TEMA: UML y Patrones de diseño

Diseño Orientado a Objetos

* Modelo para guiar al programador durante el proceso de diseño de software bajo el paradigma 00.
* Provee un conjunto de principios para calificar qué tan bien está el diseño de software
* Primera Fase: Modelado Conceptual
* Entender el dominio del problema y crear un modelo conceptual. Este modelo se puede hacer a criterio del diseñador
* Debe ser fácil de entender
* Storyboard
* ¿Cómo calza el software en el cuadro completo?
* Segunda Fase: Análisis
* Construir historial de usuario/ requerimientos
* User Stories siguen un formato “Yo como \_\_<rol>\_\_\_ quiero ser capaz de \_\_\_\_<accion>\_\_\_\_ con el fin de \_\_\_\_\_<objetivos>\_\_\_\_\_ “ .
* Se pueden crear prototipos para entender mejor el problema
* Tercera Fase: Arquitectura
* Definir la estructura de la solución para cumplir los requerimientos funcionales y de calidad
* Documentar las decisiones
* Cuarta fase : Diseño detallado
* construir de forma iterativa, un diseño detallado del sistema por construir
* Paso #1 Identifique las clases.

-Busque los sustantivos en las historias de usuario

-Algunos sustantivos se convertirán en clases. Otras se eliminan y otros se unen

-Las clases deben tener una sola responsabilidad.

* Paso #2 identifique asociaciones:

-Identificar como interactúan las clases entre si

-una asociación tiende a c convertirse en un atributo

* Patrones de diseño:
* Un patrón es una regularidad
* En el software hay regularidades o problemas recurrentes.
* Reutilizar buenas soluciones es una buena practica conocida por los ingenieros de software
* Basar nuevas soluciones en experimentar previas
* Patrón de diseño:
* Solución general para un problema común en un contexto dado.
* Es una buena practica
* Son conocidos por muchos profesionales a nivel mundial

Singleton: Patron que permite restringir la instanciación de una clase.

Únicamente se puede instanciar una sola clase.

\*En el proyecto se deben usar patrones de diseño, no forzosos.

Public class Singleton {

Private static Singleton instance = null;

Private Singleton(){

}

Public void do Something(){

}

Public static Singleton getInstance(){

If(instance == null){

Instance = new Singleton();

}

Return instance;

}

}

main:

Singleton.getInstance().doSomething();

Singleton var = Singleton.getInstance(); <

Builder: Delegar la construcción de una clase compleja a una clase complementaria.

ANTES

Public class Phone{

Private Cámara;

Private Radio;

Private TouchID;

Private Waterproo;

Public Phone(camara,pRadio,pTouchID,pwaterproof){

}

Public phone(camara){

}

}

Phone P1 = new Phone(null,null.touchID,null);

DESPUES

Public class Phone{

Private Cámara;

Private Radio;

Private TouchID;

Private Waterproo;

Private Phone(Builder builder){

This.camara = builder.camara

}

Public static class Builder{

Cámara;

Radio;

touchID;

WaterProof;

}

Public Builder withCamara(pCamara){

This.Camara = pCamara;

Return This;

}

Public Phone build(){

Return newPhone(this);

}

}

Phone p2 = new Phone.Builder()

.withCamara(\_\_\_\_\_)

.withRadio(\_\_\_\_\_\_)

.build();