|  |  |  |
| --- | --- | --- |
|  | **Parcial #1**  Desarrollo de Software Web | Código: |
| Versión 01 |
| Edición: 17/03/2021 |

Java La empresa botanical S.A requiere llevar un control de las diferentes especies de plantas que maneja con el fin de tener información disponible de las mismas entre las plantas que tienen manejan lechugas, moras y zanahorias con los siguientes datos

* Nombre
* color
* edad (lechugas 1 mes, zanahoria 3 meses y moras 6 meses)

1. Muestre el listado de plantas (con al menos 2 lechugas)en formato xml (se deben respetar los conceptos de programación orientada a objetos; Encapsulación, polimorfismo, herencia, abstracción), realizar una distribución de paquetes teniendo en cuenta el patrón MVC. (3)

* Funcionalidad (.5)
* POO (1)
* Codificación y Patrón usado (.5)
* Realice un diagrama de clases y uno de paquetes para la aplicación descrita (.5)
* Suba el código al git en la rama master (0.25)
* Suba el diagrama en la rama dev (0,25)

1. Trasformar el siguiente xml en Json (.5)
2. Explique los principios solid(.5)

Principio de Responsabilidad unica: este principio nos dice que una clase debe tener solo una razion para poder cambiar.

Principio de Abierto-cerrado:

Dice que las entidades de software deben estar abiertas para su extensión pero cerradas para su modificación.

Principio de Sustitución de Liskov:

Los objetos de una clase deben ser capaces de sustituir a los objetos de la clase base sin ser afectado.

Principio de Segregación de la Interfaz

Ninguna clase con it-nterface debe depender de métodos que no usa.

Principio de Inversión de Dependencia

Dice que las clases de nivel alto no deben depender de clases de bajo nivel, por el contrario ambas depender.

Dada la clase

Public class Bicicleta {

Public Bicicleta(){

}

Public void armarbicicleta(){

unir(armarRuedas(), armarChasis());

}

Public String armarChasis(){}

Public String armarRuedas(){}

}

Refactorizar para que cumpla principio single responsability

public class Bicicleta {

private Chasis chasis;

private Rueda ruedaDelantera;

private Rueda ruedaTrasera;

public Bicicleta() {

chasis = new Chasis();

ruedaDelantera = new Rueda();

ruedaTrasera = new Rueda();

}

public void armarBicicleta() {

chasis.armar();

ruedaDelantera.armar();

ruedaTrasera.armar();

}

}

public class Chasis {

public void armar() {

// Lógica para armar el chasis

}

}

public class Rueda {

public void armar() {

// Lógica para armar una rueda

}

}