Linkia FP

Formación Profesional Oficial a Distancia

Linkia FP

DAW – M03 – Clase 11

Repaso global UF4, UF5, UF6

Linkia FP

Contenidos

- UF4: POO, Herencia.
- UF5: Estructuras de datos avanzadas, Excepciones, Swing, Serialización y XML.
- UF6: JDBC, db4o.

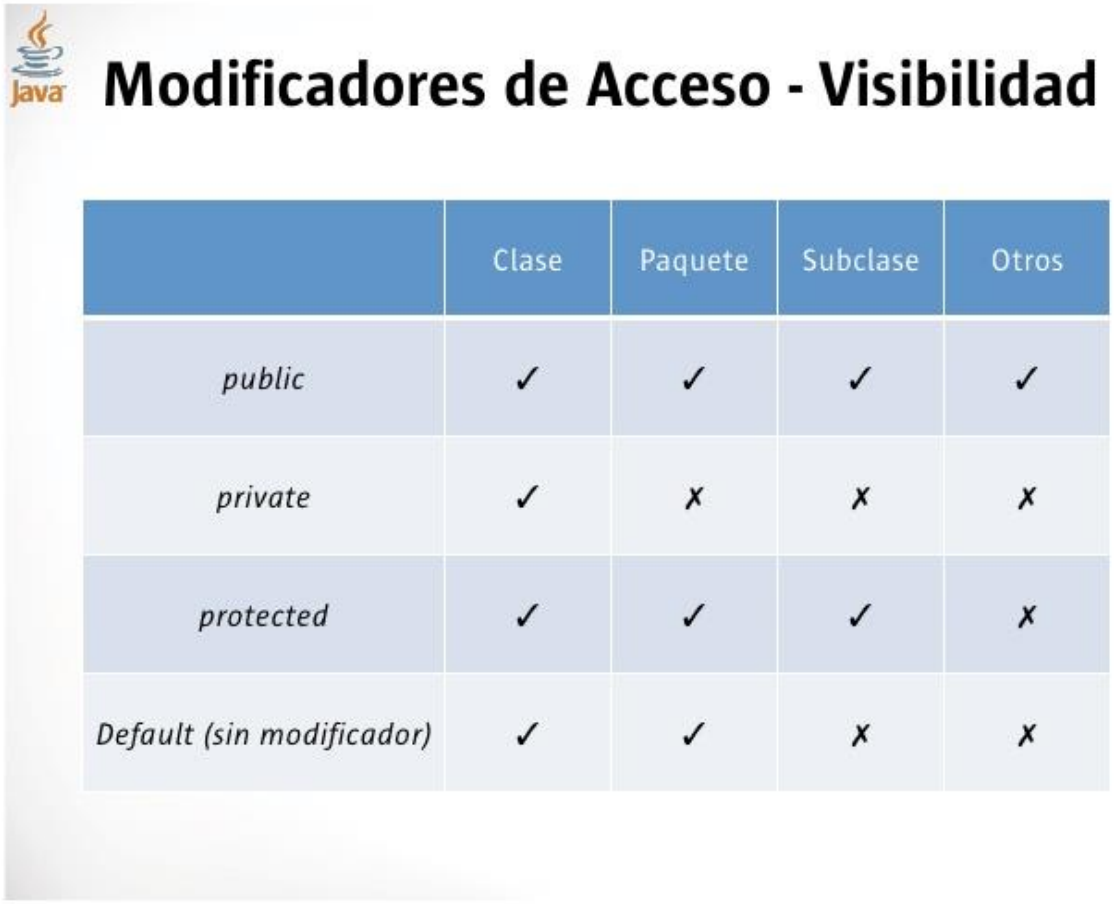
UF4 POO

- Las clases son los moldes de los cuales se generan los objetos.
- Es una instancia de una clase.
- El constructor del objeto es un procedimiento llamado automáticamente cuando se crea un objeto de esa clase.

UF4 POO

- En Java no hay destructores, ya que la liberación de memoria es llevada acabo por el Garbage Collector cuando las instancias de los objetos quedan desreferenciadas.
- Los atributos y métodos estáticos también se llaman atributos de clase y métodos de clase. Se declaran como static.

UF4 Modificadores de acceso



Modificadores de Acceso - Visibilidad

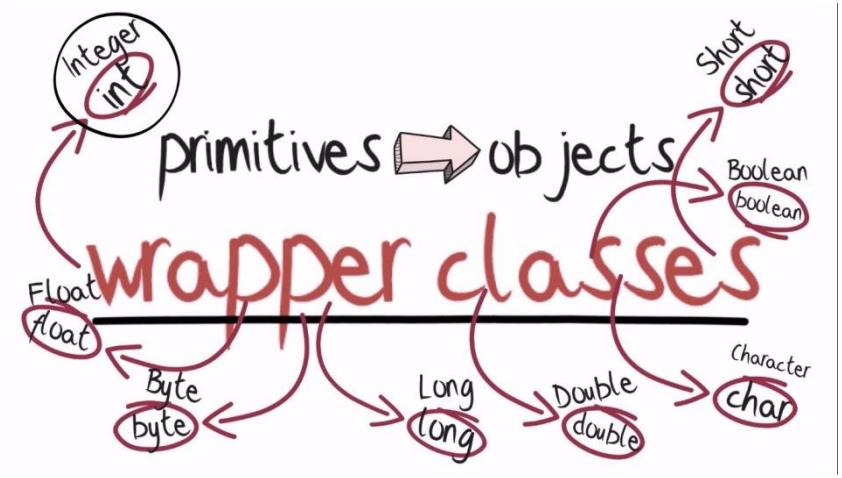
	Clase	Paquete	Subclase	Otros
public	✓	✓	✓	✓
private	✓	✗	✗	✗
protected	✓	✓	✓	✗
Default (sin modificador)	✓	✓	✗	✗

UF4 Métodos de la clase Object

- getClass(): Devuelve la clase del objeto.
- toString(): Representación textual de un objeto.
- clone(): Permite clonar un objeto.
- equals(): Permite comprobar si dos objetos son iguales.
- hashCode(): Código hash utilizado en las colecciones.
- finalize(): Método invocado por el recolector de basura (garbage collector) para borrar definitivamente un objeto.

UF4 Wrappers

- Las clases envoltorio existentes son:
  - Byte para byte.
  - Short para short.
  - Integer para int.
  - Long para long.
  - Boolean para boolean
  - Float para float.
  - Double para double y
  - Character para char.



UF4 Herencia

- La herencia es una de las características básicas de la Programación Orientada a Objetos.
- Es una de las bases que permite cosas tales como la reutilización del código, especialización, o evolución.
- La herencia es el mecanismo que:
  - sirve de soporte para registrar y utilizar las relaciones conceptuales existentes entre las clases
  - posibilita la definición de una clase a partir de otra

UF4 Herencia

- Al heredar es posible redefinir los métodos para adaptarlos a la semántica de la nueva clase.
- Los atributos no se pueden redefinir, sólo se ocultan.
  - Si la clase hija define un atributo con el mismo nombre que un atributo de la clase padre, éste no está accesible.
- Un método de la subclase con la misma signatura (nombre y parámetros) que un método de la superclase lo está redefiniendo.
  - Si se cambia el tipo de los parámetros se está sobrecargando el método original.

UF4 Herencia

- Las clases abstractas son clases que han sido pensadas para ser genéricas.
- Toda clase que contenga algún método abstracto (heredado o no) es abstracta.
- Una clase puede ser abstracta y no contener ningún método abstracto.
- Por defecto, toda definición de una interfaz es pública.
- Sólo contiene definiciones de métodos y constantes.
  - Los métodos son abstractos, no es necesario especificarlo explícitamente.

UF4 Polimorfismo

- El polimorfismo en POO permite abstraer y programar de forma general agrupando objetos con características comunes y jerarquizándolos en clases.
- Ejemplo: (Empleado hereda de Persona):

```
Empleado e = new Empleado("Fulanito", 24);
Persona p = e;
```

UF5 ArrayList

- De forma general:

```
ArrayList<tipo> nombreArray = new ArrayList<tipo>();
```

- tipo debe ser una clase. Indica el tipo de objetos que contendrá el array.
- No se pueden usar tipos primitivos. Para un tipo primitivo se debe utilizar su clase contenedora.
- Por ejemplo:

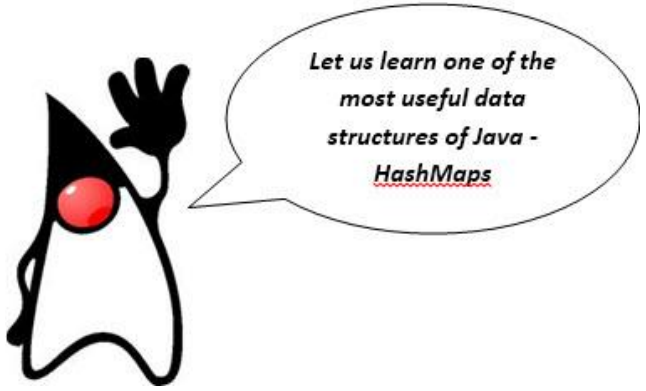
```
ArrayList<Integer> numeros = new ArrayList<Integer>();
```

- Crea el array numeros de enteros.

UF5 ArrayList

Método	Descripción
size()	Devuelve el número de elementos (int)
add(x)	Añade el objeto x al final. Devuelve true.
add(posicion, x)	Inserta el objeto x en la posición indicada.
get(posicion)	Devuelve el elemento que está en la posición indicada.
remove(posicion)	Elimina el elemento que se encuentra en la posición indicada. Devuelve el elemento eliminado.
remove(x)	Elimina la primera ocurrencia del objeto x. Devuelve true si el elemento está en la lista.
clear()	Elimina todos los elementos.
set(posición, x)	Sustituye el elemento que se encuentra en la posición indicada por el objeto x. Devuelve el elemento sustituido.
contains(x)	Comprueba si la colección contiene al objeto x. Devuelve true o false.
indexOf(x)	Devuelve la posición del objeto X. Si no existe devuelve -1

UF5 HashMap



- Un HashMap es una implementación de la interfaz Map de Java.
- Es una colección de objetos.
- El HashMap nos permite almacenar pares clave/valor, de tal manera que una clave sólo puede tener un valor, es decir, las claves no se repiten.

UF5 HashMap

- Si añadimos un elemento cuya clave ya existe, no se generará un nuevo elemento en el HashMap, sino que se reescribe el valor que pertenece a la clave existente.
- Hay que tener en cuenta que el HasMap no admite valores nulos.
- Para usar HashMap en nuestro programa, tendremos que importar la librería java.util.HashMap.

UF5 HashMap

Método	Descripción
clear()	Vacía el HashMap
containsKey(key)	Indica si el HashMap contiene un elemento con la clave key.
containsValue(value)	Indica si el HashMap contiene un elemento con el valor value.
entrySet()	Devuelve una copia del HashMap en forma de colección. Cualquier cambio en la colección afecta directamente al HashMap.
get(key)	Devuelve el valor asociado a la clave key, o nulo si no existe la clave.
isEmpty()	Indica si el HashMap está vacío.
keySet()	Devuelve una colección con las claves del HashMap.

UF5 HashMap

Método	Descripción
put(key, value)	Añade un elemento al HashMap con la clave key y el valor value. Si ya existe un elemento con clave key, sustituye su valor a value.
putAll(map)	Copia todos los elementos de map en el HashMap que invoca el método.
remove(key)	Elimina el elemento con clave key del HashMap si existe.
size()	Devuelve un número entero que es el número de elementos que tiene el HashMap.
values()	Devuelve una colección con los valores que contiene el HashMap.

UF5 Excepciones

- Una excepción en términos de lenguaje de programación es la indicación de un problema que ocurre durante la ejecución de un programa.
- La palabra excepción se refiere a que este problema ocurre con poca frecuencia generalmente cuando existe algún dato o instrucción que no se apega al funcionamiento del programa por lo que se produce un error.
- El control de excepciones permite al programador controla la ejecución del programa evitando que éste falle de forma inesperada.

UF5 Estructura de las excepciones

```
try {
      sentencias a proteger
} catch (expepcion_1) {
      control de excepción 1
}
…
} catch (excepcion_n) {
      control de excepción n
} finally {
      control opcional
}
```

UF5 Exepciones propias

- El programador puede crear sus propias clases de excepciones para tratar errores específicos que Java no contempla.
- Para ello hay que usar el concepto de herencia sobre la clase Exception.
- Normalmente, la nueva clase contendrá un único constructor con un parámetro de tipo String, que será el mensaje de error que queremos mostrar.
- Este mensaje lo mostraremos con el método getMessage() de la clase Exception

UF5 Swing

- Swing es una biblioteca gráfica para Java. Incluye widgets para interfaz gráfica de usuario tales como cajas de texto, botones, desplegables y tablas.

UF5 Swing

Componente	Descripción
JButton	Botón estándar
JLabel	Etiqueta de texto estándar
JTextField	Cuandro de texto
JTextArea	Cuadro de texto multilínea
JCheckBox	Checkbox o casilla de verificación
JRadioButton	Radiobutton o botones de opción
JComboBox	Lista desplegable
JScrollBall	Barra de desplazamiento

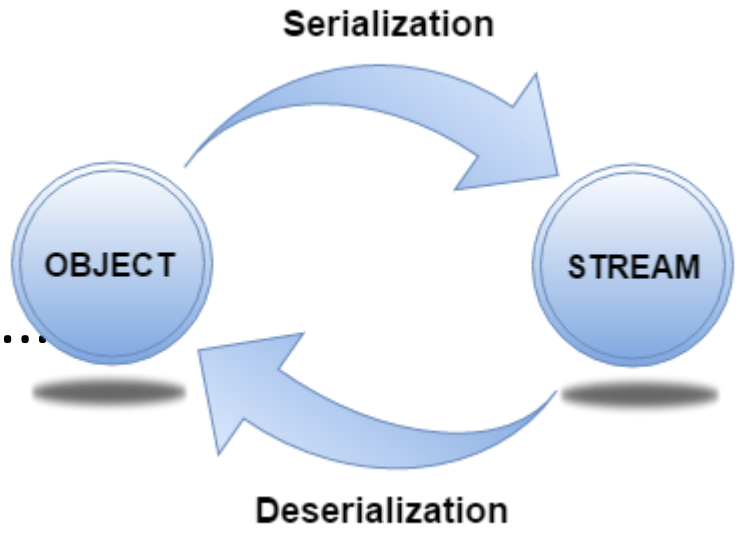
UF5 Serialización

- Para que un programa java pueda convertir un objeto en un montón de bytes y pueda luego recuperarlo, el objeto necesita ser **Serializable**.
- Al poder convertir el objeto a bytes, ese objeto se puede enviar a través de red,

  guardarlo en un fichero, y

  después reconstruirlo al otra

  lado de la red, leerlo del fichero,...



Escritura con ObjectOutputStream

- Para crear el fichero usaremos la clase **File**.
- En la escritura haremos uso de la clase ObjectOutputStream. Esta clase nos ofrece el método writeObject() que nos guardará el objeto con sus propiedades previamente construido. Al finalizar la escritura cerramos el output.

```
File archivo = new File("archivo");
ObjectOutputStream oos;
oos = new ObjectOutputStream(new FileOutputStream(archivo));
oos.writeObject(objetoDeLaClase);
oos.close();
```

Lectura con ObjectInputStream

- En la lectura haremos uso de la clase ObjectInputStream. Esta clase nos ofrece el método readObject() que nos da la opción de guardar la lectura directamente en un ArrayList. **Hemos de tener en cuenta realizar el cast.** Al finalizar la lectura cerramos el output.

```
ObjectInputStream ois;
ois = new ObjectInputStream(new FileInputStream(archivo));
lista = (ArrayList<NombreClase>) ois.writeObject();
ois.close();
```

Propiedad transient

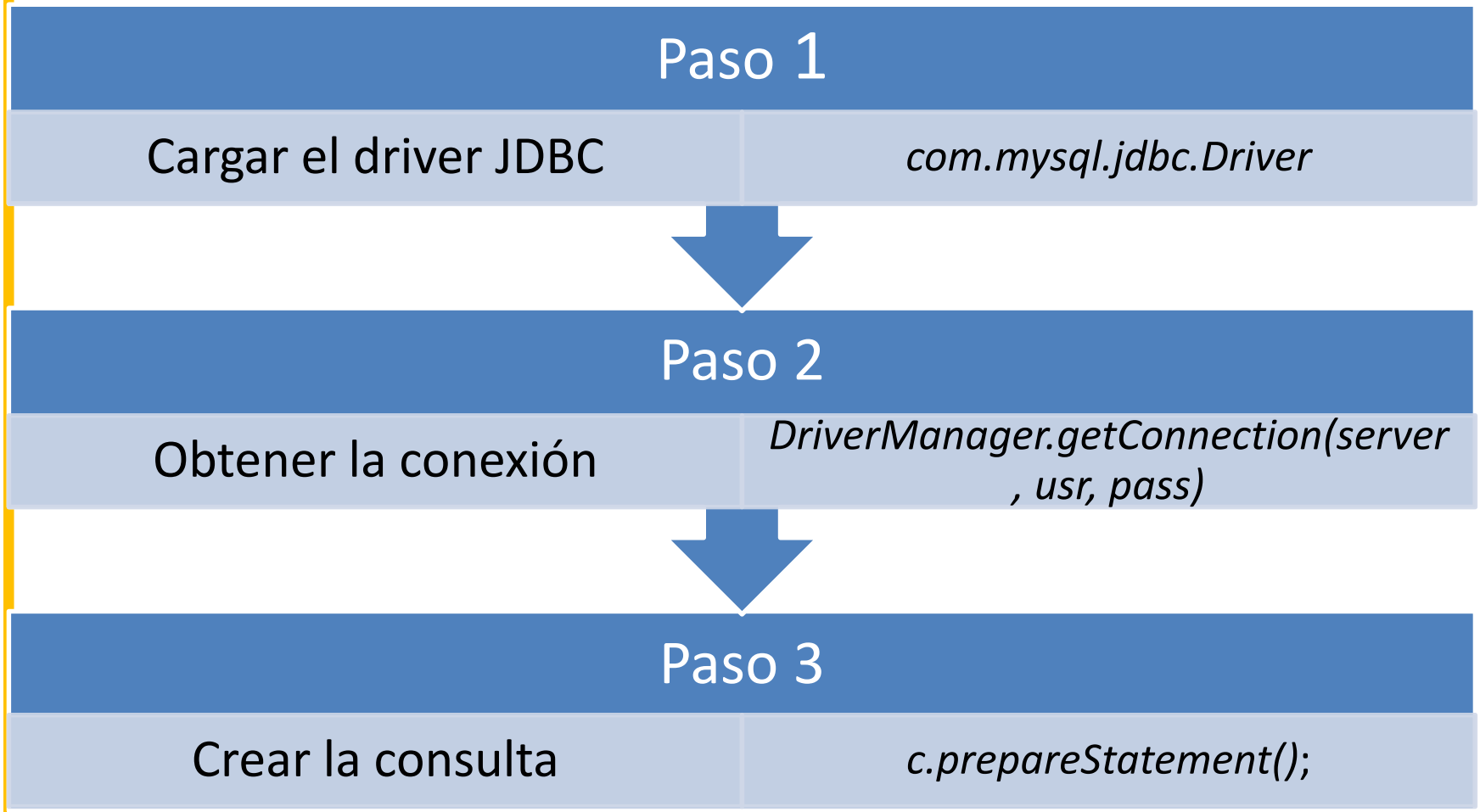
- Para evitar que una propiedad de un objeto sea serializada usaremos la propiedad **transient.**

```
public class NombreClasse implements Serializable{
        private String nombre;
        private transient int edad;
}
```

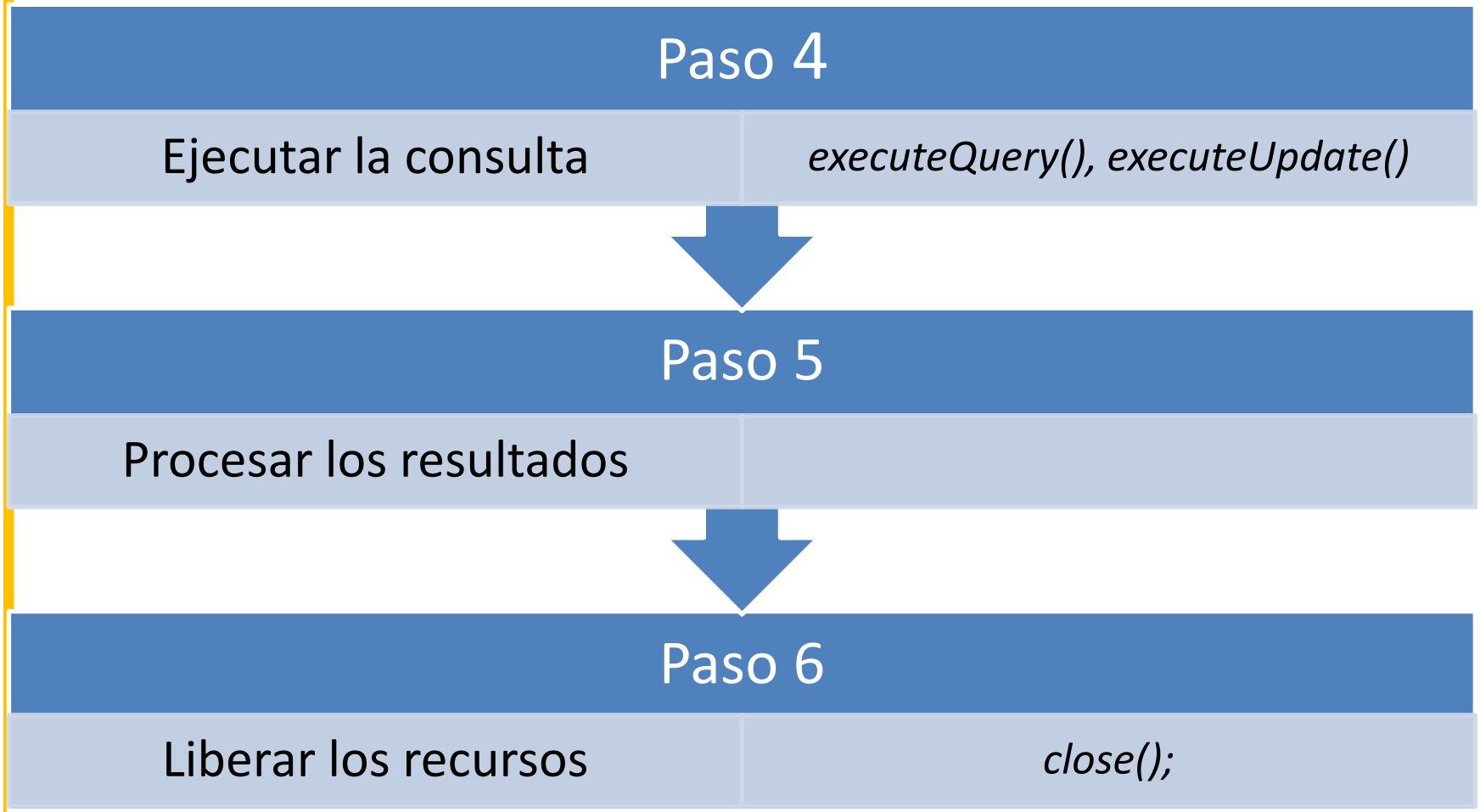
UF6 JDBC

- Al conjunto de clases encargadas de implementar la interfaz de programación de aplicaciones (API) y facilitar, con ello, el acceso a una base de datos se le denomina conector o driver.
- Cuando se construye una aplicación de base de datos, el conector oculta los detalles específicos de cada base de datos.
- Un ejemplo de conector muy extendido es el conector JDBC.

UF6 JDBC


Paso 1
Cargar el driver JDBC
com.mysql.jdbc.Driver
Paso 2
Obtener la conexión
DriverManager.getConnection(server , usr, pass)
Paso 3
Crear la consulta
c.prepareStatement();

UF6 JDBC


Paso 4
Ejecutar la consulta
executeQuery(), executeUpdate()
Paso 5
Procesar los resultados
Paso 6
Liberar los recursos
close();

UF6 JDBC

- *Connection* ofrecen un enlace activo a una base de datos a través del cual un programa en Java puede leer y escribir datos, así como explorar la estructura de la base de datos y sus capacidades.
- Interfaz *DriverManager*, complementaria de la clase *Connection*, con ella se registran los controladores JDBC y se proporcionan las conexiones que permiten manejar las URL específicas de JDBC.
- La clase *Statement* proporciona los métodos para que las sentencias, utilizando el lenguaje de consulta (SQL), sean realizadas sobre la base de datos.

Clase Statement

- Las sentencias *Statement* son las encargadas de ejecutar las sentencias SQL estáticas con *Connection.createStatement()*.
- El método *executeQuery()* de *Statement* está diseñado para sentencias que producen como resultado un único resultado (*ResultSet*), como es el caso de las sentencias SELECT.

UF6 JDBC

```
try {
   Statement stmt = con.createStatement();
   String query = "SELECT * FROM album WHERE titulo like 'B%';";
   ResultSet rs = stmt.executeQuery(query);
   while (rs.next()) {
      System.out.println("ID - " + rs.getInt("id") + ", Titulo " +
      rs.getString("titulo") + ", Autor " +
        rs.getString("autor"));
   }
   rs.close();
   stmt.close();
} catch (SQLException ex) {
   // tratar el error
}
```

Clase PreparedStatement

- Una variante a la sentencia *Statement* es la sentencia *PreparedStatement*.
- Se utiliza para ejecutar las sentencias SQL precompiladas.
- Permite que los parámetros de entrada sean establecidos de forma dinámica, ganando eficiencia.

Clase PreparedStatement

```
try {
   String query = "SELECT * FROM album WHERE titulo like ?;";
   PreparedStatement pst = con.prepareStatement(query);
   pst.setString(1, "B%");
   ResultSet rs = pst.executeQuery(query);
   while (rs.next()) {
      System.out.println("ID - " + rs.getInt("id") + ", Titulo " +
      rs.getString("titulo") + ", Autor " +
        rs.getString("autor"));
   }
   rs.close();
   pst.close();
} catch (SQLException ex) {
   // tratar el error
}
```

UF6 db4o

- Db4o (Data Base For Objects) es una implementación OODB libre (bajo GPL – General Public Licence, por lo tanto de código abierto) que permite de manera sencilla hacer persistentes objetos creado con Java y recuperarlos posteriormente en otra ejecución.

db4o

UF6 db4o

```
private ObjectContainer db;
db = Db4oEmbedded.openFile("datos.dat");

// ObjectContainer - Clase principal de
persistencia con db4o

private void grabarAlumno(Alumno a) {
 db.store(a);
}
```

UF6 db4o

```
public List<Alumno> selectAllAlumnos() {
    ArrayList<Alumno> alumnos = new ArrayList<>();
    Query q = db.query();
    q.constrain(Alumno.class);
    ObjectSet resultado = q.execute();
    while (resultado.hasNext()) {
        Alumno a = (Alumno) resultado.next();
        alumnos.add(a);
    }
    return alumnos;
}
```

UF6 db4o

```
// alumnos con edad superior a 20
Query q = db.query();
q.constrain(Alumno.class);
q.descend("nota").constraint(new Integer(20)).greater();

// ordenar por edad
q.constrain(Alumno.class);
q.descend("nota").orderAscending();
```

Que la fuerza te acompañe!

Linkia FP

Formación Profesional Oficial a Distancia