Universidad Israel

Programación Avanzada

Tema: Introducción al Desarrollo Web

Docente: *Mg. Mario Ruben Pérez Cargua*

Universidad Israel

Objetivo

Recordar conceptos importantes para el desarrollo web

Contenido

- Programación Orienta a Objetos
- MVC

Universidad Israel

- Programación Orientada a Objetos

Programación Orientada a Objetos / Elementos

Antecedentes

La programación Orientada a Objetos se basa en 5 elementos o pilares

Programación Orientada a Objetos / Elementos..

Abstracción

Abstracción

El término abstracción consiste en **ver a algo como un todo sin saber cómo está formado internamente**.

Abstracción: Capacidad del ser humano para entender una situación excluyendo detalles y sólo viéndola a alto nivel. El hombre ha comprendido el mundo con la abstracción. Esta propiedad permite distinguir a un objeto de los demás, observando sus características y comportamientos, pensando en qué es y no en cómo se codificaría en un lenguaje. Con la abstracción se destaca lo importante y se ignora lo irrelevante, o sea, hay ocultamiento de información. Hay abstracción de datos al declarar una variable tipo *integer*, ya que internamente el compilador lo implementa en 2 *bytes*, lo cual es transparente al programador, o al declarar una variable *date*, el compilador controla los días de los meses, acepta sólo operaciones válidas entre las fechas, permitiendo al programador abstraerse de esos detalles. Estos tipos de datos abstractos coleccionan valores y operaciones, los cuales se usan transparentemente sin importar su implementación: otro lo implementa y yo lo uso.

Programación Orientada a Objetos / Elementos..

Encapsulamiento

Encapsulamiento

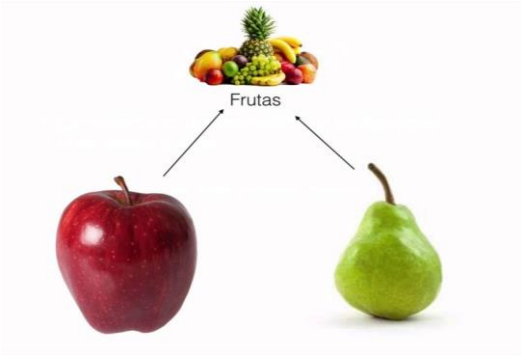
Encapsulación de información: Ocultamiento de información, datos o funciones especiales a los usuarios. En el caso de la programación, el encapsulamiento es lo que permite que tanto la estructura (campos) como el comportamiento (métodos) se encuentren dentro del mismo cuerpo de código de la clase con la que se crean los objetos. Dentro de la clase se deben agrupar tanto la información o datos de los campos como las operaciones o métodos o funciones que operan sobre esta información

Programación Orientada a Objetos / Elementos..

Herencia

Herencia

La herencia es uno de los mecanismos de los lenguajes de programación orientada a objetos basados en clases, por medio del cual una clase se deriva de otra de manera que extiende su funcionalidad. La clase de la que se hereda se suele denominar *clase base*, *clase padre*, *superclase*, *clase ancestro* (el vocabulario que se utiliza suele depender en gran medida del lenguaje de programación), y la que hereda se denomina *clase hija*

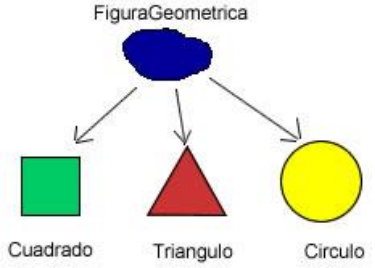


Programación Orientada a Objetos / Elementos..

Polimorfismo

Polimorfismo

En programación orientada a objetos se denomina polimorfismo a la capacidad que tienen los objetos de una clase de responder al mismo mensaje o evento en función de los parámetros utilizados
durante su invocación.
Un objeto polimórfico es una entidad que puede contener valores de diferentes tipos durante la ejecución del programa.



Dicho de otra forma, el polimorfismo consiste en conseguir que un objeto de una clase se comporte como un objeto de cualquiera de sus subclases, dependiendo de la forma de llamar a los métodos de dicha clase o subclases. Una forma de conseguir objetos polimórficos es mediante el uso de punteros a la superclase.
De esta forma podemos tener dentro de una misma estructura (arrays,listas,pilas,colas,..) objetos de distintas
subclases, haciendo que el tipo base de dichas estructuras sea un puntero a la superclase. Otros lenguajes nos dan la posibilidad de crear objetos polimórficos con el operador new.

Programación Orientada a Objetos / Elementos..

Modularidad

Modularidad

Proceso de crear partes de un todo que se integran perfectamente entre sí para que funcionen por un objetivo general, y a las cuales se les pueden agregar más componentes que se acoplen perfectamente al todo, o extraerle componentes sin afectar su funcionamiento.

Un ejemplo clásico es un conjunto de módulos que, al integrarlos conforman un armario, el cual puede agregarle más funcionalidad si se le agregan más módulos, o al contrario. También se puede cambiar su finalidad si se acomodan esos módulos para darle otro objetivo: volverlo una mesa.
Esto ayuda a la descomposición de problemas en subproblemas, es decir, a la solución de problemas por composición de soluciones a subproblemas.

Programación Orientada a Objetos / Ejemplos

Ejemplo 1/2

Abstracción: Cuando creamos la **clase Celular**, estamos haciendo abstracción puesto que cuando creemos un objeto de tipo Celular, al usuario le interesa el objeto celular, no le interesa que es lo que tiene dentro la clase ni como está implementada, si no utilizar los métodos del objeto, llamar y cortar llamada e informar características.

Encapsulamiento: Las palabras reservadas **private, public, default,** dan cierto grado de visibilidad del contenido de la clase, por ejemplo la palabra reservada **private** solo permite acceder a ese atributo desde dentro de la misma clase, si intentas acceder desde el objeto creado, por ejemplo *celular.marca* lanzará un error.

Programación Orientada a Objetos / Ejemplos

Ejemplo 1/2

```java
package javaapplication1;
/**
 *
 * @author mrperez
 */
public class Celular {
        //atributos de la clase
        private String marca;
        private String modelo;
        private String color;
        // constructor con parámetros
        public Celular(String marca, String modelo, String color) {
                this.marca = marca;
                this.modelo = modelo;
                this.color = color;
        }
        //constructor vacio
        public Celular(){
        }
        // método hacer llamada
        public void llamar(String nombre){
                System.out.println("LLamando a "+nombre);
        }
        //método finalizar llamada
        public void cortarLlamada(){
                System.out.println("Llamada finalizada");
        }
        //método para informar de la caracteristicas del celular
        public void informarCaracteristicas(){
                System.out.println(String.format("Celular Marca: %s", marca));
                System.out.println(String.format("Celular  Modelo: %s", modelo));
                System.out.println(String.format("Celular Color: %s", color));
        }
```

Programación Orientada a Objetos / Ejemplos

Ejemplo 2/2

Como puedes ver con la **herencia** se ha utilizado atributos definidos en la clase Celular, y el método informarCaracterísticas () se ha sobrescrito, la sobrescritura es el proceso de modificar el código al método de la clase padre, la sobrescritura es un tema que se tratará más a fondo en un próximo tutorial.

Programación Orientada a Objetos / Ejemplos

Ejemplo 2/2

```
package javaapplication1;
/**
 *
 * @author mrperez
 */
public class SmartPhone extends Celular{
        private float pixelesCamara;
        private float tamanioMemoriaInterna;
        private float tamanioMemoriaExterna;
        //constructor por defecto
        public SmartPhone() {
        }
        //constructor con los atributos de la clase incluso los heredados
        public SmartPhone(String marca, String modelo, String color, float pixelesCamara, float tamanioMemoriaRam,
                        float tamaoDisco) {
                super(marca, modelo, color);
                this.pixelesCamara = pixelesCamara;
                this.tamanioMemoriaInterna = tamanioMemoriaRam;
                this.tamanioMemoriaExterna = tamaoDisco;
        }
        // método sobrescrito (override), utilizo código de la clase Celular y añado código que necesito
        @Override
        public void informarCaracteristicas() {
                // TODO Auto-generated method stub
                super.informarCaracteristicas();
                System.out.println(String.format("SmartPhone calidad cámara: %s pixeles", pixelesCamara));
                System.out.println(String.format("SmartPhone tamaño memoria interna: %s GB", tamanioMemoriaInterna
                System.out.println(String.format("SmartPhone tamaño memoria externa: %s GB", tamanioMemoriaExterna
        }
        //getters y setters
        public float getPixelesCamara() {
                return pixelesCamara;
```

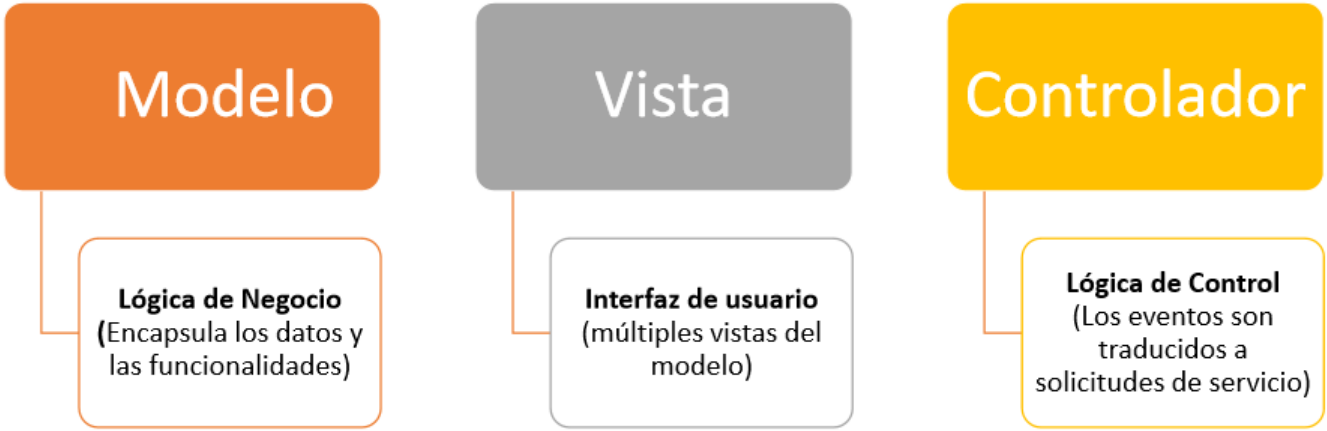
Universidad Israel

- Modelo - Vista - Controlador

Programación Avanzada/ MVC

Definición

El Modelo Vista Controlador (MVC) es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos(Modelo, Vista y Controlador)



Programación Avanzada/MVC

Elementos MVC

Modelo

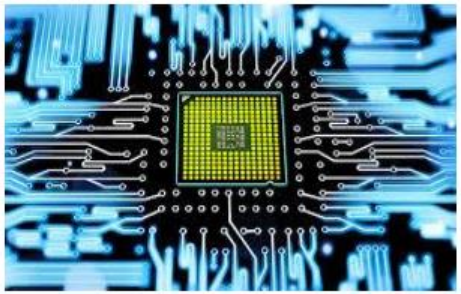
- **Es la capa donde se trabaja con los datos**, por tanto contendrá mecanismos para acceder a la información y también para actualizar su estado).

Vista

- Las vistas son las porciones de la aplicación MVC que presentan salida al usuario, Presenta el "modelo" (información y lógica de negocio) en un formato adecuado para interactuar (**interfaz de usuario**).

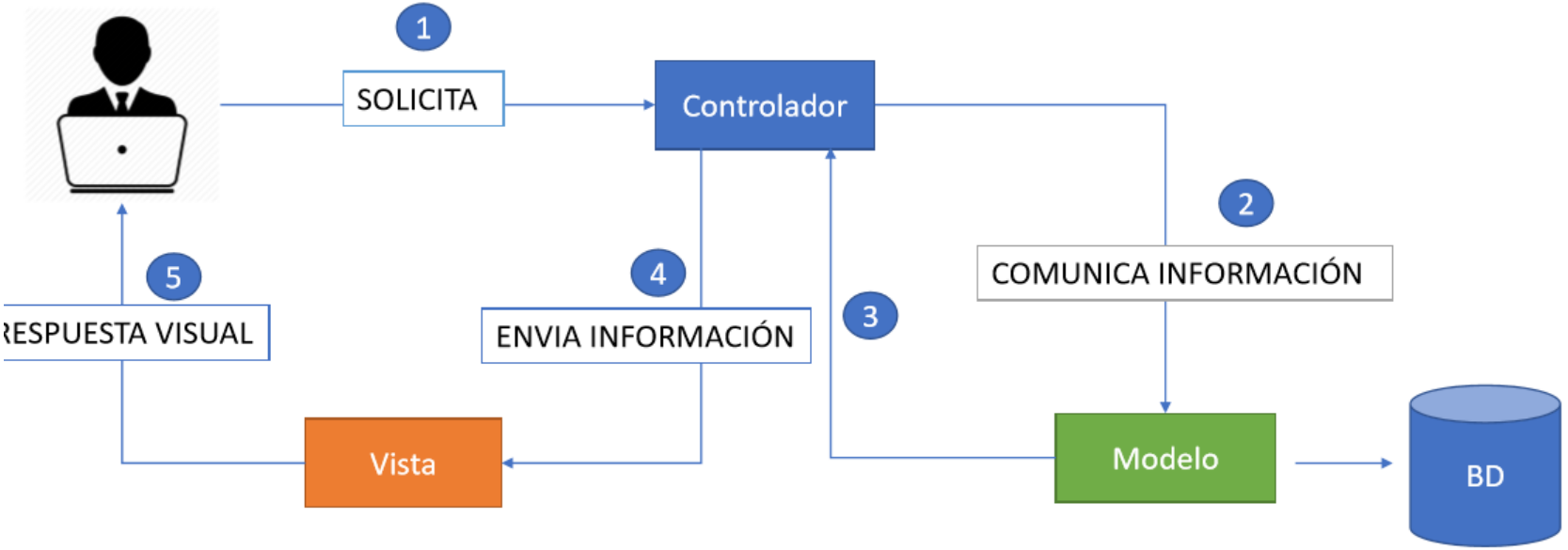
Controlador

- El controlador es el cerebro de la aplicación MVC.
- Responde a eventos (acciones del usuario) e invoca peticiones al "modelo" cuando se hace alguna solicitud sobre la información



Programación Avanzada/ MVC..

Funcionamiento



Programación Avanzada/ MVC..

Ventajas y Desventajas

Ventajas

- Reutilización de código
- Separación de conceptos
- Facilidad de mantenimiento

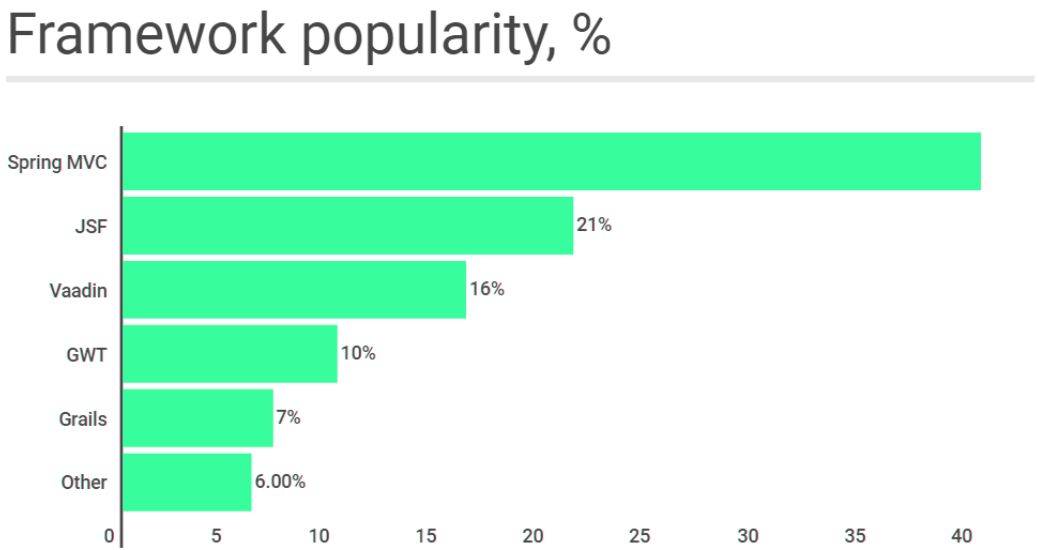
Desventajas

- Si el diseño del software o sitio no es complejo posiblemente no lo necesite.
- La curva de aprendizaje es muy alta
- Puede obtener una gran cantidad de archivos.

Programación Avanzada/ Framework

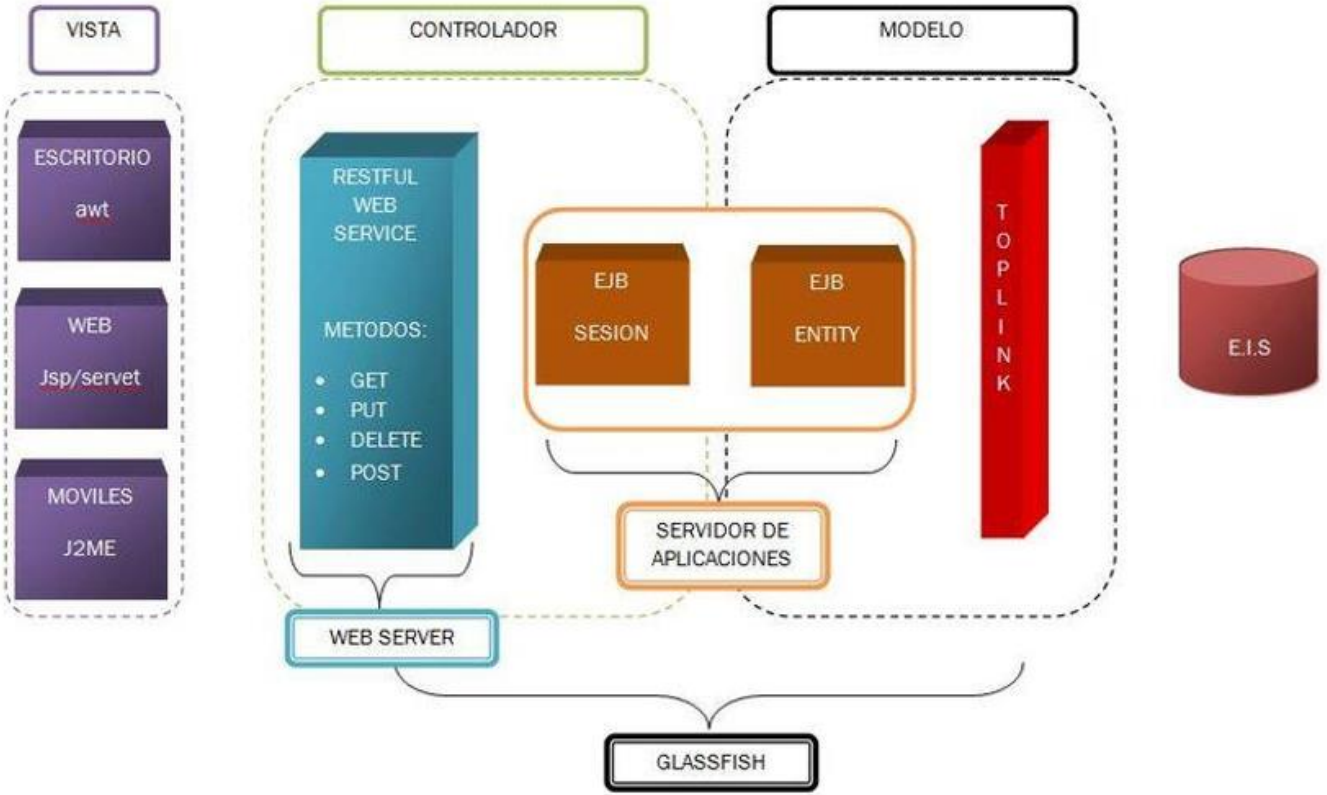
Frameworks

Un Framework es una estructura de soporte definida en la cual un proyecto de software puede ser organizado y desarrollado. Típicamente, un Framework puede incluir soporte de programas, bibliotecas y un lenguaje de scripting para ayudar a desarrollar y unir los diferentes componentes de un proyecto



https://data-flair.training/blogs/java-frameworks/

Arquitectura MVC Básica



Programación Avanzada/ Maven

Que es Maven?

Maven es una herramienta open-source, que se creó en 2001 con el objetivo de simplificar los procesos de build (compilar y generar ejecutables a partir del código fuente).
Antes de que Maven proporcionara una interfaz común para hacer builds del software, cada proyecto solía tener a alguna persona dedicada exclusivamente a configurar el proceso de build.

https://www.javiergarzas.com/2014/06/maven-en-10-min.html

Maven

Software

Maven es una herramienta de software para la gestión y construcción de proyectos Java creada por Jason van Zyl, de Sonatype, en 2002. Es similar en funcionalidad a Apache Ant, pero tiene un modelo de configuración de construcción más simple, basado en un formato XML. Wikipedia

Escrito en: Java

Desarrollador: Apache Software Foundation

Última versión estable: 3.6.2 (info); 27 de agosto de 2019 (6 meses y 1 día)

Tipo de programa: build automation; software libre; sistema de gestión de paquetes

Licencia: Licencia Apache 2.0

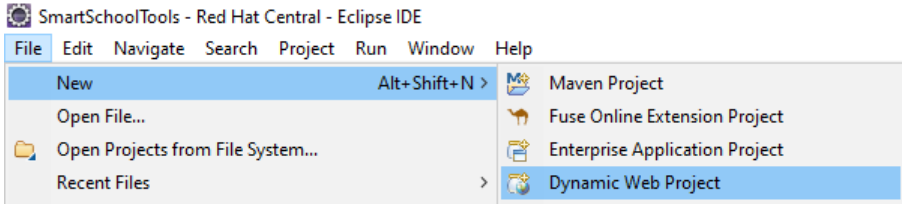
Plataforma: máquina virtual Java

Programación Avanzada/ Crear Proyecto

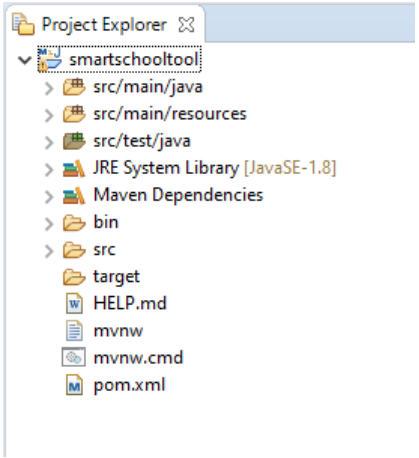
Crear Proyecto Maven

Crear Proyecto

File/new/Dynamic Web Proyect



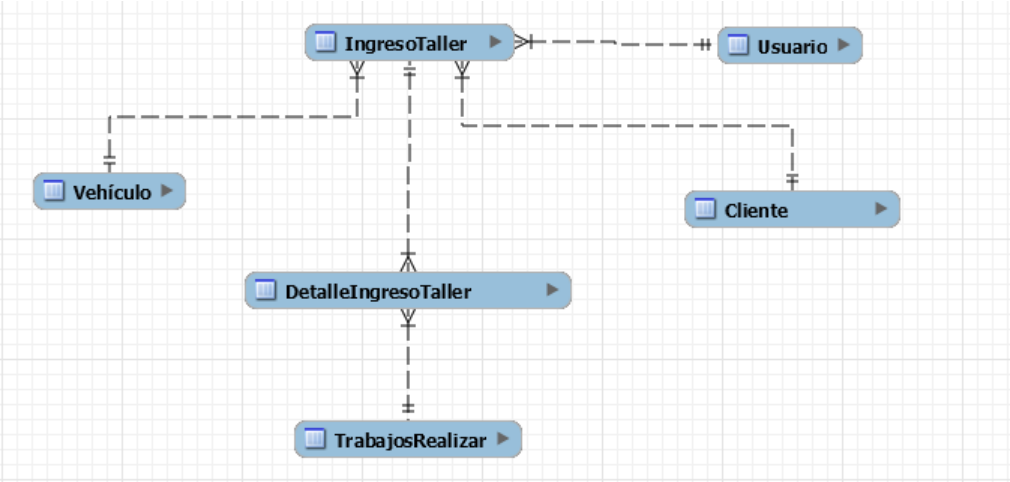
Arquitectura del proyecto.
src/main/java
src/main/resources
src/main/webapp
src/test/java
src/test/resource



Direccionamiento actividades Semestre

Ejercicios de Clase

La empresa “TécniCar”, dedicada al mantenimiento preventivo, correctivo vehículos multi-marca, nos ha solicitado crear el Módulo de Ingreso a Taller, Se ha considerado las siguientes Entidades Básicas



Universidad Israel

Gracias

Responsabilidad con pensamiento positivo

Tareas

- **Realizar una investigación sobre las Buenas Prácticas del desarrollo de Software y el estilo CamelCase**
- **Crear Arquitectura MVC en Eclipse Java**

NOTA: El deber será enviado en formato rar, especificando su nombre y número de semana.

Ejemplo: **Semana1_Mario_Pérez.rar**