Programación Java

Tutorial Java. Aprende a programar con Java desde cero.

Clases en Java

La clase es la unidad fundamental de programación en Java.

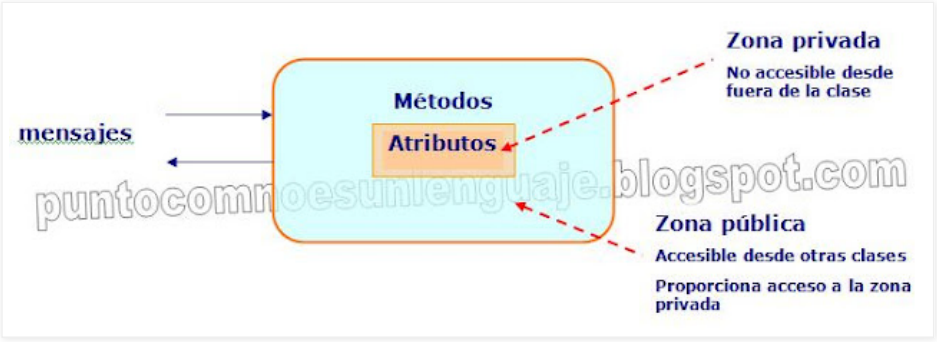
Un programa Java Orientado a Objetos está formado por un conjunto de clases. A partir de esas clases se crearán objetos que interactuarán entre ellos enviándose mensajes para resolver el problema.

Una clase representa al conjunto de objetos que comparten una estructura y un comportamiento comunes.

Puede considerarse como una *plantilla* o *prototipo* de objetos: define los atributos que componen ese tipo de objetos y los métodos que pueden emplearse para trabajar con esos objetos.

Las clases incluyen por tanto atributos y métodos. Los atributos definen el estado de cada objeto de esa clase y los métodos su comportamiento.

Los atributos debemos considerarlos como la zona más interna, oculta a los usuarios del objeto. El acceso a esta zona se realizará a través de los métodos.



La sintaxis general para definir una clase en Java es:

```
[modificadorDeAcceso] class NombreClase [extends NombreSuperClase][implements Interface1, Interfac

       //atributos de la clase (0 ó más atributos)
       [modificadorDeAcceso]  tipo nombreAtributo;


       //métodos de la clase (0 ó más métodos)
       [modificadorDeAcceso] tipoDevuelto nombreMetodo([lista parámetros]) [throws listaExcepcione
                       // instrucciones del método
                       [return valor;]
       }
}
```

Todo lo que aparece entre corchetes es opcional, por lo tanto la definición mínima de una clase en Java es:

```
class NombreClase{

}
```

Como hemos visto antes, el concepto de clase incluye la idea de ocultación de datos, que básicamente consiste en que no se puede acceder a los datos directamente (zona privada), sino que hay que hacerlo a través de los métodos de la clase.

PUBLICACIONES DEL BLOG

JAVA - Ejercicios básicos resueltos

ENTRADAS POPULARES

De esta forma se consiguen dos objetivos importantes:

- Que el usuario no tenga acceso directo a la estructura interna de la clase, para no poder generar código basado en la estructura de los datos.
- Si en un momento dado alteramos la estructura de la clase todo el código del usuario no tendrá que ser retocado.

El **modificador de acceso** se utiliza para definir el nivel de ocultación o visibilidad de los miembros de la clase (atributos y métodos) y de la propia clase.

Los modificadores de acceso **ordenados de menor a mayor visibilidad** son:

MODIFICADOR DE ACCESO	EFECTO	APLICABLE A
private	Restringe la visibilidad al **interior de la clase**. Un atributo o método definido como private solo puede ser usado en el interior de su propia clase.	Atributos Métodos
<Sin modificador>	Cuando no se especifica un modificador, el elemento adquiere el *acceso por defecto o friendly*. También se le conoce como acceso de package (paquete). Solo puede ser usado por las **clases dentro de su mismo paquete**.	Clases Atributos Métodos
protected	Se emplea en la herencia. El elemento puede ser utilizado por **cualquier clase dentro de su paquete y por cualquier subclase** independientemente del paquete donde se encuentre.	Atributos Métodos
public	Es el nivel máximo de visibilidad. El elemento es visible desde cualquier clase.	Clases Atributos Métodos

class: palabra reservada para crear una clase en Java.

NombreClase: nombre de la clase que se define. Si la clase es pública, el nombre del archivo que la contiene debe coincidir con este nombre. Debe describir de forma apropiada la entidad que se quiere representar. Los nombres deben empezar por mayúscula y si está formado por varias palabras, la primera letra de cada palabra irá en mayúsculas.

extends *NombreSuperclase*: (opcional) extends es la palabra reservada para indicar la herencia en Java. NombreSuperClase es la clase de la que hereda esta clase. Si no aparece extends la clase hereda directamente de una clase general del sistema llamada **Object.**

Object es la raíz de toda la jerarquía de clases en Java.

Es la superclase de las que heredan directa o indirectamente todas las clases Java.

Cuando una clase deriva de otra, se llama **superclase** a la clase base de la que deriva la nueva clase (clase derivada o subclase) **La clase derivada hereda todos los atributos y métodos de su superclase**.

implements *NombreInterface1*, *NombreInterface2*, ... : (opcional) implements es la palabra reservada para indicar que la clase implementa el o los interfaces que se indican separados por comas.

Un interface es un conjunto de constantes y declaraciones de métodos (lo que en C/C++ equivaldría al prototipo) no su implementación o cuerpo.

Si una clase implementa un interface está obligada a implementar todos los métodos de la interface. Veremos los interface más adelante.

- En Java solo puede haber una clase pública por archivo de código fuente .java
- El nombre de la clase pública debe coincidir con el nombre del archivo fuente. Por ejemplo, si el nombre de la clase pública es Persona el archivo será Persona.java
- En una aplicación habrá una clase principal que será la que contenga el método main. Esta clase deberá haber sido declarada como pública

Junto al modificador de acceso pueden aparecer **otros modificadores** aplicables a clases, atributos y métodos:

MODIFICADOR	EFECTO	APLICABLE A
abstract	Aplicado a una clase, la declara como clase abstracta. No se pueden crear objetos de una clase abstracta. Solo pueden usarse como superclases. Aplicado a un método, la definición del método se hace en las subclases.	Clases Métodos

final	Aplicado a una clase significa que no se puede extender (heredar), es decir que no puede tener subclases. Aplicado a un método significa que no puede ser sobrescrito en las subclases. Aplicado a un atributo significa que contiene un valor constante que no se puede modificar	Clases Atributos Métodos
static	Aplicado a un atributo indica que es una variable de clase. Esta variable es única y compartida por todos los objetos de la clase. Aplicado a un método indica que se puede invocar sin crear ningún objeto de su clase.	Atributos Métodos
volatile	Un atributo volatile puede ser modificado por métodos no sincronizados en un entorno multihilo.	Atributos
transient	Un atributo transient no es parte del estado persistente de las instancias.	Atributos
Sincronizad	Métodos para entornos multihilo.	Métodos

Como ejemplo, vamos a definir una clase Persona.

Clase Persona en Java:

```
public class Persona {

    private String nombre;
    private int edad;

    public void setNombre(String nom) {
        nombre = nom;
    }

    public String getNombre() {
        return nombre;
    }

    public void setEdad(int ed) {
        edad = ed;
    }

    public int getEdad() {
        return edad;
    }
}
```

La clase Persona es pública y tiene dos atributos, nombre y edad, y cuatro métodos.
Los métodos que aparecen en esta clase se conocen como métodos de acceso ó setters/getters.
Son métodos que solo sirven para asignar y obtener los valores de los atributos individualmente.
En cada clase es conveniente escribir un método set y otro get para cada atributo.

MIEMBROS DE UNA CLASE: ATRIBUTOS Y MÉTODOS

ATRIBUTOS

Una clase puede tener cero o más atributos.

Sirven para almacenar los datos de los objetos. En el ejemplo anterior almacenan el nombre y la edad de cada objeto Persona.

Se declaran generalmente al principio de la clase.

La declaración es similar a la declaración de una variable local en un método.

La declaración contiene un modificador de acceso de los vistos anteriormente: private, package, protected, public.

Pueden ser variables de tipo primitivo o referencias a objetos.

En la clase Persona se ha declarado edad de tipo primitivo y nombre String. Ambas private y por lo tanto solo accesibles desde los métodos de la propia clase.

```
private String nombre;
private int edad;
```

Los atributos toman el valor inicial por defecto:

- 0 para tipos numéricos
- '\0' para el tipo char
- null para String y resto de referencias a objetos.

También se les puede asignar un valor inicial en la declaración aunque lo normal es hacerlo en el constructor.

El valor de los atributos en cada momento determina el estado del objeto.

Podemos distinguir dos tipos de atributos o variables:

- **Atributos de instancia**: son todos los atributos no static. Cada objeto de la clase tiene sus propios valores para estas variables, es decir, cada objeto que se crea incluirá su propia copia de los atributos con sus propios valores.
- **Atributos de clase**: son los declarados **static**. También se llaman atributos estáticos. Un atributo de clase no es específico de cada objeto. Solo hay una copia del mismo y su valor es compartido por todos los objetos de la clase. Un atributo de clase existe y puede utilizarse aunque no existan objetos de la clase. Podemos considerarlo como una *variable global* a la que tienen acceso todos los objetos de la clase.

  Para acceder a un atributo de clase se escibe:

  NombreClase.Atributo;

  Por ejemplo, en la clase Persona podemos añadir un atributo contadorPersonas que indique cuantos objetos de la clase se han creado. Sería un atributo de clase ya que no es un valor propio de cada persona:

  static int contadorPersonas;

  Cada vez que se crea una persona podemos incrementar su valor:

  Personas.contadorPersonas++;

  Si lo declaramos además como private:

  private static int contadorPersonas

  solo podremos acceder al atributo a través de un método.

MÉTODOS

Una clase puede contener cero o más métodos.

Definen el comportamiento de los objetos de la clase.

A través de los métodos se accede a los datos de la clase.

Desde el punto de vista de la POO **el conjunto de métodos de la clase se corresponden con el conjunto de mensajes a los que los objetos de esa clase pueden responder**.

Al conjunto de métodos de una clase se le llama **interfaz** de la clase.

Es conveniente que todas las clases implementen los métodos de acceso ó setters/getters para cada atributo.

Los métodos pueden clasificarse en:

- **Métodos de instancia**: Son todos los métodos no static. Operan sobre las variables de instancia de los objetos pero también tienen acceso a los atributos estáticos.

  La sintaxis de llamada a un método de instancia es:

  idObjeto.metodo(parametros); // Llamada típica a un método de instancia

  Todas las instancias de una clase comparten la misma implementación para un método de instancia.

  Dentro de un método de instancia, el identificador de una variable de instancia hace referencia al atributo de la instancia concreta que hace la llamada al método (suponiendo que el identificador del atributo no ha sido *ocultado* en el método).

- **Métodos de clase**: Son los métodos declarados como **static**. Tienen acceso solo a los atributos estáticos de la clase. No es necesario instanciar un objeto para poder utilizar un método estático.

  Para acceder a un método de clase se escibe:

  NombreClase.metodo;

Por ejemplo para la clase Fecha podemos escribir un método estático que incremente el contador de personas:

```
public static void incrementarContador(){
    contadorPersonas++;
}
```

Para invocar a este método:

```
Persona.incrementarContador();
```

La API de Java proporciona muchas clases con métodos estáticos, por ejemplo, los métodos de la clase Math: Math.sqrt(), Math.pow(), etc.

40 comentarios:

Óscar Núñez Aguado 1 de febrero de 2022, 11:16

Excelente material! Como pequeña contribución te comento una pequeña errata en la 7ª línea empezando por el final: Donde dice "Por ejemplo para la clase Fecha podemos escribir un método estático que incremente el contador de personas:" debería ser "clase Persona", no?
Un saludo y muchas gracias por tu trabajo

Responder

TuSoftwareProfessional 12 de octubre de 2021, 1:58

Hola

Responder

Anónimo 21 de junio de 2021, 23:14

Hola gracias por el contenido, aprendí bastante, además que me resolvió ciertas dudas por ejemplo, lo que significa una superclase, el como se heredan las clases, que son las clases y métodos, eso entre otros conceptos.De nuevo te agradezco, porque aún, el tema es vigente para los que apenas aprendemos el lenguaje de programación en Java, muy interesante.

Responder

Armando 14 de enero de 2021, 18:31

Excelente blog, ayuda mucho yo lo estoy utilizando para mi formacion...

Responder

Anónimo 4 de julio de 2019, 3:03

Examen "Nombre apellido1 apellido2"

```
import java.io.*;
import java.util.*;
import java.util.regex.*;

public class Examen1{
public boolean getApellidos2(String readFile, String writeFile){
boolean done= false;
try{
Scanner input = new Scanner(new BufferedReader(new fileReader(readFile)));
PrintStream output = new PrintStream(writeFile);
Pattern p = Pattern.compile("\\b\\w+\\s+\\w+\\s+([A-Z]\\w+)\\b");
while(input.hasNextLine()){
Matcher m = p.matcher(input.nextLine());
if(m.find()){
output.println(m.group(1));
}
}
input.close();
output.close();
done=true;
}catch(FileNOtFoundException e){
done= false;
}
return done;

}
```

Examen "Leer de un fichero y obtener las palabras de longitud par y guardalas en otro fichero"

```
import java.io.*;
import java.util.*;
import java.util.regex.*;

public class Examen2{
private boolean method(String m){
if(m.length()%2==0){
return true;
}else{
return false;
}
}

public boolean getEvenWords(String readFile, String writeFile){
boolean done = false;
try{
Scanner input = new Scanner(new BufferedReader(new FileReader)(readFile)));
PrintStream output = new PrintStream(writeFile);
String[] parts = writeFile.split("-");

while(input.hasNextLine()){
String line = input.nextLine();
String[] word = line.split("");
String res="";


for(String m: word){
if(method(m)==true){
res+=m+"";
}
}
output.println(res.trimm());
}
input.close();
output.close();
done=true;
}catch(Exception e){
done = false;
}
return done;
}

}
```

Examen 3"El de los 30 amigos"

```
import java.io.*;
import java.util.*;
import java.util.regex.*;

public class Examen3{
public bollean getShortest(String readFile, String writeFile){
boolean done = false;
String res="";

try{
Scanner input = new Scanner(new BufferedReader(new FileReader(readFile)));
PrintStream output = new PrintStream (writeFile);
while(input.hasNextLine()){
Matcher m = p.marcher(input.nextLine());
if(m.find()){
res = m.group();
}else{
res = "";
}
while(m.find()){
if(m.group().length()<=res.legth()){
res = m.group();
}
}
output.println(res);
}
input.close();
output.close();
done=true;
}catch(FileNotFoundException e){
done = false;
}
return done;
}
}
```

Responder

Anónimo 6 de abril de 2019, 20:58

Hola, alguien me puede ayudar con un programa similar?

Responder

Anónimo 26 de abril de 2018, 9:40

```
package es.ulpgc.eii.containers.unbounded;

public class ExtendedQueueOfInt extends QueueOfInt
implements Comparable, Cloneable{

@Override
public int compareTo(QueueOfInt arg0) {
ExtendedQueueOfInt queso = (ExtendedQueueOfInt) arg0;
Node aux3= this.front;
Node aux4= queso.front;
while (aux3!=null && aux4!=null){
if (aux3.info>aux4.info){
return -1;
}if (aux3.info<aux4.info){
return 1;
}if (aux3.info==aux4.info){
aux3= aux3.next;
aux4= aux4.next;
}
}if (aux3==null && aux4!=null){
return 1;
}if (aux4==null && aux3!=null){
return -1;
}if (aux3.info==aux4.info && aux3.next==null && aux3.next==null) {
return 0;
}
return 0;
}

@Override
public boolean equals(Object arg0) {
ExtendedQueueOfInt helado = (ExtendedQueueOfInt) arg0;
Node aux= this.front;
Node aux22= helado.front;
if (front==null && helado.front==null){
return true;
}
while (aux != null ||aux22 != null){
if (aux==null||aux22==null||aux.info != aux22.info){
return false;
}
aux= aux.next;
aux22= aux22.next;
}
return true;
}

@Override
public String toString() {
String result="<";
Node aux= front;
if (front!=null){
while (aux.next != null){
result= result + aux.info + "-";
aux= aux.next;
}
result= result + aux.info;
}
result = result + "<";
return result;
}

@Override
public ExtendedQueueOfInt clone() {
ExtendedQueueOfInt clon = new ExtendedQueueOfInt();
Node principio= front;
if (front==null){
return clon;
}else{
while(principio!=null){
clon.insert (principio.info);
principio = principio.next;
}
}
return clon;
}
}
```

Responder

Anónimo 25 de abril de 2018, 18:08

```
package es.ulpgc.eii.containers.unbounded;

public class ExtendedQueueOfInt extends QueueOfInt
implements Comparable, Cloneable{

@Override
public int compareTo(QueueOfInt q) {
Node n1 = front;
Node n2 = q.front;
while (n1 != null && n2 != null) {
if (n1.info > n2.info) {
return -1;
} else {
if (n1.info < n2.info) {
return 1;
} else {
n1 = n1.next;
n2 = n2.next;
}
}
}
if (n1 == null && n2 != null) {
return -1;
} else {
if (n1 != null && n2 == null) {
return 1;
}
}
return 0;
}

@Override
public boolean equals(Object obj) {
ExtendedQueueOfInt q = (ExtendedQueueOfInt) obj;
Node n1 = front;
Node n2 = q.front;
while (n1 != null && n2 != null) {
if (n1.info > n2.info || n1.info < n2.info) {
return false;
} else {
n1 = n1.next;
n2 = n2.next;
}
}
if (n1 == null && n2 != null || n1 != null && n2 == null) {
return false;
} else {
return true;
}
}

@Override
public String toString() {
Node n = front;
if (front == null) {
return "<<";
} else {
String res = "<";
while (front != null) {
res += n.info + "-";
n = n.next;
}
return res.substring(0, res.length()-1) + "<";
}
}

@Override
public ExtendedQueueOfInt clone() {
ExtendedQueueOfInt res = new ExtendedQueueOfInt();
Node n1 = front;
while (n1 != null) {
res.insert(n1.info);
n1 = n1.next;
}
return res;
}
```

Responder

Anónimo 25 de abril de 2018, 18:08

ou

Responder

Anónimo 25 de abril de 2018, 18:04

hola

Responder

Respuestas

Responder

Anónimo 25 de abril de 2018, 18:08

```
@Override
public int compareTo(QueueOfInt q) {
Node n1 = front;
Node n2 = q.front;
while (n1 != null && n2 != null) {
if (n1.info > n2.info) {
return -1;
} else {
if (n1.info < n2.info) {
return 1;
} else {
n1 = n1.next;
n2 = n2.next;
}
}
}
if (n1 == null && n2 != null) {
return -1;
} else {
if (n1 != null && n2 == null) {
return 1;
}
}
return 0;
}

@Override
public boolean equals(Object obj) {
ExtendedQueueOfInt q = (ExtendedQueueOfInt) obj;
Node n1 = front;
Node n2 = q.front;
while (n1 != null && n2 != null) {
if (n1.info > n2.info || n1.info < n2.info) {
return false;
} else {
n1 = n1.next;
n2 = n2.next;
}
}
if (n1 == null && n2 != null || n1 != null && n2 == null) {
return false;
} else {
return true;
}
}

@Override
public String toString() {
Node n = front;
if (front == null) {
return "<<";
} else {
String res = "<";
while (front != null) {
res += n.info + "-";
n = n.next;
}
return res.substring(0, res.length()-1) + "<";
}
}

@Override
public ExtendedQueueOfInt clone() {
ExtendedQueueOfInt res = new ExtendedQueueOfInt();
Node n1 = front;
while (n1 != null) {
res.insert(n1.info);
n1 = n1.next;
}
return res;
}
```

Vera Ascate 15 de noviembre de 2017, 2:49

alguien que me ayude...
Una Editorial de Libros y Discos desea crear fichas que almacen en titulo y precio de cada publicación. Crear la clase denominada (Publicacion) que tenga los atributos anteriores. A partir de esta clase diseñar 2 clases derivadas: Libro con atributos adicionales número de páginas y año de Publicación, Disco con atributos adicionales duración en Minutos. Escribir un Programa usando un Menú que solicite los datos de Libro y Disco y luego visualice los datos Ingresados.

Responder

Faquu 24 de junio de 2017, 20:02

Muy buena info!

Responder

Respuestas

Responder

Enrique 7 de julio de 2017, 11:27

Gracias Faquu!

Raphael-25 15 de mayo de 2017, 6:43

Muy buena tu información.

Responder

Respuestas

Responder

Enrique 7 de julio de 2017, 11:26

Gracias Raphael!

Unknown 9 de enero de 2017, 23:01

Excelente la explicación, mejor que en los libros.

Responder

Respuestas

Responder

Enrique 10 de enero de 2017, 23:47

Muchas gracias!! Espero que la explicación te sea de ayuda para entender los objetos en Java

Unknown 16 de noviembre de 2016, 4:06

```
public class modificacion {
public static void main(String[] args) {
double [][] A = {{6,4,-11},{4,2,10},{-6,-2,-7}};
System.out.println("La matriz A original es:");
visualizar(A);
double [][] B;
B=modificar_A(A);
System.out.println("La matriz B es:");
visualizar(B);
System.out.println("La matriz A despues de modificar es:");
visualizar(A);

}
static double [][] modificar_A(double [][] A){
for (int i = 0; i < A.length; i++) {
for (int j = 0; j < A[0].length; j++) {
A[i][j]=A[i][j]*0.5;
}
}
return A;
}
static void visualizar(double [][] A){
for (int i = 0; i < A.length; i++) {
for (int j = 0; j < A[0].length; j++) {
System.out.printf("%8.2f",A[i][j]);
}
System.out.println("");
}
}
}
```

La matriz A original es:
6,00 4,00 -11,00
4,00 2,00 10,00
-6,00 -2,00 -7,00
La matriz B es:
3,00 2,00 -5,50
2,00 1,00 5,00
-3,00 -1,00 -3,50
La matriz A despues de modificar es:
3,00 2,00 -5,50
2,00 1,00 5,00
-3,00 -1,00 -3,50
BUILD SUCCESSFUL (total t

quería saber por que pasa eso amigo, me puedes explicar?, yo pensaba que A no se iba a ver afectada por el método, pero veo lo contrario.

Responder

Respuestas

Responder

Unknown 16 de noviembre de 2016, 4:12

Y porque cuando hago esto tampoco me resulta:
static double [][] modificar_A(double [][] A){
double [][] B = A;
for (int i = 0; i < A.length; i++) {
for (int j = 0; j < A[0].length; j++) {
B[i][j]=B[i][j]*0.5;
}
}
return B;
}
la salida es
La matriz A original es:
6,00 4,00 -11,00
4,00 2,00 10,00
-6,00 -2,00 -7,00
La matriz B es:
3,00 2,00 -5,50
2,00 1,00 5,00
-3,00 -1,00 -3,50
La matriz A despues de modificar es:
3,00 2,00 -5,50
2,00 1,00 5,00
-3,00 -1,00 -3,50

Pero amigo cuando resolví el método aplicando esto si me resultó:
static double [][] modificar_A(double [][] A){
double [][] B = new double [A.length][A.length];
for (int i = 0; i < A.length; i++) {
for (int j = 0; j < A[0].length; j++) {
B[i][j]=A[i][j];
}
}
for (int i = 0; i < A.length; i++) {
for (int j = 0; j < A[0].length; j++) {
B[i][j]=B[i][j]*0.5;
}
}
return B;
}
La matriz A original es:
6,00 4,00 -11,00
4,00 2,00 10,00
-6,00 -2,00 -7,00
La matriz B es:
3,00 2,00 -5,50
2,00 1,00 5,00
-3,00 -1,00 -3,50
La matriz A despues de modificar es:
6,00 4,00 -11,00
4,00 2,00 10,00
-6,00 -2,00 -7,00

Te sugiero que me respondas por cada caso, es decir al comienzo luego el segundo caso de B=A, y el último

Unknown 10 de noviembre de 2016, 20:31

Gracias!!

Responder

Unknown 19 de febrero de 2016, 22:55

Estoy haciendo un curso de Java en Aula Mentor(proyecto del ministerio de educación de España) y siempre tengo que venir aquí a buscar información...
Creo que eso lo dice todo sobre la calidad de la teoría de Aula Mentor y la de esta página...

Felicidades y sobre todo...gracias!!

Responder

Respuestas

Responder

Anónimo 5 de mayo de 2023, 6:06

```
--vent princ
private void jButton_mostrarCantFacturasActionPerformed(java.awt.event.ActionEvent evt) {
// TODO add your handling code here:
//------- Implementar la lógica correspondiente -----------
OperacionesCRUD obj_BD = OperacionesCRUD.getInstance();
try
{
int cantFacturas = obj_BD.obtenerCantFacturasGeneradas();
String cant = String.valueOf(cantFacturas);
jTextField_cantFacturas.setText(cant);
}
catch(SQLException err)
{
System.out.println(err.getMessage());
}
//-----------------------------------------------------------------------------------
}
```

Anónimo 5 de mayo de 2023, 6:07

```
//OK
private void jButton_mostrarListadoProductosStockActionPerformed(java.awt.event.ActionEvent evt) {
// TODO add your handling code here:
//------- Implementar la lógica correspondiente -------------------------
//1. Obtener la instancia única objeto de la clase OperacionesCRUD
OperacionesCRUD obj_BD = OperacionesCRUD.getInstance();
try
{
//2. Obtener el vector lista con los códigos de facturas del método que consulta a la BD y que implementa el objeto obj_BD (OperacionesCRUD)
Vector listadoNombresProductosStock = obj_BD.obtenerListadoNombresProductosStock();
System.out.println(listadoNombresProductosStock.toString());
//3. Crear el objeto DefaultTableModel del jTable_ListadoNombresProductosStock que debe mostrar el vector lista de códigos de facturas
DefaultTableModel tablaVisual = (DefaultTableModel) jTable_ListadoNombresProductosStock.getModel();
//4. Setear el componente jTable_ListadoNombresProductosStock, desde el objeto DefaultTableModel, con cero filas y cero columnas de datos a mostrar.
tablaVisual.setRowCount(0);
tablaVisual.setColumnCount(0);

//5. Agregar, desde el objeto DefaultTableModel, la columna de datos con cabecera "Descripcion Productos en Stock" y vector lista de datos con los códgios de las facturas generadas.
tablaVisual.addColumn("Descripcion Productos en Stock",listadoNombresProductosStock );
}
catch(SQLException err)
{
err.printStackTrace();
}
```

Anónimo 5 de mayo de 2023, 6:08

```
private void jButton_mostrarDetallesFacturaActionPerformed(java.awt.event.ActionEvent evt) {
// TODO add your handling code here:
//------- Implementar la lógica correspondiente -------------------------
//1. Obtener la instancia única objeto de la clase OperacionesCRUD
OperacionesCRUD obj_BD = OperacionesCRUD.getInstance();
//2. Obtener la lista de valores del modelo del jComboBox_listadoCodigosFacturas con los datos de códigos de las facturas generadas.
DefaultComboBoxModel recordList_Carr_Est = (DefaultComboBoxModel)jComboBox_listadoCodigosFacturas.getModel();
//3. Obtener el código de factura seleccionado desde el combobox jComboBox_listadoCodigosFacturas
String codigo = (String) recordList_Carr_Est.getSelectedItem();
try
{
```

```
//4. Obtener la matriz de datos del detalle de productos de la factura que devuelve el método que consulta a la BD y que implementa el objeto obj_BD (OperacionesCRUD)
ArrayList> matrizDetallesProductosFactura = obj_BD.obtenerDetallesProductosXFactura(codigo);
//5. Crear el objeto DefaultTableModel del jTable_detallesProductosFactura que debe mostrar la matriz de datos del detalle de la factura seleccionada para filtrar.
DefaultTableModel tablaVisual = (DefaultTableModel) jTable_detallesProductosFactura.getModel();

//6. Setear, desde el objeto DefaultTableModel, el componente jTable_detallesProductosFactura con cero filas a mostrar.
tablaVisual.setRowCount(0);
//7. Obtener la cantidad de filas que arroja la consulta a la base de datos almacenada en la matriz de datos.
int total = matrizDetallesProductosFactura.size();
//8. Recorrer cada fila de la matriz de datos con el detalle de productos de la factura y para cada iteración resolver:
for (int i=0; i<total; i++){
//8.1 En cada iteración, agregar al jTable_detallesProductosFactura, desde el objeto DefaultTableModel, los datos de cada una de las filas que están almacenadas en la matriz de datos.
Vector Fila = (Vector) matrizDetallesProductosFactura.get(i);
tablaVisual.addRow(Fila);
}
}
catch(SQLException err)
{
err.printStackTrace();
}
```

Anónimo 5 de mayo de 2023, 6:10

```
Conexion:
public class ConexionBD
{
//jdbc:oracle:thin:@//::
private static final String DB_URL = "jdbc:oracle:thin:@//localhost:1521/xe";
private static final String DB_USER = "FACTURACION";
private static final String DB_PASSWORD = "itb";

public static Connection iniciarConexion()
{

CONSULTAS
--El total de facturas registradas hasta el momento en el sistema
select
count (*) as total_facturas
from
facturas

--El listado con los nombres de productos en stock para la venta
select
s.descripcion as nombre_pro
from
stock_productos s

--El detalle de los productos de las facturas generadas con los datos de nombre del
---producto, cant solicitada y precio total por cada producto de la factura, a filtrar por el --código de la factura
select
s.descripcion as nombre_prod,
d.cantidad_prod as cantidad_pro,
d.precio_total_prod as precio_total
from
stock_productos s, detalle_facturas d
where
d.codigo_factura = '& codigo_factura'
--El detalle de los
select s.descripcion, d.cantidad_prod, d.precio_total_prod
from DETALLE_FACTURAS d join stock_productos s on( s.codigo = d.codigo_producto)
where d.CODIGO_FACTURA = (CodigoFactura)
```

Anónimo 9 de julio de 2015, 6:53

5

Responder

Respuestas

Responder

Anónimo 5 de mayo de 2023, 6:04

```
//Inciso c
public ArrayList> obtenerDetallesProductosXFactura(String pCodigoFactura) throws SQLException
{
//1. Conectar a la base de datos
```

```
this.iniciarConexionBD();
//2. Crear la matriz de datos para almacenar el detalle de los productos de la factura segun la consulta SQL, cada fila es un Vector de String
ArrayList> matrizDetallesProductosFactura = new ArrayList<>();
//------------------- COMPLETAR LA LÓGICA CORRESPONDIENTE ----------------------------------------------------------------------------------
//3. definir espacio de trabajo para la declaración y ejecución de la consulta sql
Statement stm = this.conexion.createStatement();
//4. Definir el texto String de la consulta SQL.
String sql = "SELECT s.descripcion , d.cantidad_prod , d.precio_total_prod \n" +
"FROM DETALLE_FACTURAS d JOIN stock_productos s ON( s.codigo = d.codigo_producto)\n" +
"WHERE d.CODIGO_FACTURA = '"+pCodigoFactura+"'";
//5. Ejecutar la consulta y amacenar en el objeto ResultSet
ResultSet tabla = stm.executeQuery(sql);
//6. Recorrer el objeto ResultSet mediante un while y para cada iteración resolver:
while (tabla.next()){
//En cada iteración:
//6. Crear el Vector fila para almacenar los datos de cada fila según los campos de columnas de la consulta SQL.
Vector filaDatos = new Vector <>();
//6.1 Agregar al vector fila el dato del campo descripcion de la consulta SQL
String Descripcion_Producto = tabla.getString("descripcion");
//6.2 Agregar al vector fila el dato del campo cantidad_prod de la consulta SQL
String Cantidad_Producto = tabla.getString("cantidad_prod");
//6.3 Agregar al vector fila el dato del campo precio_total_prod de la consulta SQL
String Precio_Producto = tabla.getString("precio_total_prod");
//6.4 Agregar el ventor fila, con los datos de las tres columnas de la consulta SQL, a la tabla matriz de datos.
filaDatos.add(Descripcion_Producto);
filaDatos.add(Cantidad_Producto);
filaDatos.add(Precio_Producto);
matrizDetallesProductosFactura.add(filaDatos);
}
//-------------------------------------------------------------------------------------------------------------------------------------
//7. Cerrar la conexion a la base de datos
this.cerrarConexionBD();
//8. Retornar la matriz de datos con los resultados de la consulta SQL.
return matrizDetallesProductosFactura;
}
}
```

Unknown 7 de mayo de 2015, 23:39

Muchas Gracias

Responder

Alberto 19 de abril de 2015, 11:37

Te felicito por la claridad de las explicaciones y te agradezco compartas tu conocimiento. Muchos éxitos

Responder

Respuestas

Responder

Enrique 23 de abril de 2015, 19:24

Gracias Alberto por el comentario. Me alegro mucho de que te resulte útil el blog.
Un saludo.

Anónimo 5 de mayo de 2023, 6:03

```
//Inciso b
public Vector obtenerListadoNombresProductosStock() throws SQLException
{
//1. Conectar a la base de datos
this.iniciarConexionBD();
//2. Crear el vector para almacenar la lista de nombres de productos en stock resultante de la consulta SQL a la base de datos.
Vector listadoNombresProductosStock = new Vector<>();

//3. definir espacio de trabajo para la declaración y ejecución de la consulta sql
Statement stm = this.conexion.createStatement();
//4. Definir el texto String de la consulta SQL.
String sql = "select descripcion from stock_productos p";
//5. Ejecutar la consulta y amacenar en el objeto ResultSet
ResultSet tabla = stm.executeQuery(sql);
//6. Recorrer el objeto ResultSet mediante un while y para cada iteración resolver:
while (tabla.next()){
//En cada iteración:
//6.1. Obtener el nombre del producto de la fila actual del objeto ResultSet
```

```
String lista = tabla.getString("descripcion");
//6.2. Almacenar en el vector creado en el paso 2. el valor de nombre de producto de la fila actual del objeto ResultSet
listadoNombresProductosStock.add(lista);
}
//-------------------------------------------------------------------------------------------------------------------
//7. Cerrar la conexion a la base de datos
this.cerrarConexionBD();
//8. Retornar el objeto vector con la lista de nombres de carreras almacenados en la base de datos
return listadoNombresProductosStock;
}
```

Anónimo 12 de marzo de 2015, 0:24

Igual llego tarde... pero muchas gracias por el blog es de gran ayuda y está explicado perfectamente

Responder

Respuestas

Responder

Anónimo 5 de mayo de 2023, 6:01

--oper CRUD

```
//Inciso a
public int obtenerCantFacturasGeneradas() throws SQLException
{
//1. Conectar a la base de datos
this.iniciarConexionBD();
//2. Variable para almacenar el total de facturas almacenadas en la base de datos.
int cantFacturas = 0;
//------------------- COMPLETAR LA LÓGICA CORRESPONDIENTE ---------------------------------------------------
//3. definir espacio de trabajo para la declaración y ejecución de la consulta sql
Statement stm = this.conexion.createStatement();
//4. Definir el texto String de la consulta SQL.
String sql = "select count(*) as Total_Facturas from facturas" ;
//5. Ejecutar la consulta y amacenar en el objeto ResultSet
ResultSet tabla = stm.executeQuery(sql);

//6. Recorrer el objeto ResultSet mediante un while y para cada iteración resolver:
while(tabla.next())
{
//6.1 En la única iteración, obtener y alamcenar el dato de la cantidad de facturas que arrojó la consulta SQL
cantFacturas = tabla.getInt("Total_Facturas");
}
//-------------------------------------------------------------------------------------------------------------------
//7. Cerrar la conexion a la base de datos
this.cerrarConexionBD();
//8. Retornar el total de carreras que está almancenado en la base de datos
return cantFacturas;
}
```

Jordi 26 de julio de 2014, 12:28

Muy bueno, quise aprender Java con un librote enorme y al final he recurrido a este blog, muchas veces, para aclarar lo del libro.

Responder

Respuestas

Responder

Enrique 29 de julio de 2014, 0:33

Gracias Jordi. Espero que sigas visitando el blog y que te siga siendo de ayuda.

Anónimo 24 de octubre de 2013, 22:56

Enrique Gracias por el magnifico aporte que has hecho con todo este blog que has creado de java la verdad que bien que personas como vos que tienen este conocimiento lo compartan para los demas!! felicitaciones y espero que tengas exitos!

Responder

Respuestas

Responder

Enrique 4 de noviembre de 2013, 18:38

Gracias tavoqg por el comentario y por seguir el blog. Un saludo.

Anónimo 2 de octubre de 2013, 14:55

Buen aporte, expresado con tus propias palabras. :D

Responder

Respuestas

Responder

Enrique 24 de octubre de 2013, 18:08

Gracias ! ;)

Escribe tu comentario

Entrada más reciente Inicio Entrada antigua

Suscribirse a: Enviar comentarios (Atom)