# Planeación:

Objetivo: crear una aplicación donde se registre el cliente y se pueda asignar una categoría dependiendo el área en donde se va a trabajar. La cual involucra las tecnologías de la información como motor para satisfacer las necesidades del cliente reduciendo costos a lo largo del tiempo y aumentando la productividad.

Es necesario el diseño de una estrategia que permita utilizar la tecnología para la identificación de las circunstancias que afectan primordialmente la relación con nuestros clientes, analizar la situación y decidir de manera oportuna coordinada por todas las áreas de contacto y soporte.

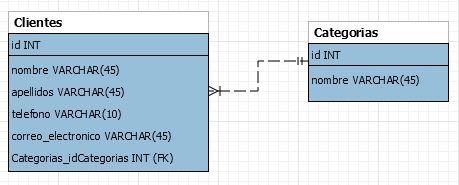
Alcance:el sistema se desarrollará en su primera etapa realizando las funciones de agregar, modificar y eliminar un cliente, así como una categoría.

Este sistema deberá cumplir con requerimientos específicos demandados por el cliente.

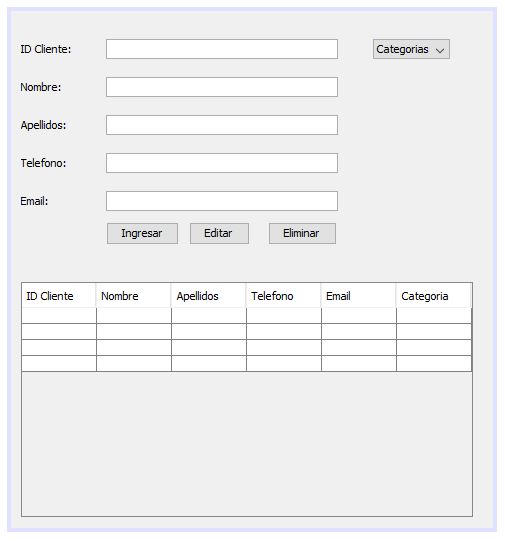
Requerimientos:

* no poder eliminar una categoría, si está relacionada con un cliente.
* Nombre y apellidos son obligatorios.
* Correo electrónico valido.
* El nombre de la categoría no puede quedar vacío.
* El correo electrónico no se puede duplicar.

Modelo de base de datos:



Modelo:



Definición de línea de código:

**Convención de nombres**:

**Nombre de archivos**: Se escribe con mayúscula al inicio, tipo de extensión y versión.

Ejemplo:

DocumentoAplicacion.java

**Nombre de clases**: Se escribe con mayúscula al inicio del nombre de la clase y si se conforma por más de dos palabras se usa mayúscula al inicio de cada palabra.

Ejemplo:

public class Main {

…

}

**Nombre de funciones y métodos**: se escribe con mayúscula al inicio de cada palabra y haciendo referencia a la acción que pretende cumplir dicho método o función.

Ejemplo:

public tipo Funcion {

…

}

**Nombre de variables / propiedades:** se escribe con mayúscula al inicio de cada palabra haciendo referencia al objetivo que se desea manipular para poder identificar dicha propiedad, ya sea con un nombre completo o abreviado y haciendo referencia al tipo de dato que sea.

Ejemplo:

Public int Numero;

**Nombre de constantes:** se escribe con mayúscula al inicio de cada palabra haciendo referencia al objetivo que se desea manipular para poder identificar dicha propiedad, ya sea con un nombre completo o abreviado, pero siempre iniciando con la abreviación Cons (referente a constante) seguido de la acción referenciada.

Ejemplo:

String ConsEjemplo;

**Nombre de las tablas de base de datos:**  se escribe con mayúsculas al inicio y si se compone de más de una palabra, se separará con un guion bajo (\_), dicho nombre solo se escribirá en plural.

Ejemplo:

CREATE TABLE IF NOT EXISTS `mydb`.`Clientes`

Estructuras de control:

Las estructuras de control determinan la secuencia de ejecución de las sentencias de un programa.

Las estructuras de control se dividen en tres categorías:

1. Secuencial
2. Condicional o Selectiva
3. Iterativa o Repetitiva.

**1. ESTRUCTURA SECUENCIAL**

El orden en que se ejecutan por defecto las sentencias de un programa es secuencial. Esto significa que las sentencias se ejecutan en secuencia, una después de otra, en el orden en que aparecen escritas dentro del programa.

La estructura secuencial está formada por una sucesión de instrucciones que se ejecutan en orden una a continuación de la otra.

Cada una de las instrucciones están separadas por el carácter punto y coma (;).

Las instrucciones se suelen agrupar en bloques.

El bloque de sentencias se define por el carácter llave de apertura ({) para marcar el inicio del mismo, y el carácter llave de cierre (}) para marcar el final.

Ejemplo:

{

instrucción 1;

instrucción 2;

instrucción 3;

}

**2. ESTRUCTURA CONDICIONAL, ALTERNATIVA O SELECTIVA**

La estructura condicional determina si se ejecutan unas instrucciones u otras según se cumpla o no una determinada condición.

En java la estructura condicional se implementa mediante:

* Instrucción if.
* Instrucción switch.
* Operador condicional ? :

**2.1 INSTRUCCION if**

Puede ser del tipo:

* Condicional simple: if
* Condicional doble: if ... else ...
* Condicional múltiple: if .. else if ..

La condición debe ser una expresion booleana es decir debe dar como resultado un valor booleano (true ó false).

Condicional simple: se evalúa la condición y si ésta se cumple se ejecuta una determinada acción o grupo de acciones. En caso contrario se saltan dicho grupo de acciones.

if(expresión\_booleana){

instrucción 1

instrucción 2

.......

}

Si el bloque de instrucciones tiene una sola instrucción no es necesario escribir las llaves { } aunque para evitar confusiones se recomienda escribir las llaves siempre.

Condicional doble: Se evalúa la condición y si ésta se cumple se ejecuta una determinada instrucción o grupo de instrucciones. Si no se cumple se ejecuta otra instrucción o grupo de instrucciones.

if(expresión booleana){

instrucciones 1

}

else{

instrucciones 2

}

**Condicional múltiple**: Se obtiene anidando sentencias if ... else. Permite construir estructuras de selección más complejas.

if (expresion\_booleana1)

instruccion1;

else if (expresion\_booleana2)

instruccion2;

else

instruccion3;

Cada else se corresponde con el if más próximo que no haya sido emparejado.

Una vez que se ejecuta un bloque de instrucciones, la ejecución continúa en la siguiente instrucción que aparezca después de las sentencias if .. else anidadas.

**2.2 INSTRUCCION switch**

Se utiliza para seleccionar una de entre múltiples alternativas.

La forma general de la instrucción switch en Java es la siguiente:

switch (expresión){

case valor 1:

instrucciones;

break;

case valor 2:

instrucciones;

break;

· · ·

default:

instrucciones;

}

La instrucción switch se puede usar con datos de tipo byte, short, char e int. También con tipos enumerados y con las clases envolventes Character, Byte, Short e Integer. A partir de Java 7 también pueden usarse datos de tipo String en un switch.

Funcionamiento de la instrucción switch:

* Primero se evalúa la expresión y salta al case cuya constante coincida con el valor de la expresión.
* Se ejecutan las instrucciones que siguen al case seleccionado hasta que se encuentra un break o hasta el final del switch. El break produce un salto a la siguiente instrucción a continuación del switch.
* Si ninguno de estos casos se cumple se ejecuta el bloque default (si existe). No es obligatorio que exista un bloque default y no tiene porqué ponerse siempre al final, aunque es lo habitual.

**2.3 OPERADOR CONDICIONAL ? :**

Se puede utilizar en sustitución de la sentencia de control if-else.

Los forman los caracteres ? y :

Se utiliza de la forma siguiente:

expresión1 ? expresión2 : expresión3

Si expresión1 es cierta entonces se evalúa expresión2 y éste será el valor de la expresión condicional. Si expresión1 es falsa, se evalúa expresión3 y éste será el valor de la expresión condicional.

**3. ESTRUCTURA ITERATIVA O REPETITIVA**

Permiten ejecutar de forma repetida un bloque específico de instrucciones.

Las instrucciones se repiten mientras o hasta que se cumpla una determinada condición. Esta condición se conoce como condición de salida.

Tipos de estructuras repetitivas:

* + ciclo while
  + ciclo do – while
  + ciclo for

**3.1 CICLO WHILE**

Las instrucciones se repiten mientras la condición sea cierta. La condición se comprueba al principio del bucle por lo que las acciones se pueden ejecutar 0 ó más veces.

La ejecución de un bucle while sigue los siguientes pasos:

1. Se evalúa la condición.

2. Si el resultado es false las instrucciones no se ejecutan y el programa sigue ejecutándose por la siguiente instrucción a continuación del while.

3. Si el resultado es true se ejecutan las instrucciones y se vuelve al paso 1.

Ejemplo:

public class Ejemplo1While {

public static void main(String[] args) {

int suma = 0, num;

Scanner sc = new Scanner(System.in);

System.out.print("Introduzca un número: ");

num = sc.nextInt();

while (num >= 0){

suma = suma + num;

System.out.print("Introduzca un número: ");

num = sc.nextInt();

}

System.out.println("La suma es: " + suma );

}

}

**2 CICLO DO – WHILE**

Las instrucciones se ejecutan mientras la condición sea cierta.

La condición se comprueba al final del bucle por lo que el bloque de instrucciones se ejecutarán al menos una vez. Esta es la diferencia fundamental con la instrucción while. Las instrucciones de un bucle while es posible que no se ejecuten si la condición inicialmente es falsa.

La ejecución de un bucle do - while sigue los siguientes pasos:

1. Se ejecutan las instrucciones a partir de do{

2. Se evalúa la condición.

3. Si el resultado es false el programa sigue ejecutándose por la siguiente instrucción a continuación del while.

4. Si el resultado es true se vuelve al paso 1

Ejemplo:

public class Ejemplo1DoWhile {

public static void main(String[] args) {

int valor;

Scanner in = new Scanner( System.in );

do {

System.out.print("Escribe un entero < 100: ");

valor = in.nextInt();

}while (valor >= 100);

System.out.println("Ha introducido: " + valor);

}

}

**3.3 CICLO FOR**

Hace que una instrucción o bloque de instrucciones se repitan un número determinado de veces mientras se cumpla la condición.

La estructura general de una instrucción for en Java es la siguiente:

for(inicialización; condición; incremento/decremento){

instrucción 1;

...........

instrucción N;

}

A continuación de la palabra for y entre paréntesis debe haber siempre tres zonas

separadas por punto y coma:

* + zona de inicialización.
  + zona de condición
  + zona de incremento ó decremento.

Si en alguna ocasión no es necesario escribir alguna de ellas se pueden dejar en blanco, pero los dos punto y coma deben aparecer.

Inicialización es la parte en la que la variable o variables de control del bucle toman su valor inicial. Puede haber una o más instrucciones en la inicialización, separadas por comas. La inicialización se realiza solo una vez.

Condición es una expresión booleana que hace que se ejecute la sentencia o bloque de sentencias mientras que dicha expresión sea cierta. Generalmente en la condición se compara la variable de control con un valor límite.

Incremento/decremento es una expresión que decrementa o incrementa la variable de control del bucle. La ejecución de un bucle for sigue los siguientes pasos:

1. Se inicializa la variable o variables de control (inicialización)

2. Se evalúa la condición.

3. Si la condición es cierta se ejecutan las instrucciones. Si es falsa, finaliza la ejecución del bucle y continúa el programa en la siguiente instrucción después del for.

4. Se actualiza la variable o variables de control (incremento/decremento)

5. Se vuelve al punto 2.

Ejemplo:

public class Ejemplo0For {

public static void main(String[] args) {

int i;

for(i=1; i<=10;i++)

System.out.println(i + " ");

}

}

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **/Proyecto/MiApp.Clientes** | | | | | |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
| **PSP0 Project Plan Summary** | | | |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
| Project Owner | Omar Moises Solano GonzÃ¡lez | | | | | | | | | |
| Start Date | ene 23, 2017 09:16:53 AM | | | |  |  |  |  |  |  |
| Completion Date | feb 12, 2017 12:32:07 PM | | | | Completed: \* | | | | | |
| Keywords |  | | | | | | | | | |
| Language |  | | | | | | | | | |
|  |  |  |  |  |  |  |  |  |  |  |
| **Time in Phase (min.)** | **Plan** | **Actual** | **To Date** | **To Date %** |  |  |  |  |  |  |
| Planning |  | 0:15 | 0:15 | 1.60% |  |  |  |  |  |  |
| Design |  | 0:25 | 0:25 | 2.66% |  |  |  |  |  |  |
| Code |  | 8:00 | 8:00 | 51.10% |  |  |  |  |  |  |
| Compile |  | 5:00 | 2:00 | 12.80% |  |  |  |  |  |  |
| Test |  | 6:00 | 5:00 | 31.90% |  |  |  |  |  |  |
| Postmortem |  | 3:00 | 0:00 | 0% |  |  |  |  |  |  |
| Total | 20:00 | 22:40 | 15:40 |  |  |  |  |  |  |  |
| **Defects Injected** |  | **Actual** | **To Date** | **To Date %** |  |  |  |  |  |  |
| Before Development |  | 0 | 0 | #DIV/0! |  |  |  |  |  |  |
| Planning |  | 0 | 0 | #DIV/0! |  |  |  |  |  |  |
| Design |  | 0 | 0 | #DIV/0! |  |  |  |  |  |  |
| Code |  | 0 | 0 | #DIV/0! |  |  |  |  |  |  |
| Compile |  | 0 | 0 | #DIV/0! |  |  |  |  |  |  |
| Test |  | 0 | 0 | #DIV/0! |  |  |  |  |  |  |
| Total Development |  | 0 | 0 |  |  |  |  |  |  |  |
| **Defects Removed** |  | **Actual** | **To Date** | **To Date %** |  |  |  |  |  |  |
| Planning |  | 0 | 0 | #DIV/0! |  |  |  |  |  |  |
| Design |  | 0 | 0 | #DIV/0! |  |  |  |  |  |  |
| Code |  | 0 | 0 | #DIV/0! |  |  |  |  |  |  |
| Compile |  | 0 | 0 | #DIV/0! |  |  |  |  |  |  |
| Test |  | 0 | 0 | #DIV/0! |  |  |  |  |  |  |
| Total Development |  | 0 | 0 |  |  |  |  |  |  |  |
| After Development |  | 0 | 0 |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
| *Adapted from "PSP Materials," copyright © 2006 Carnegie Mellon University. Used by permission.* | | | | | | | | |  |  |