admCuentas.verificaCheck('0'));

}

@Test

public void siPasoElCheckMeDebeDarVerdad() {

assertTrue(admCuentas.verificaCheck('1'));

System.out.println("Check correcto.");

}

@Test

public void siPasoUnaCadenaNombreHotelMeDebeRetornarUnaListaDeSugerencias() {

ArrayList<Hotel> lista = new ArrayList<Hotel>();

String nomRetornado = "", nomEsperado = "San Blas", nomAsignado = "San";

lista = admCuentas.listaHoteles(nomAsignado);

for (Hotel hotel : lista) {

nomRetornado = hotel.getNombre().trim();

System.out.println(hotel.getNombre());

}

// El esperado es el último de la lista.

assertEquals(nomEsperado, nomRetornado);

}

@Test

public void siPasoUnaCuentaExistenteMeDebeDarError() {

assertFalse(admCuentas.buscaCuenta("jpalacios@gmail.com"));

System.out.println("Cuenta de correo ya existe.");

}

@Test

public void siPasoUnaCuentaNoExistenteMeDebeDarVerdad() {

assertTrue(admCuentas.buscaCuenta("mcorrea@celtik.com.ar"));

System.out.println("Correcto, Cuenta no existe.");

}

@Test

public void siRegistroUnaCuentaExistenteMeDebeDarError() {

assertFalse(admCuentas.escribirCuenta(new Cuenta("jpalacios@gmail.com", "juan002", "Sheraton", "juanpalacios", "1")));

}

@Test

public void siRegistroUnaCuentaNoExistenteMeDebeDarVerdad() {

// Cada vez que se quiera correr el Test se debe cambiar de datos para que así sea un registro nuevo.

assertTrue(admCuentas.escribirCuenta(new Cuenta("pcabrera@hotmail.com", "ana001", "El Olivar", "http://elolivar.clerk.com", "1")));

}

**Administración de Cuenta: AdmNumHabitacionesTest**

@Test

public void siPasoUnNumeroMenorQueUnoMeDebeDarError() {

assertFalse(admNumHabitaciones.verificaNumero(0));

}

@Test

public void siPasoUnNumeroMayorQueUnoMeDebeDarVerdad() {

assertTrue(admNumHabitaciones.verificaNumero(5));

System.out.println("Correcto");

}

# Código Fuente

El código en su totalidad se encuentra en el repositorio del grupo en el GitHub.

# Clases

**Cuenta**

correo String - dirección de correo electrónico

clave String - contraseña de la cuenta

nomHotel String - nombre del hotel

dirClerk String - dirección web del hotel en clerk

check String - check de aceptación de términos y condiciones.

**Hotel**

nombre String - nombre del hotel

# Conclusiones

En esta parte del proyecto y según lo desarrollado hasta el momento, nos damos cuenta que la POO, a diferencia de otros paradigmas, nos facilita la forma de desarrollar software. Esto quiere decir que nos simplifica la codificación haciéndolo más escalable y fácil. Es cierto que esto tiene sus ventajas, pero para poder aprovecharlo debemos tener pasión por lo que hacemos y mucha práctica, además de aprender el trabajo en equipo, lo cual es fundamental para el desarrollo de aplicaciones grandes.

Por último, cabe mencionar que para entender cómo funciona el paradigma de la programación orientada a objetos es necesario ver un programa como una colección de objetos que interactúan entre sí enviándose mensajes y cambiando su **estado** durante la ejecución.

Resolver un problema bajo el paradigma de la programación orientada a objetos implica determinar y caracterizar los diferentes objetos que intervienen en el problema, definir sus propiedades y métodos y ponerlos a interactuar.