DISEÑO DE EXPERIMENTOS DE ISW (SI732)
PRACTICA CALIFICADA 1
2024-02

Sección: SW71, SW72, WS72, WS73, WV71, WX72, WX74
Profesores: Tinoco Licas, Juan Carlos
Noriega Melendez, Julio Manuel
Ramirez Argume, Leo Carlos Israel

Duración: 110 min

Indicaciones:

1. El examen consta de 2 preguntas, y tendrá 110 minutos para resolverlas.
2. Las preguntas son tipo procedimental y la entrega de su respuesta es a través de envío de **archivo {código_alumno}.zip** y archivo de Word adjunto. Utilice el documento de Word upc-pre-202402-si732-pc1-WS72-file.docx para responder, tanto a nivel de las preguntas relacionadas con implementación y evidencias. Coloque contenido en cada página de respuesta según el título e indicaciones.
3. Ante problemas técnicos, debe de forma obligatoria adjuntar evidencias del mismo, como capturas de pantalla, videos, fotos, etc. Siendo requisito fundamental que, en cada evidencia se pueda apreciar claramente la fecha y hora del sistema operativo del computador donde el alumno está rindiendo el examen.
4. Está permitido consultar material del curso y manuales pero no utilizar IA.
5. El código se presenta en Java y C#. El alumno puede escoger el lenguaje de programación de su dominio y las herramientas para hacer las pruebas.
6. De identificarse dos soluciones exactamente iguales de parte de dos o más alumnos, la prueba se calificará desaprobatoria para todos ellos.

Enunciado:

Se le presenta el siguiente código que es el extracto de un proyecto de una veterinaria. El proyecto se presenta en Java y en C++.

Ud. Es libre de agregar las líneas de código que reflejen historias de usuario o reglas de negocio a validar o puede tomar estas líneas. Si es que agrega líneas de código, la lógica tiene que ver con el proyecto que se muestra.

Código en Java

```java
import java.time.LocalDateTime;
import java.time.temporal.ChronoUnit;
import java.util.List;

class Dueno {
    private String nombre;
    private String apellido;
    private String telefono;
    private String direccion;

    public Dueno(String nombre, String apellido, String telefono, String direccion) {
        this.nombre = nombre;
        this.apellido = apellido;
```

```
        this.telefono = telefono;
        this.direccion = direccion;
    }

    // Getters y setters
}

class Mascota {
    private String nombre;
    private String especie;
    private String raza;
    private int edad;
    private Dueno dueno;
    private Clinica clinica;

    public Mascota(String nombre, String especie, String raza, int edad, Dueno dueno, Clinica
clinica) {
        this.nombre = nombre;
        this.especie = especie;
        this.raza = raza;
        this.edad = edad;
        this.dueno = dueno;
        this.clinica = clinica;
    }

    public String getNombre() {
        return nombre;
    }

    // Getters y setters
}

class Clinica {
    private String nombre;
    private String direccion;
    private String telefono;
    private List<Mascota> mascotas;

    public Clinica(String nombre, String direccion, String telefono) {
        this.nombre = nombre;
        this.direccion = direccion;
        this.telefono = telefono;
    }

    // Getters y setters
}

class Cita {
    private Mascota mascota;
    private LocalDateTime fecha;
    private String motivo;
    private String estado; // "Reservada", "Cancelada", "Cerrada"
    private LocalDateTime fechaReserva;
    private LocalDateTime fechaCancelacion;
    private LocalDateTime fechaCierre;

    public Cita(Mascota mascota, LocalDateTime fecha, String motivo) {
        this.mascota = mascota;
        this.fecha = fecha;
        this.motivo = motivo;
        this.estado = "Reservada";
        this.fechaReserva = LocalDateTime.now();
        System.out.println("Cita reservada para " + mascota.getNombre() + " el " + fecha + " por
motivo: " + motivo + ". Fecha de reserva: " + fechaReserva);
    }

    public void reservar(LocalDateTime fechaReserva) {
        if (ChronoUnit.MINUTES.between(LocalDateTime.now(), this.fecha) >= 30) {
            this.estado = "Reservada";
            this.fechaReserva = fechaReserva;
            System.out.println("La cita ha sido reservada para " + mascota.getNombre() + " en " +
fechaReserva + ".");
        } else {
```

```
            System.out.println("No se puede reservar la cita. Debe hacerse al menos 30 minutos
antes de la fecha programada.");
        }
    }

    public void cancelar(LocalDateTime fechaCancelacion) {
        if (ChronoUnit.HOURS.between(LocalDateTime.now(), this.fecha) >= 3) {
            this.estado = "Cancelada";
            this.fechaCancelacion = fechaCancelacion;
            System.out.println("La cita ha sido cancelada para " + mascota.getNombre() + " en " +
fechaCancelacion + ".");
        } else {
            System.out.println("No se puede cancelar la cita. Debe hacerse al menos 3 horas antes
de la fecha programada.");
        }
    }

    public void cerrar(LocalDateTime fechaCierre) {
        this.estado = "Cerrada";
        this.fechaCierre = fechaCierre;
        System.out.println("La cita ha sido cerrada para " + mascota.getNombre() + " en " +
fechaCierre + ".");
    }

    // Getters y setters
}

// Ejemplo de uso
public class Main {
    public static void main(String[] args) {
        Dueno dueno = new Dueno("Juan", "Pérez", "123456789", "Calle Principal 123");
        Mascota mascota = new Mascota("Firulais", "Perro", "Labrador", 5, dueno, null);
        Clinica clinica = new Clinica("Clínica Veterinaria", "Avenida Central 456", "987654321");

        Cita cita = new Cita(mascota, LocalDateTime.of(2024, 9, 15, 10, 0), "Vacuna anual");

        // Intentar reservar la cita
        cita.reservar(LocalDateTime.now());

        // Intentar cancelar la cita
        cita.cancelar(LocalDateTime.now());

        // Cerrar la cita ejecutada
        cita.cerrar(LocalDateTime.now());
    }
}
```

Código en C#

```
using System;
using System.Collections.Generic;
using System.Linq;

class Dueno
{
    public string nombre { get; set; }
    public string apellido { get; set; }
    public string telefono { get; set; }
    public string direccion { get; set; }

    public Dueno(string nombre, string apellido, string telefono, string direccion)
    {
        this.nombre = nombre;
        this.apellido = apellido;
        this.telefono = telefono;
        this.direccion = direccion;
    }
}

class Mascota
{
    public string nombre { get; set; }
```

```
    public string especie { get; set; }
    public string raza { get; set; }
    public int edad { get; set; }
    public Dueno dueno { get; set; }
    public Clinica clinica { get; set; }

    public Mascota(string nombre, string especie, string raza, int edad, Dueno dueno, Clinica
clinica)
    {
        this.nombre = nombre;
        this.especie = especie;
        this.raza = raza;
        this.edad = edad;
        this.dueno = dueno;
        this.clinica = clinica;
    }
}

class Clinica
{
    public string nombre { get; set; }
    public string direccion { get; set; }
    public string telefono { get; set; }
    public List<Mascota> mascotas { get; set; }

    public Clinica(string nombre, string direccion, string telefono)
    {
        this.nombre = nombre;
        this.direccion = direccion;
        this.telefono = telefono;
    }
}

class Cita
{
    public Mascota mascota { get; set; }
    public DateTime fecha { get; set; }
    public string motivo { get; set; }
    public string estado { get; set; } // "Reservada", "Cancelada", "Cerrada"
    public DateTime fechaReserva { get; set; }
    public DateTime fechaCancelacion { get; set; }
    public DateTime fechaCierre { get; set; }

    public Cita(Mascota mascota, DateTime fecha, string motivo)
    {
        this.mascota = mascota;
        this.fecha = fecha;
        this.motivo = motivo;
        this.estado = "Reservada";
        this.fechaReserva = DateTime.Now;
        Console.WriteLine($"Cita reservada para {mascota.nombre} el {fecha} por motivo: {motivo}.
Fecha de reserva: {fechaReserva}");
    }

    public void reservar(DateTime fechaReserva)
    {
        if ((this.fecha - fechaReserva).TotalMinutes >= 30)
        {
            this.estado = "Reservada";
            this.fechaReserva = fechaReserva;
            Console.WriteLine($"La cita ha sido reservada para {mascota.nombre} en
{fechaReserva}.");
        }
        else
        {
            Console.WriteLine("No se puede reservar la cita. Debe hacerse al menos 30 minutos
antes de la fecha programada.");
        }
    }

    public void cancelar(DateTime fechaCancelacion)
    {
        if ((this.fecha - fechaCancelacion).TotalHours >= 3)
```

```
        {
            this.estado = "Cancelada";
            this.fechaCancelacion = fechaCancelacion;
            Console.WriteLine($"La cita ha sido cancelada para {mascota.nombre} en
{fechaCancelacion}.");
        }
        else
        {
            Console.WriteLine("No se puede cancelar la cita. Debe hacerse al menos 3 horas antes
de la fecha programada.");
        }
    }

    public void cerrar(DateTime fechaCierre)
    {
        this.estado = "Cerrada";
        this.fechaCierre = fechaCierre;
        Console.WriteLine($"La cita ha sido cerrada para {mascota.nombre} en {fechaCierre}.");
    }
}

class Program
{
    static void Main(string[] args)
    {
        Dueno dueno = new Dueno("Juan", "Pérez", "123456789", "Calle Principal 123");
        Mascota mascota = new Mascota("Firulais", "Perro", "Labrador", 5, dueno, null);
        Clinica clinica = new Clinica("Clínica Veterinaria", "Avenida Central 456", "987654321");

        Cita cita = new Cita(mascota, new DateTime(2024, 9, 15, 10, 0, 0), "Vacuna anual");

        // Intentar reservar la cita
        cita.reservar(DateTime.Now);

        // Intentar cancelar la cita
        cita.cancelar(DateTime.Now);

        // Cerrar la cita ejecutada
        cita.cerrar(DateTime.Now);
    }
}
```

En base al proyecto:

Pregunta 1: Unit test (10 p.).

Para el proyecto o código descrito previamente elabore **3 pruebas unitarias y 1 prueba integral** utilizando el patrón AAA, Arrange (Organizar/Inicializa), Act (Actuar) y Assert (Confirmar/Comprobar). Utilice para ello **un lenguaje de programación** que sea parte de su stack de desarrollo.

Pregunta 2: Evidences (3 p.).

En base a las pruebas realizadas previamente coloque para cada prueba:

1. El código de la prueba
2. Las evidencias de ejecución
3. Explicación resumen de la prueba
4. Sustento corto de por qué realizó esta prueba

Rúbrica de calificación

Criterio de Calificación	Excelente	Promedio	Necesita mejora	Deficiente	Calificación
C01. Building y ejecución	Al abrir el proyecto y ordenar la ejecución de las pruebas, ésta se ejecuta sin problemas.	Las pruebas no llegan a iniciar y ejecutarse, sin embargo el proceso de building llega a concluir.	Al cargar el proyecto el proceso de building presenta errores y no llega a concluir.	No elabora solución.	
	3.0 puntos	**2.0 puntos**	**1.0 punto**	**0 puntos**	
C02. Unit test	Elabora de forma correcta todas las Pruebas unitarias solicitadas, comprobando el correcto funcionamiento y garantizando la cobertura del código de los métodos.	Elabora de forma correcta 3 Pruebas unitarias solicitadas, comprobando el correcto funcionamiento y garantizando la cobertura del código de los métodos.	Elabora de parcial algunas Pruebas unitarias solicitadas, comprobando el correcto funcionamiento y garantizando de forma parcial la cobertura del código de los métodos.	No elabora las Pruebas unitarias solicitadas.	
	5.5 puntos	**3.5 puntos**	**1.5 puntos**	**0 puntos**	
C03. Patrón AAA	Utiliza de forma correcta el patrón AAA, Arrange (Organizar/Inicializa), Act (Actuar) y Assert (Confirmar/Comprobar) en la elaboración de las pruebas.	Utiliza de forma parcial el patrón AAA, Arrange (Organizar/Inicializa), Act (Actuar) y Assert (Confirmar/Comprobar) en la elaboración de las pruebas.	Solo Utiliza una parte del patrón AAA, Arrange (Organizar/Inicializa) o Act (Actuar) o Assert (Confirmar/Comprobar) en la elaboración de las pruebas.	No Utiliza el patrón AAA, Arrange (Organizar/Inicializa), Act (Actuar) y Assert (Confirmar/Comprobar) en la elaboración de las pruebas. O no elabora las Pruebas unitarias solicitadas.	
	4.0 puntos	**2.5 puntos**	**1.0 punto**	**0 puntos**	
C04. Mocking framework	Utiliza un Mocking framework (Mockito) de forma correcta para simular el comportamiento real del componentes o servicios que utiliza o llama el método, para su ejecución en la elaboración de las pruebas.	Utiliza un Mocking framework (Mockito) de forma parcial para simular el comportamiento real del componentes o servicios que utiliza o llama el método, para su ejecución en la elaboración de las pruebas.	Utiliza un Mocking framework (Mockito) para simular el comportamiento de alguno de los componentes o servicios que utiliza o llama el método, para su ejecución en la elaboración de las pruebas.	No utiliza un Mocking framework (Mockito) para simular el comportamiento componentes o servicios que utiliza o llama el método para su ejecución. O no elabora las Pruebas unitarias solicitadas.	
	4.5 puntos	**3.0 puntos**	**1.5 puntos**	**0 puntos**	
C05. Evidences	Presenta las evidencias de forma completa y clara de la ejecución correcta de todas las pruebas realizadas.	Presenta las evidencias de forma completa y clara de la ejecución correcta de 3 pruebas realizadas.	Presenta las evidencias de forma parcial de la ejecución correcta de alguna de las pruebas realizadas.	No presenta las evidencias de la ejecución correcta de las pruebas realizadas. O no elabora las Pruebas unitarias solicitadas.	
	3.0 puntos	**2.0 puntos**	**1.0 punto**	**0 puntos**	
Total	**20 puntos**	**13 puntos**	**6 puntos**	**0 puntos**	

Lima, 12 de septiembre del 2024