Curso de desarrollo de software

Cuarta Práctica Calificada

Indicaciones:

- Para esta evaluación utiliza solo el repositorio utilizado en tus actividades anteriores y guarda todas tus respuestas en un repositorio llamado PracticaCalificada4. Si el estudiante utiliza el repositorio original dado en el texto , la pregunta será eliminada.
- Cada estudiante debe presentar un repositorio personal donde se encuentra esta evaluación y las otras evaluaciones dadas en clases. No se admite otros escenarios distintos. La calificación está sujeta a las actividades anteriores también.
- Las calificaciones son individuales, por lo que evita presentar solo código sin ninguna explicación o imagen. Todo ejercicio sin explicaciones o imágenes que no apoyan tus respuestas tienen nota de 0.
- Esta práctica es sobre 14 y puntúan con otros trabajos desarrollados en clase.
- No se va a utilizar alguna herramienta de apoyo como el ChatGPT por que la evaluación no lo amerita. Si de ser así, se te eliminará el examen, ya que es un ejemplo de que no has hecho las actividades ni has estudiado.

Preguntas:

1. El comando grep de Unix busca en archivos líneas que coincidan con una expresión regular. Tu tarea es implementar un comando grep simplificado, que admita la búsqueda de cadenas fijas. Busca en archivos líneas que coincidan con una cadena de búsqueda y devuelva todas las líneas coincidentes.
   El comando grep toma tres argumentos:

   - La cadena a buscar.
   - Cero o más indicadores para personalizar el comportamiento del comando.
   - Uno o más archivos para buscar.

   Luego lee el contenido de los archivos especificados (en el orden especificado), busca las líneas que contienen la cadena de búsqueda y finalmente devuelve esas líneas en el orden en que fueron encontradas. Al buscar en varios archivos, cada línea coincidente va precedida por el nombre del archivo y dos puntos (':').

   El comando grep admite los siguientes indicadores:

- - n Antepone el número de línea y dos puntos (':') a cada línea en la salida, colocando el número después del nombre del archivo (si está presente).
- -l Muestra solo los nombres de los archivos que contienen al menos una línea coincidente.
- -i Coincidencia utilizando una comparación que no distingue entre mayúsculas y minúsculas.
- -v Invierte el programa: recopila todas las líneas que no coinciden.
- -x Busca solo líneas donde la cadena de búsqueda coincida con la línea completa.

Para facilitar esto, el archivo grep.js se configuró con un shebang y un comentario que explica lo que hace:

```
#!/usr/bin/env nodo

// La línea anterior es un shebang. En sistemas operativos o entornos tipo Unix,

// esto permitirá que el script se ejecute por node y así convertir este JavaScript

// archivo en un ejecutable. En otras palabras, para ejecutar este archivo, puede ejecutar

// lo siguiente desde tu terminal:

//

// ./grep.js args

// Si no tienes un sistema operativo o entorno tipo Unix, por ejemplo

// Windows sin WSL, puedes usar lo siguiente dentro de una terminal de ventana,

// como cmd.exe:

//

// node grep.js args

//

// Lea más sobre shebangs aquí: https://en.wikipedia.org/wiki/Shebang_(Unix)
```

Para recuperar los argumentos con los que se inició el proceso, utiliza process.argv. (implementa utilizando JavaScript) (1-punto)

2. Practiquemos la herencia y la programación orientada a objetos en Javascript. Diseña 2 clases, una llamada "Pokemon" y otra llamada "Charizard". Las clases deben hacer lo siguiente:

Clase Pokémon:

- El constructor toma 3 parámetros (HP, ataque, defensa)
- El constructor debe crear 6 campos (HP, ataque, defensa, movimiento, nivel, tipo). Los valores de (mover, nivelar, tipo) debe inicializarse en ("", 1, "").

- Implementa un método flight que arroje un error que indique que no se especifica ningún movimiento.
- Implementa un método canFly que verifica si se especifica un tipo. Si no, arroja un error. Si es así, verifica si el tipo incluye ‘’flying’. En caso afirmativo, devuelve verdadero; si no, devuelve falso.

Clase Charizard:

- El constructor toma 4 parámetros (HP, ataque, defensa, movimiento)
- El constructor configura el movimiento y el tipo (para "disparar/volar") además de establecer HP, ataque y defensa como el constructor de superclase.
- Sobreescribe el método fight . Si se especifica un movimiento, imprime una declaración que indique que se está utilizando el movimiento y devuelve el campo de ataque. Si no arroja un error. (implementa utilizando JavaScript) (1 punto).

Se crea la clase Pokemon:

```
class Pokemon {
    constructor(HP, ataque, defensa) {
        this.HP = HP;
        this.ataque = ataque;
        this.defensa = defensa;
        this.movimiento = "";
        this.nivel = 1;
        this.tipo = "";
    }

    flight() {
        throw new Error("No se especifica ningún movimiento.");
    }

    canFly() {
        if (!this.tipo) {
            throw new Error("No se especificó ningún tipo.");
        }
        return this.tipo.includes("flying");
    }
}
```

Se crea la clase Charizard: En esta clase se hereda de la clase pokemon

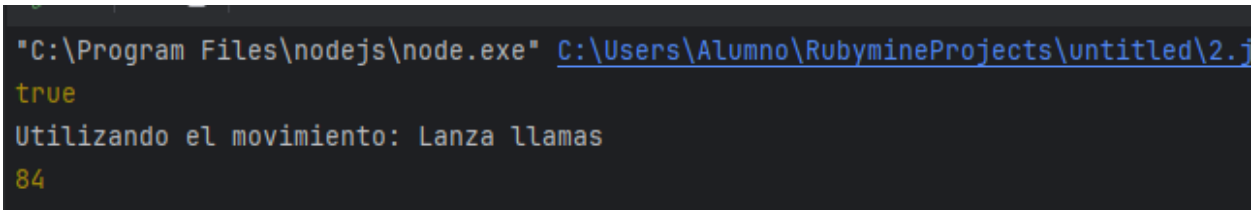
```
class Charizard extends Pokemon {
    constructor(HP, ataque, defensa, movimiento) {
        super(HP, ataque, defensa);
        this.movimiento = movimiento;
        this.tipo = "flying";
    }

    fight() {
        if (this.movimiento) {
            console.log(`Utilizando el movimiento: ${this.movimiento}`);
```

```
            return this.ataque;
        } else {
            throw new Error("No se especifica ningún movimiento.");
        }
    }
}
```

Ejemplo de uso:

```
try {
    const charizard = new Charizard(78, 84, 78, "Lanza llamas");
    console.log(charizard.canFly()); // Debería devolver true, ya que el tipo
incluye "flying"
    console.log(charizard.fight()); // Debería imprimir que se está
utilizando el movimiento y devolver el ataque
} catch (error) {
    console.error(error.message);
}
```

En este caso como charizard pertenece a la clase Charizard ya incluye el tipo flying, retornaria un true y el otro caso para ver su ataque tambien devuelve como indica la definicion:



3. El principio de inversión de dependencia establece que los módulos de alto nivel no deberían depender de los módulos de bajo nivel, y tanto los módulos de alto nivel como los de bajo nivel deberían depender de abstracciones. También establece que las abstracciones no deberían depender de implementaciones concretas, sino que las implementaciones concretas deberían depender de abstracciones.

Una implementación concreta del principio de inversión de dependencia es la inyección de dependencia, o la idea de que todo de lo que depende un objeto debe pasarse al objeto, para permitir la máxima flexibilidad. Ruby no requiere inyección de dependencia tanto como otros lenguajes de programación debido a su flexibilidad al permitir métodos singleton en casi todos los objetos. Sin embargo, la inyección de dependencia todavía se puede utilizar en Ruby y Digamos que tienes una clase CurrentDay para representar el día actual y deseas tener un método work_hours que devuelva las horas de trabajo para el día actual y método workday? que devuelve si el día actual es un día laborable o no laborable. En tu aplicación, ya tienes una clase MonthlySchedule que conoce el horario de trabajo para un mes determinado, que inicializa con el año y el mes. Aquí hay un enfoque simple para implementar esta clase:

class CurrentDay

def initialize

```
    @date = Date.today
    @schedule = MonthlySchedule.new(@date.year,
                                    @date.month)
  end
  def work_hours
     @schedule.work_hours_for(@date)
  end
  def workday?
      !@schedule.holidays.include?(@date)
   end
 end
```

¿Cuál es el problema con este enfoque dado, cuando quieres probar el metodo workday?. Utiliza la inyección de dependencia aplicado al siguiente código.

```
before do
   Date.singleton_class.class_eval do
      alias_method :_today, :today
      define_method(:today){Date.new(2020, 12, 16)}
    end
end
after do
  Date.singleton_class.class_eval do
      alias_method :today, :_today
       remove_method :_today
  end
end
```

¿Qué sucede en JavaScript con el DIP en este ejemplo? (1 punto).

En el ejemplo proporcionado en JavaScript con el patrón de inversión de dependencia, se está modificando dinámicamente el comportamiento del método today de la clase Date para que devuelva una fecha específica. Esto puede considerarse como una forma de inyección de dependencia, ya que se

está cambiando la dependencia del comportamiento de Date para adaptarse a las necesidades de las pruebas.

Los ejercicios a partir de aquí se recomiendan hacerlos en orden.

Pregunta: (Para responder esta pregunta utiliza el repositorio y las actividades que has desarrollado de Introducción a Rails)

1. Modifique la lista de películas de la siguiente manera. Cada modificación va a necesitar que realice un cambio en una capa de abstracción diferente
    a. Modifica la vista Index para incluir el número de fila de cada fila en la tabla de películas. Se le hace las siguientes modificaciones al archivo index.hmtl.erb:

```
<div id="movies">
  <div class="row">

    <div class="col-1">n° fila</div>
    <div class="col-8">Movie Title</div>
    <div class="col-2">Rating</div>
    <div class="col-2">Release Date</div>
  </div>
  <%- @movies.each.with_index(1) do |movie, index| %>
    <div class="row">

      <div class="col-8"><%= index %></div>
      <div class="col-8"> <%= link_to movie.title, movie_path(movie) %>
</div>
      <div class="col-2"> <%= movie.rating %></div>
      <div class="col-2"> <%= movie.release_date.strftime('%F') %> </div>
    </div>
  <% end %>
</div>
```

b. Modifica la vista Index para que cuando se sitúe el ratón sobre una fila de la tabla, dicha fila cambie temporalmente su color de fondo a amarillo u otro color.
En el archivo show.js dentro de la carpeta javascripts hacemos la modificación
Se hace el cambio en el archivo application.css en la carpeta stylesheets:

```
.highlighted-row {
  background-color: yellow;S
}
```

Y se agrega las siguiente líneas en el archivo show.js

```
$(document).on('turbolinks:load', function() {
    $('#movies tbody tr').hover(
        function() {
            $(this).addClass('highlighted-row');
            },
        function() {
            $(this).removeClass('highlighted-row');
            }
    );
});
```

c. Modifica la acción Index del controlador para que devuelva las películas ordenadas alfabéticamente por título, en vez de por fecha de lanzamiento. No intentes ordenar el resultado de la llamada que hace el controlador a la base de datos. Los gestores de bases de datos ofrecen formas para especificar el orden en que se quiere una lista de resultados y, gracias al fuerte acoplamiento entre ActiveRecord y el sistema gestor de bases de datos (RDBMS) que hay debajo, los métodos find y all de la biblioteca de ActiveRecord en Rails ofrece una manera de pedirle al RDBMS que haga esto.
Se hace la modificación en el archivo movies_controller.rb en la función index:

```
def index
  @movies = Movie.order(:title)
end
```

The Help	PG-13	2011-08-10
21		
The Roundup	G	2023-11-05
22		
When Harry Met Sally	R	1989-07-21
23		

d. Simula que no dispones de ese fuerte acoplamiento de ActiveRecord, y que no puedes asumir que el sistema de almacenamiento que hay por debajo pueda devolver la colección de ítems en un orden determinado. Modifique la acción Index del controlador para que devuelva las películas ordenadas alfabéticamente por título. Utiliza el método sort del módulo Enumerable de Ruby.

En el archivo movies_controller.rb hacemos el siguiente cambio en la función index:

```
def index
  @movies = Movie.all.sort_by(&:title)
end
```

Aladdin	G	1992-11-25
6		
Ed	G	2023-10-31
7		
Raiders of the Lost Ark	PG	1981-06-12
8		

Para que esta página puntué por lo menos debes hacer 3 ítems.

Pregunta: (para responder esta pregunta utiliza el repositorio y las actividades que has realizado de Rails avanzado, en particular asociaciones) - 2 puntos

1. Extienda el código del controlador del código siguiente dado con los métodos edit y update para las críticas. Usa un filtro de controlador para asegurarte de que un usuario sólo puede editar o actualizar sus propias críticas. Revisa el código dado en la evaluación y actualiza tus repositorios de actividades (no se admite nada nuevo aquí). Debes mostrar los resultados.

Preguntas: (estas preguntas son utilizando el repositorio de todas tus actividades relacionada a JavaScript, por lo tanto, no hay respuestas únicas) - 6 puntos

1. Un inconveniente de la herencia de prototipos es que todos los atributos (propiedades) de los objetos son públicos. Sin embargo, podemos aprovechar las clausuras para obtener atributos privados. Crea un sencillo constructor para los objetos User que acepte un nombre de usuario y una contraseña, y proporcione un método checkPassword que indique si la contraseña proporcionada es correcta, pero que deniegue la inspección de la contraseña en sí. Esta expresión de “sólo métodos de acceso” se usa ampliamente en jQuery. Sugerencia: El constructor debe devolver un objeto en el que una de sus propiedades es una función que aprovecha las clausuras de JavaScript para ‘recordar’ la contraseña proporcionada inicialmente al constructor. El objeto devuelto no debería tener ninguna propiedad que contenga la contraseña).
2. Extienda la función de validación en ActiveModel para generar automáticamente código JavaScript que valide las entradas del formulario antes de que sea enviado. Por ejemplo, puesto que el modelo Movie de RottenPotatoes requiere que el título de cada película sea distinto de la cadena vacía, el código JavaScript deberías evitar que el formulario “Add New Movie” se enviara si no se cumplen los criterios de validación, mostrar un mensaje de ayuda al usuario, y resaltar el(los) campo(s) del formulario que ocasionaron los problemas de validación. Gestiona, al menos, las validaciones integradas, como que los títulos sean distintos de cadena vacía, que las longitudes máximas y mínima de la cadena de caracteres sean correctas, que los valores numéricos estén dentro de los límites de los rangos, y para puntos adicionales, realiza las validaciones basándose en expresiones regulares.

3. En el código utilizado en la sección de eventos y funciones callback, supongamos que no puedes modificar el código del servidor para añadir la clase CSS adult a las filas de la tabla movies. ¿Cómo identificaría las filas que están ocultas utilizando sólo código JavaScript del lado cliente?
4. Siguiendo la estrategia del ejemplo de jQuery de la misma sección anterior de eventos y funciones callback, utiliza JavaScript para implementar un conjunto de casillas de verificación (checkboxes) para la página que muestra la lista de películas, una por cada calificación (G, PG, etcétera), que permitan que las películas correspondientes permanezcan en la lista cuando están marcadas. Cuando se carga la página por primera vez, deben estar marcadas todas; desmarcar alguna de ellas debe esconder las películas con la clasificación a la que haga referencia la casilla desactivada.
5. Escribe el código AJAX necesario para crear menús en cascada basados en una asociación has_many. Esto es, dados los modelos de Rails A y B, donde A has_many (tiene muchos) B, el primer menú de la pareja tiene que listar las opciones de A, y cuando se selecciona una, devolver las opciones de B correspondientes y rellenar el menú B.
6. Extienda la funcionalidad del ejemplo dado en la actividad de AJAX: JavaScript asíncrono y XML de forma que, si el usuario expande u oculta repetidamente la misma fila de la tabla de películas, sólo se haga una única petición al servidor para la película en cuestión la primera vez. En otras palabras, implementa una memoria caché con JavaScript en el lado cliente para la información de la película devuelta en cada llamada AJAX.