Edwin Maraví
emaravi@cjavaperu.com

CJAVA


# Misión

Nuestro equipo trabaja para integrar la tecnología Java en la sociedad como solución a todas sus necesidades.

CJAVA

siempre para apoyarte


# Visión

Poder aportar al desarrollo del País  usando tecnología Java.

CJAVA
siempre para apoyarte

# Servicios Académicos

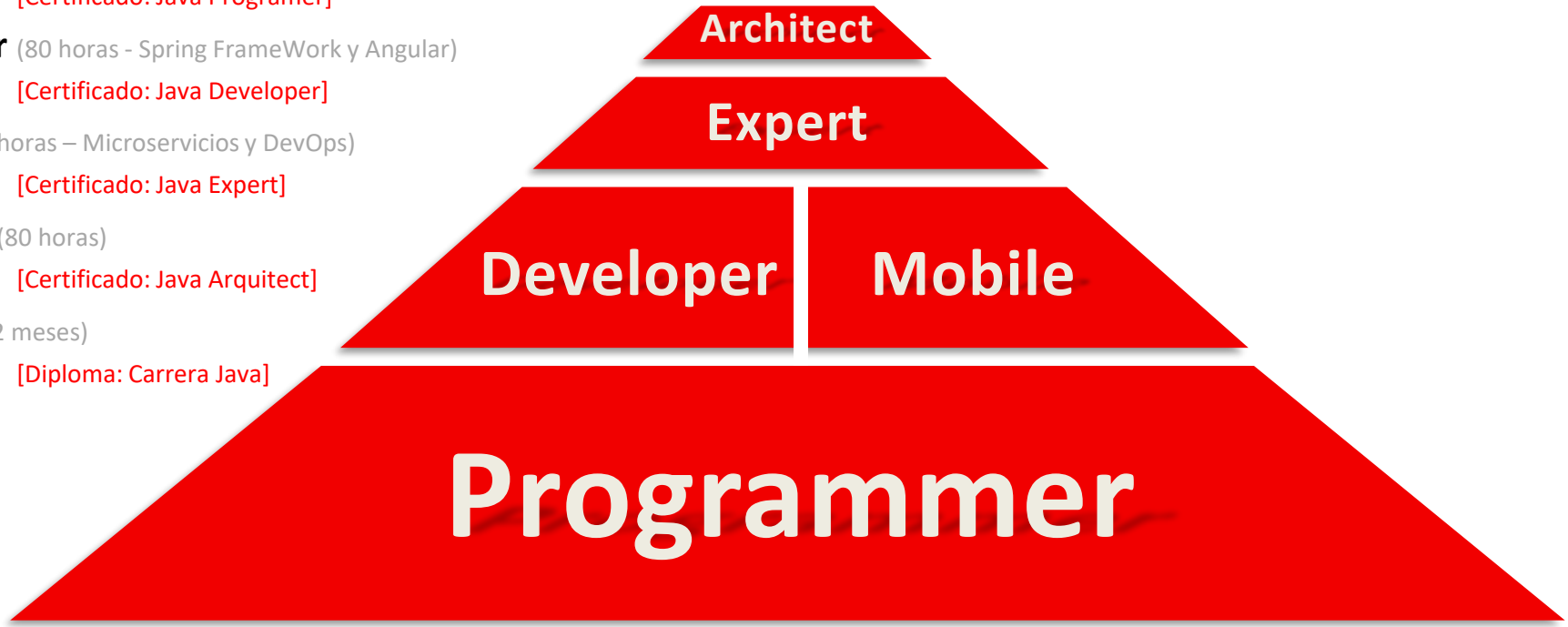
**Programer** (80 horas - Certificación Java 11)
[Certificado: Java Programer]

**Developer** (80 horas - Spring FrameWork y Angular)
[Certificado: Java Developer]

**Expert** (80 horas – Microservicios y DevOps)
[Certificado: Java Expert]

**Architect** (80 horas)
[Certificado: Java Arquitect]

**Carrera** (12 meses)
[Diploma: Carrera Java]



CJAVA
siempre para apoyarte

# Quienes Somos

Somos una organización orientada a desarrollar, capacitar e investigar tecnología JAVA a través de un prestigioso staff de profesionales a nivel nacional.


# Contáctenos

Av. Arenales 395 oficina 405
Santa Beatriz - Lima 01 - Perú

Teléfono: 433-6948

RPC / WhatsApp: 932 656 459

Email: info@cjavaperu.com

# Síguenos

/cjava.peru.1

/cjava_peru

/cjavaperu


# Introducción a JDBC

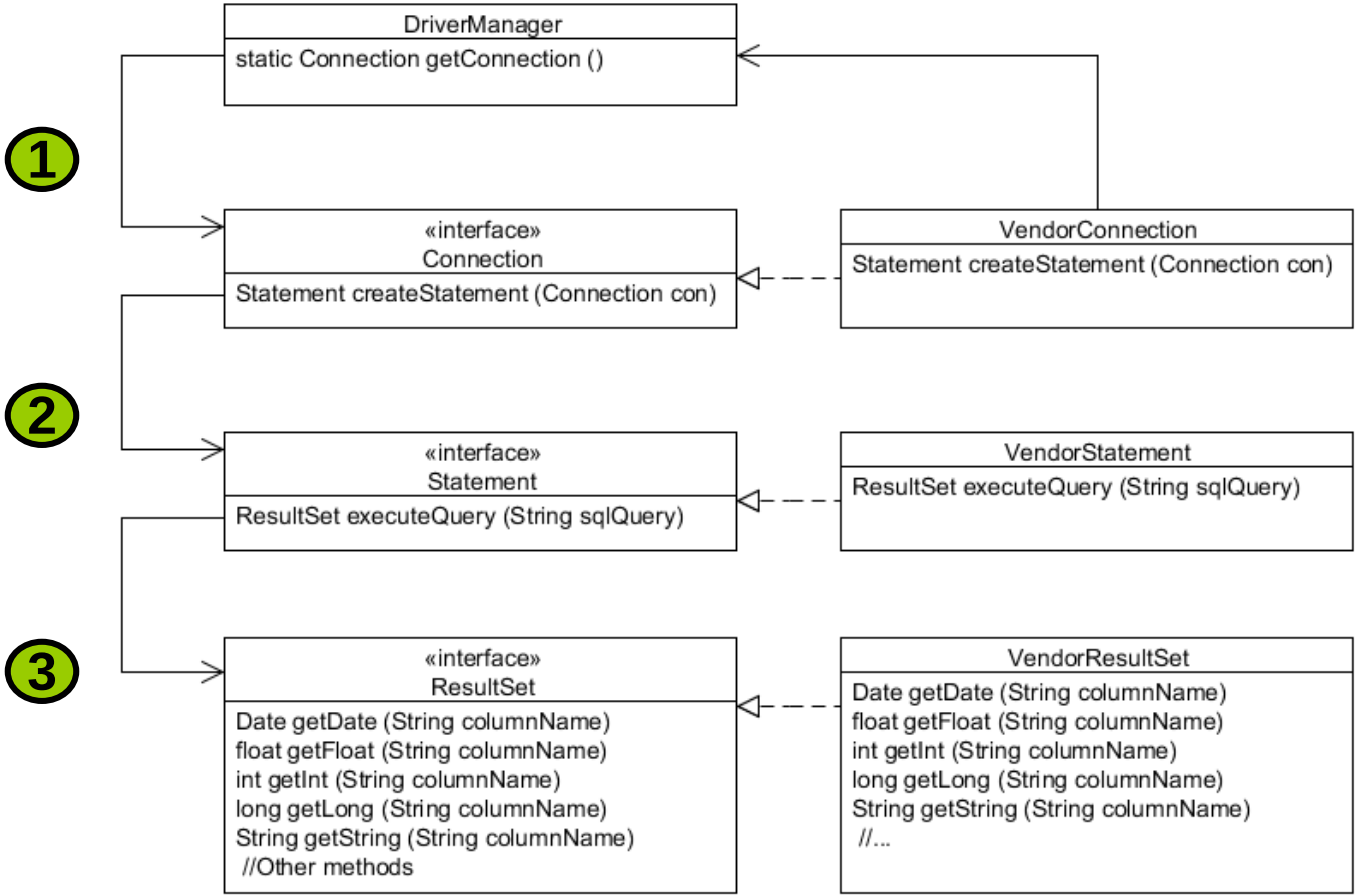
CJAVA
siempre para apoyarte

# Objetivos

Al finalizar esta lección, debería estar capacitado para:

- Definir el diseño de la API de JDBC
- Conectarse a una base de datos mediante un controlador JDBC
- Enviar consultas y obtener resultados de la base de datos
- Especificar información sobre el controlador JDBC de forma externa
- Usar transacciones con JDBC
- Usar RowSetProvider y RowSetFactory de JDBC 4.1
- Usar un patrón de objeto de acceso a datos para separar datos y métodos de negocio

# Uso de la API de JDBC



# Uso de clases de controlador de proveedor

La clase DriverManager se utiliza para obtener una instancia de un objeto de conexión, mediante el controlador JDBC nombrado en la URL de JDBC:

```
String url = "jdbc:derby://localhost:1527/EmployeeDB";
Connection con = DriverManager.getConnection (url);
```

- La sintaxis de URL para un controlador JDBC es:

```
jdbc:<driver>:[subsubprotocol:][databaseName][;attribute=value]
```

- Cada proveedor puede implantar su propio subprotocolo.
- La sintaxis de URL para un controlador Thin de Oracle es:

```
jdbc:oracle:thin:@//[HOST][:PORT]/SERVICE
```

Ejemplo:

```
jdbc:oracle:thin:@//myhost:1521/orcl
```

# Componentes de la API de JDBC clave

Cada clase de controlador JDBC del proveedor también implanta las clases de API clave que usará para conectarse a la base de datos, ejecutar consultas y manipular datos:

- java.sql.Connection: conexión que representa la sesión entre la aplicación Java y la base de datos.

```
Connection con = DriverManager.getConnection(url,
    username, password);
```

- java.sql.Statement: objeto usado para ejecutar una sentencia SQL estática y devolver el resultado.

```
        Statement stmt = con.createStatement();
```

- java.sql.ResultSet: objeto que representa un juego de resultados de la base de datos.

```
        String query = "SELECT * FROM Employee";
        ResultSet rs = stmt.executeQuery(query);
```

# Uso de un objeto ResultSet

```
String query = "SELECT * FROM Employee";
ResultSet rs = stmt.executeQuery(query);
```

ResultSet cursor

rs.next()

rs.next()

rs.next()

rs.next()

rs.next()

La primera llamada de método next() devuelve true y rs apunta a la primera fila de datos.

| | | | | |
|---|---|---|---|---|
| 110 | Troy | Hammer | 1965-03-31 | 102109.15 |
| 123 | Michael | Walton | 1986-08-25 | 93400.20 |
| 201 | Thomas | Fitzpatrick | 1961-09-22 | 75123.45 |
| 101 | Abhijit | Gopali | 1956-06-01 | 70000.00 |

`null`

La última llamada de método next() devuelve false y la instancia rs ahora es nula.

# Unión de todo

```
1     package com.example.text;
2
3     import java.sql.DriverManager;
4     import java.sql.ResultSet;
5     import java.sql.SQLException;
6     import java.util.Date;
7
8     public class SimpleJDBCTest {
9
10       public static void main(String[] args) {
11           String url = "jdbc:derby://localhost:1527/EmployeeDB";
12           String username = "public";
13           String password = "tiger";
14           String query = "SELECT * FROM Employee";
15           try (Connection con =
16                DriverManager.getConnection (url, username, password);
17                Statement stmt = con.createStatement ();
18                ResultSet rs = stmt.executeQuery (query)) {
```

La contraseña, nombre de usuario y URL de JDBC codificada sirven solo de ejemplo.

CJAVA
siempre po

Pase por todas las filas de ResultSet..

# Unión de todo

```
19    while (rs.next()) {
20                  int empID = rs.getInt("ID");
21                  String first = rs.getString("FirstName");
22                  String last = rs.getString("LastName");
23                  Date birthDate = rs.getDate("BirthDate");
24                  float salary = rs.getFloat("Salary");
25                  System.out.println("Employee ID:   " + empID +
    "\n"
26                  + "Employee Name: " + first + " " + last + "\n"
27                  + "Birth Date:    " + birthDate + "\n"
28                  + "Salary:        " + salary);
29              } // end of while
30          } catch (SQLException e) {
31              System.out.println("SQL Exception: " + e);
32          } // end of try-with-resources
33      }
34    }
```

# Escritura de código JDBC portátil

El controlador JDBC proporciona una capa "aislante" mediante programación entre la aplicación Java y la base de datos. Sin embargo, también debe considerar la semántica y la sintaxis SQL al escribir aplicaciones de base de datos.

- La mayoría de las bases de datos soportan un juego estándar de semántica y sintaxis SQL descrita por la especificación de nivel de entrada SQL-92 de ANSI (American National Standards Institute).
- Puede comprobar mediante programación el soporte para esta especificación desde su controlador:

```
Connection con = DriverManager.getConnection(url, username,
   password);
DatabaseMetaData dbm = con.getMetaData();
if (dbm.supportsANSI92EntrySQL()) {
   // Support for Entry-level SQL-92 standard
}
```

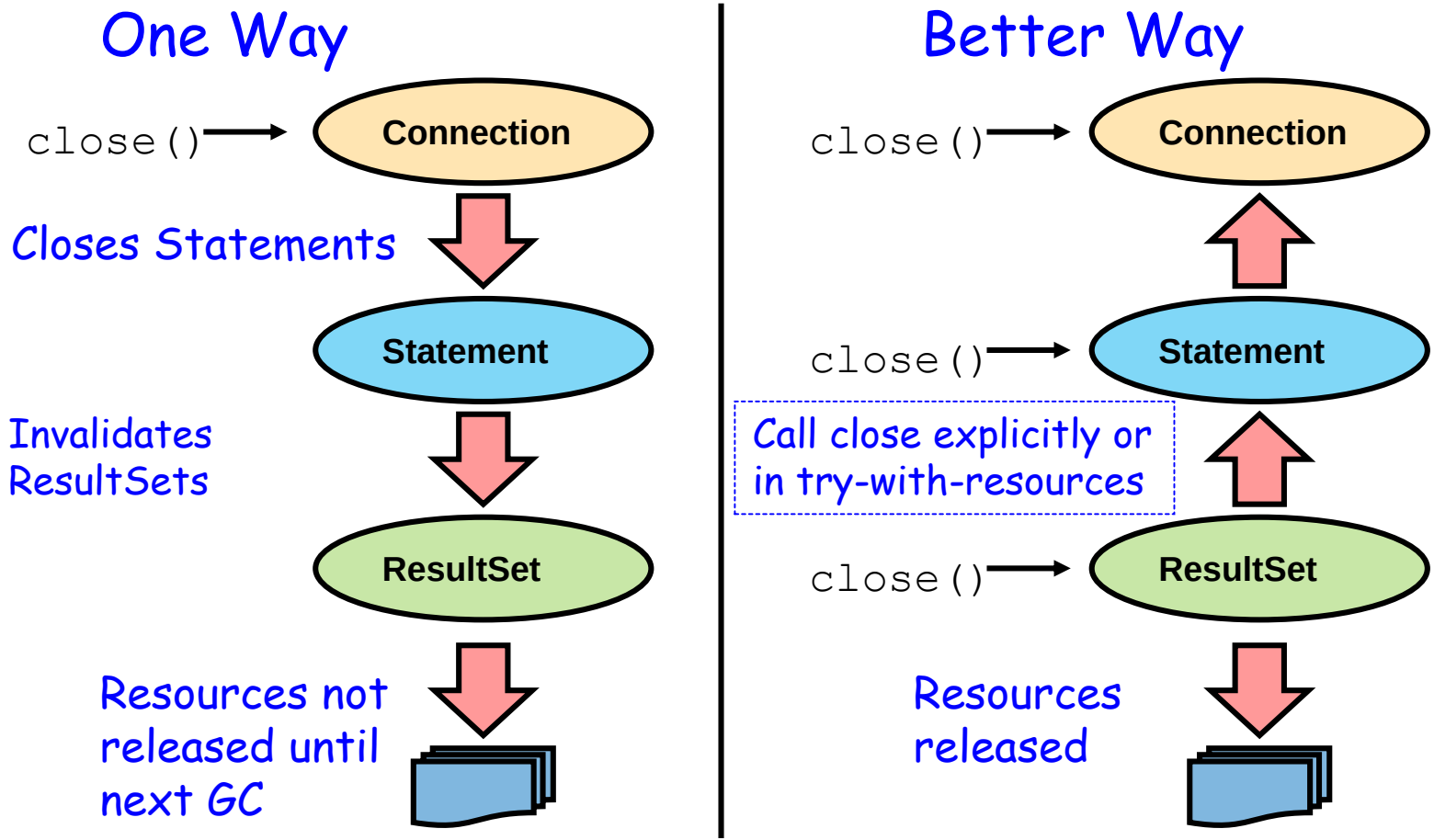
# Clase SQLException

SQLException se puede usar para notificar detalles sobre los errores de la base de datos resultantes. Para informar de todas las excepciones devueltas, puede iterar con la SQLException devuelta:

```
1     catch(SQLException ex) {
2         while(ex != null) {
3             System.out.println("SQLState:  " + ex.getSQLState());
4             System.out.println("Error Code:" + ex.getErrorCode());
5             System.out.println("Message:   " + ex.getMessage());
6             Throwable t = ex.getCause();
7             while(t != null) {
8                 System.out.println("Cause:" + t);
9                 t = t.getCause();
10            }
11            ex = ex.getNextException();
12        }
13    }
```

Códigos de estado, códigos de error y mensajes dependientes del proveedor

# Cierre de objetos de JDBC



# Construcción try-with-resources

Con la siguiente sentencia try-with-resources:

```
try (Connection con =
     DriverManager.getConnection(url, username, password);
     Statement stmt = con.createStatement();
     ResultSet rs = stmt.executeQuery (query)){
```

Con la siguiente sentencia try-with-resources:

- El compilador comprueba para ver que el objeto entre paréntesis implanta java.lang.AutoCloseable.
  - Esta interfaz incluye un método: void close().
- El método de cierre se llama automáticamente al final de bloque try en el orden adecuado (de la última declaración a la primera).
- Es posible incluir varios recursos que se puedan cerrar en el bloque try, separados por punto y coma.

# Try-with-resources: práctica incorrecta

Puede ser tentador escribir try-with-resources de forma más compacta:

```
try (ResultSet rs = DriverManager.getConnection(url,
username, password).createStatement().executeQuery(query)) {
```

- Sin embargo, solo se llama el método de cierre de ResultSet, lo cual no es una buena práctica.
- Recuerde siempre qué recursos debe cerrar al usar trywith-resources.

# Escritura de consultas y obtención de resultados

Para ejecutar consultas SQL con JDBC, debe crear un objeto de envoltorio de consulta SQL, una instancia del objeto Statement.

```
Statement stmt = con.createStatement();
```

- Utilice la instancia Statement para ejecutar una consulta SQL:

```
ResultSet rs = stmt.executeQuery (query);
```

- Tenga en cuenta que hay tres métodos de ejecución de Statement:

| Method | Returns | Used for |
|---|---|---|
| `executeQuery(sqlString)` | `ResultSet` | `SELECT` statement |
| `executeUpdate(sqlString)` | `int` (rows affected) | `INSERT`, `UPDATE`, `DELETE`, or a DDL |
| `execute(sqlString)` | `boolean` (true if there was a `ResultSet`) | Any SQL command or commands |

# ResultSetMetaData

En un momento dado puede necesitar detectar dinámicamente el número de columnas y su tipo.

```
1    int numCols = rs.getMetaData().getColumnCount();
2    String [] colNames = new String[numCols];
3    String [] colTypes = new String[numCols];
4    for (int i= 0; i < numCols; i++) {
5        colNames[i] = rs.getMetaData().getColumnName(i+1);
6        colTypes[i] = rs.getMetaData().getColumnTypeName(i+1);
7    }
8    System.out.println ("Number of columns returned: " + numCols);
9    System.out.println ("Column names/types returned: ");
10   for (int i = 0; i < numCols; i++) {
11       System.out.println (colNames[i] + " : " + colTypes[i]);
12   }
```

Observe que estos métodos se indexan desde 1, no 0.

# Obtención de recuento de filas

Una pregunta común al ejecutar una consulta es cuántas filas se han devuelto.

```
1    public int rowCount(ResultSet rs) throws SQLException{
2        int rowCount = 0;
3        int currRow = rs.getRow();
4        // Valid ResultSet?
5        if (!rs.last()) return -1;
6        rowCount = rs.getRow();
7        // Return the cursor to the current position
8        if (currRow == 0) rs.beforeFirst();
9        else rs.absolute(currRow);
10      return rowCount;
11   }
```

Move the cursor to the last row, this method returns `false` if the `ResultSet` is empty.

Returning the row cursor to its original position before the call is a good practice.

- Para usar esta técnica, ResultSet debe ser desplazable.

# Control del tamaño de recuperación de ResultSet

Por defecto, el número de filas recuperadas a la vez por una consulta se determina mediante el controlador JDBC. Es posible que desee controlar este comportamiento en el caso de juegos de datos grandes.

- Por ejemplo, si desea limitar el número de filas recuperadas en la caché a 25, puede definir el tamaño de recuperación:

```
rs.setFetchSize(25);
```

- Las llamadas a rs.next() devuelven los datos en la caché hasta la fila 26a, momento en el que el controlador recuperará otras 25 filas.

# Uso de PreparedStatement

PreparedStatement es una subclase de Statement que permite transferir argumentos a una sentencia SQL precompilada.

```
double value = 100_000.00;
String query = "SELECT * FROM Employee WHERE Salary > ?";
PreparedStatement pStmt = con.prepareStatement(query);
pStmt.setDouble(1, value);
ResultSet rs = pStmt.executeQuery();
```

Parámetro para sustitución.

Sustituye value para el primer parámetro en la sentencia preparada.

- En este fragmento de código, una sentencia preparada devuelve todas las columnas de todas las filas cuyo salario es mayor de 100 000 dólares.
- PreparedStatement es útil cuando tiene sentencias SQL que va a ejecutar varias veces.

CJAVA
siempre para apoyarte

# Uso de CallableStatement

CallableStatement permite que sentencias no SQL (como procedimientos almacenados) se ejecuten en la base de datos.

```
CallableStatement cStmt
       = con.prepareCall("{CALL EmplAgeCount (?, ?)}");
int age = 50;
cStmt.setInt (1, age);
ResultSet rs = cStmt.executeQuery();
cStmt.registerOutParameter(2, Types.INTEGER);
boolean result = cStmt.execute();
int count = cStmt.getInt(2);
System.out.println("There are " + count +
          " Employees over the age of " + age);
```

- Los procedimientos almacenados se ejecutan en la base de datos.

CJAVA
siempre para apoyarte

# ¿Qué es una transacción?

- Una transacción es un mecanismo para manejar grupos de operaciones como si fueran solo una.
- Puede darse el caso de que ocurran todas las operaciones de una transacción o ninguna en absoluto.
- Las operaciones implicadas en una transacción pueden depender de una o más bases de datos.

CJAVA
siempre para apoyarte

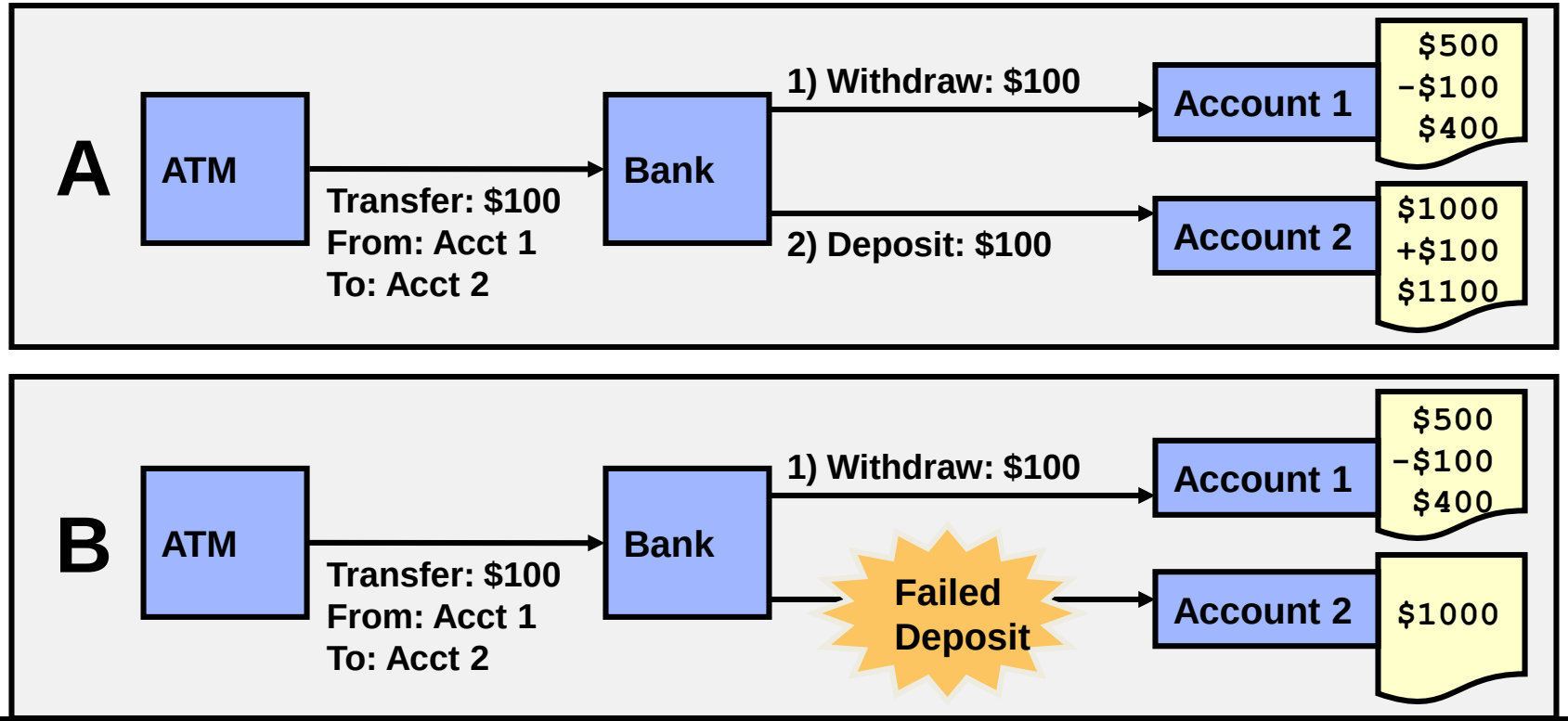
# Propiedades ACID de una transacción

Una transacción formalmente la define el juego de propiedades que se conoce por el acrónimo ACID.

- **Atomicidad:** una transacción se hace o se deshace completamente. En caso de fallo, todas las operaciones y procedimientos se deshacen y se realiza un rollback de todos los datos a su estado anterior.
- **Consistencia:** una transacción transforma un sistema desde un estado consistente a otro estado consistente.
- **Aislamiento:** cada transacción ocurre de forma independiente de otras transacciones que ocurren al mismo tiempo.
- **Durabilidad:** las transacciones completadas permanecen como permanentes, incluso durante un fallo del sistema.

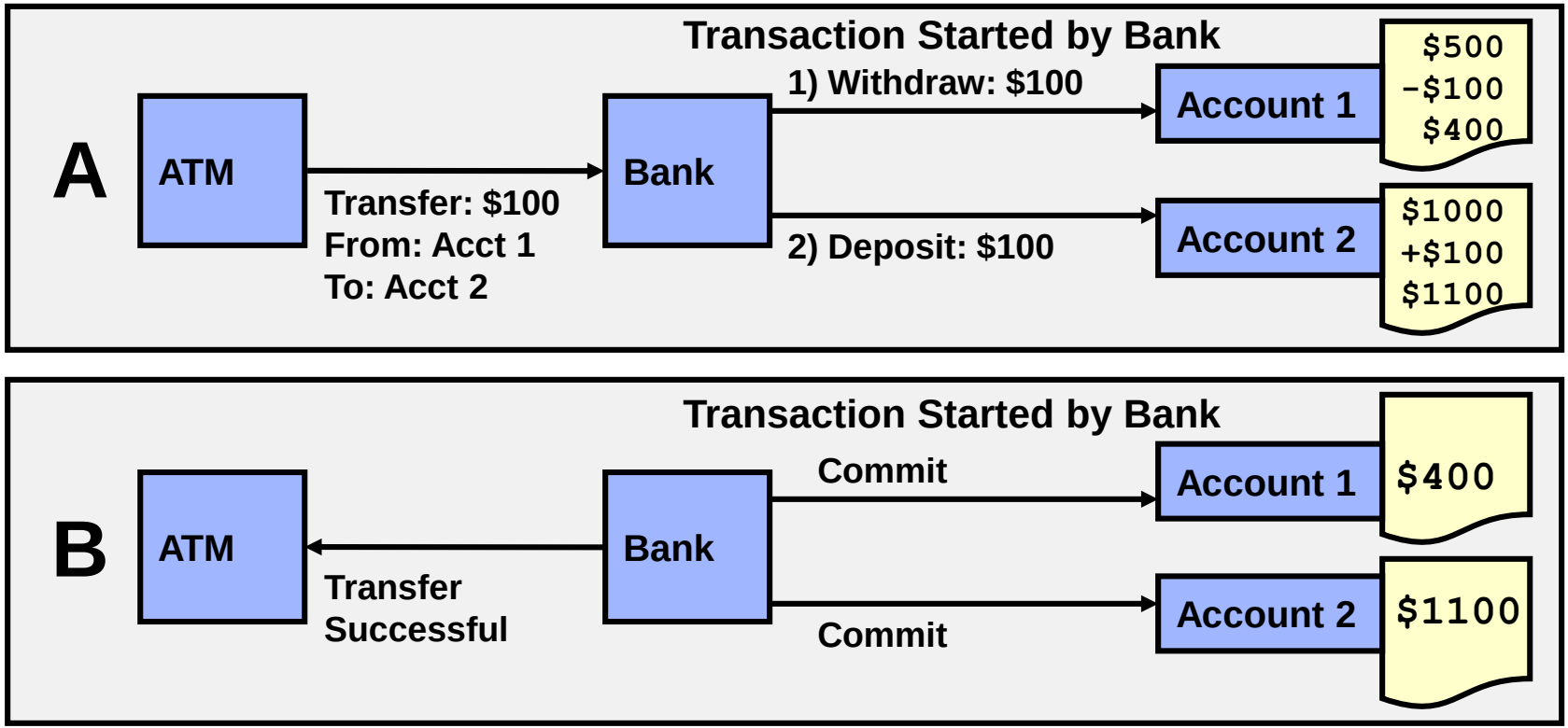
# Transferencia sin transacciones

- Transferencia correcta (A)
- Transferencia incorrecta (las cuentas se dejan en un estado inconsistente) (B)



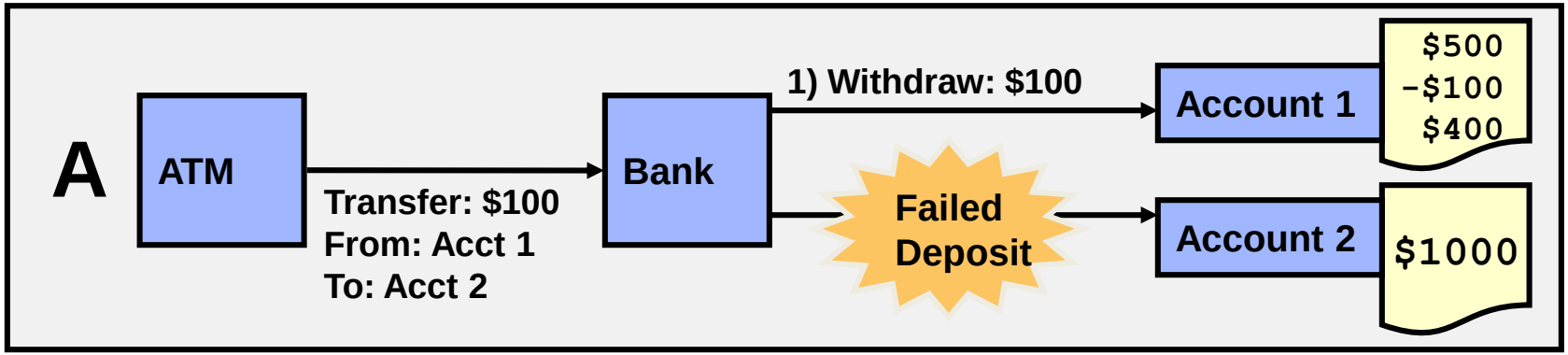
# Transferencia correcta con transacciones

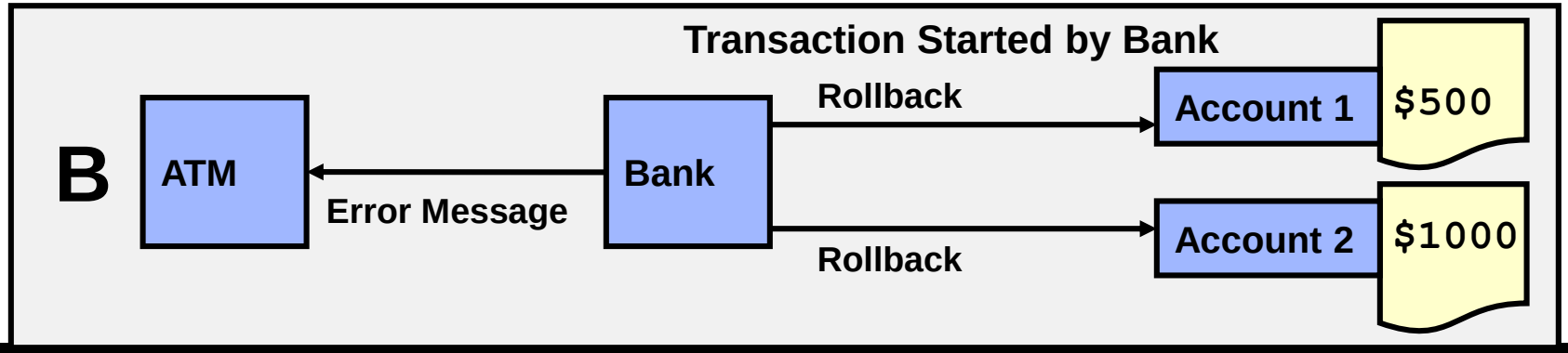
- Los cambios en una transacción se almacenan en buffer. (A)
- Si una transferencia es correcta, los cambios se confirman (se hacen permanentes). (B)



# Transferencia incorrecta con transacciones

- Los cambios en una transacción se almacenan en buffer. (A)
- Si se produce un problema, la transacción realiza un rollback al último estado consistente. (B)



# Transacciones JDBC

Por defecto, cuando se crea un objeto Connection, se hace en modo de confirmación automática.

- Cada sentencia SQL individual se trata como una transacción y se confirma automáticamente después de la ejecución.
- Para agrupar dos o más sentencias, debe desactivar el modo de confirmación automática.

```
con.setAutoCommit (false);
```

- Debe llamar de forma explícita el método de confirmación para completar la transacción con la base de datos.

```
con.commit();
```

- También puede realizar un rollback mediante programación de las transacciones en caso de fallo.

```
con.rollback();
```

# RowSet 1.1: RowSetProvider y RowSetFactory

La especificación de la API de JDK 7 presenta la nueva API de RowSet 1.1. Una de las nuevas funciones de esta API es RowSetProvider.

- javax.sql.rowset.RowSetProvider se usa para crear un objeto RowSetFactory:

```
myRowSetFactory = RowSetProvider.newFactory();
```

- La implantación por defecto RowSetFactory es:

```
com.sun.rowset.RowSetFactoryImpl
```

- RowSetFactory se usa para crear uno de los tipos de objeto RowSet de RowSet 1.1.

# Uso de RowSetFactory de RowSet 1.1

RowSetFactory se usa para crear instancias de implantaciones de RowSet:

| RowSet type | Provides |
|---|---|
| `CachedRowSet` | A container for rows of data that caches its rows in memory |
| `FilteredRowSet` | A RowSet object that provides methods for filtering support |
| `JdbcRowSet` | A wrapper around ResultSet to treat a result set as a JavaBeans component |
| `JoinRowSet` | A RowSet object that provides mechanisms for combining related data from different RowSet objects |
| `WebRowSet` | A RowSet object that supports the standard XML document format required when describing a RowSet object in XML |

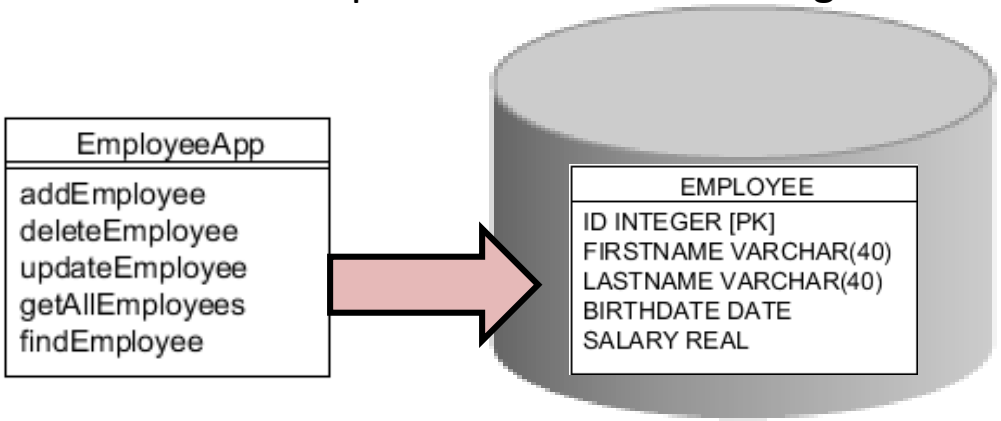
CJAVA
siempre para apoyarte

# Ejemplo: Uso de JdbcRowSet

```
10   try (JdbcRowSet jdbcRs =
11        RowSetProvider.newFactory().createJdbcRowSet()) {
12       jdbcRs.setUrl(url);
13       jdbcRs.setUsername(username);
14       jdbcRs.setPassword(password);
15       jdbcRs.setCommand("SELECT * FROM Employee");
16       jdbcRs.execute();
17       // Now just treat JDBC Row Set like a ResultSet object
18       while (jdbcRs.next()) {
19           int empID = jdbcRs.getInt("ID");
20           String first = jdbcRs.getString("FirstName");
21           String last = jdbcRs.getString("LastName");
22           Date birthDate = jdbcRs.getDate("BirthDate");
23            float salary = jdbcRs.getFloat("Salary");
24       }
25     //... other methods
26   }
```

# Objetos de acceso a datos

Piense en una tabla de empleado como la del código JDBC del ejemplo.

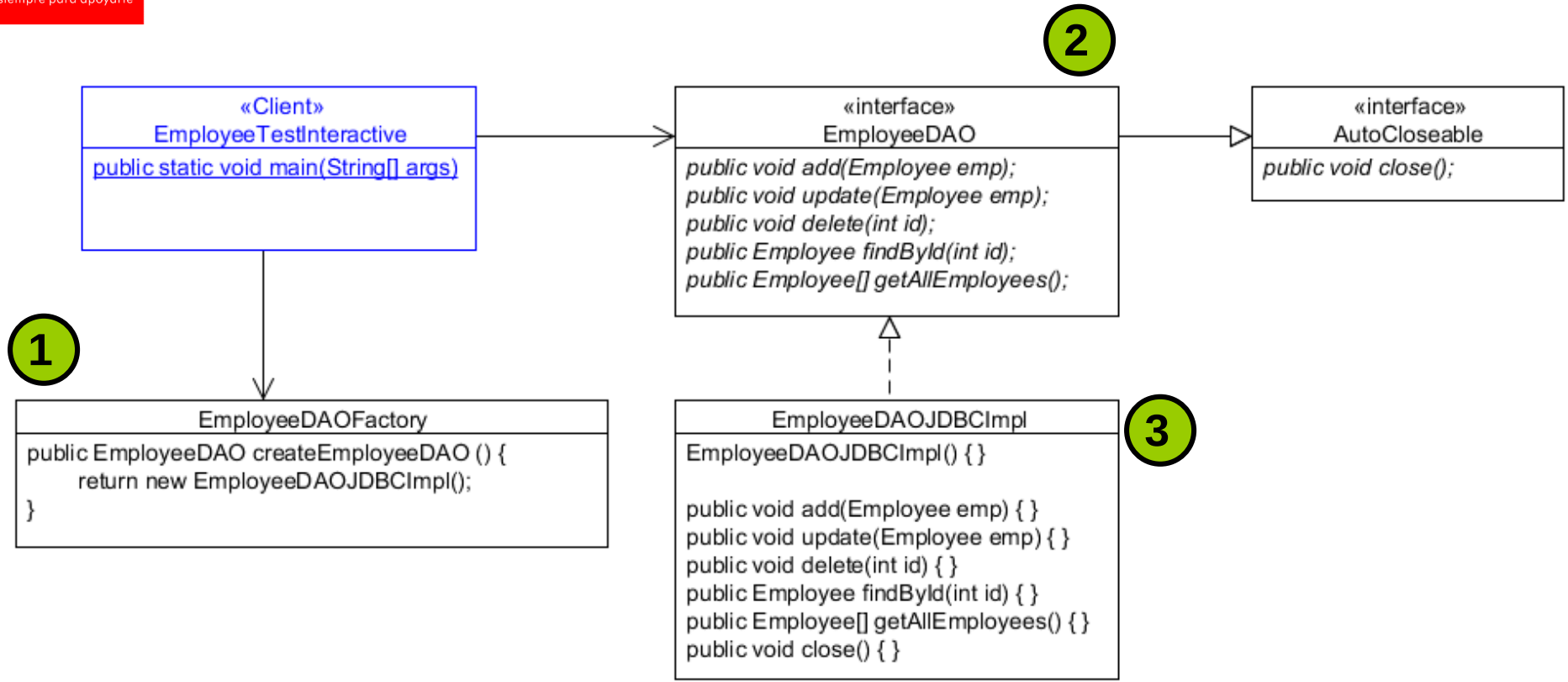



Piense en una tabla de empleado como la del código JDBC del ejemplo.

- Al combinar el código que accede a la base de datos con la lógica "negocio", los métodos de acceso a datos y la tabla Employee se acoplan.
- Los cambios en la tabla (como la adición de un campo) requerirán un cambio completo en la aplicación.
- Los datos del empleado no se encapsulan en la aplicación del ejemplo.

CJAVA
siempre para apoyarte

# Patrón de objeto de acceso a datos



CJAVA
siempre para apoyarte

# Resumen

En esta lección, debe haber aprendido a hacer lo siguiente:

- Definir el diseño de la API de JDBC
- Conectarse a una base de datos mediante un controlador JDBC
- Enviar consultas y obtener resultados de la base de datos
- Especificar información sobre el controlador JDBC de forma externa
- Usar transacciones con JDBC
- Usar RowSetProvider y RowSetFactory de JDBC 4.1
- Usar un patrón de objeto de acceso a datos para separar datos y métodos de negocio


# Localización

CJAVA
siempre para apoyarte

# Objetivos

Al finalizar esta lección, debería estar capacitado para:

- Describir las ventajas de localizar una aplicación
- Definir lo que representa una configuración regional
- Leer y definir la configuración regional mediante el objeto Locale
- Crear un grupo de recursos para cada configuración regional
- Llamar a un grupo de recursos desde una aplicación
- Cambiar la configuración regional para un grupo de recursos
- Aplicar formato a texto para la localización mediante NumberFormat y DateFormat

CJAVA
siempre para apoyarte

# ¿Por qué localizar?

La decisión de crear una versión de una aplicación para uso internacional se suele presentar al inicio de un proyecto de desarrollo.

- Software que tiene en cuenta el idioma y la región
- Fechas, números y monedas con formato para países específicos
- Capacidad para conectarse a datos específicos del país sin cambiar el código

# Aplicación de ejemplo

Localizar una aplicación de ejemplo:

- Interfaz de usuario basada en texto
- Localización de menús
- Muestra de localizaciones de moneda y fecha

```
=== Localization App ===
1. Set to English
2. Set to French
3. Set to Chinese
4. Set to Russian
5. Show me the date
6. Show me the money!
q. Enter q to quit
Enter a command:
```

CJAVA
siempre para apoyarte

# Locale

Locale especifica un idioma y país determinado:

- Idioma
  - Código ISO 639: Alfa-2 o Alfa-3
  - “en” para inglés, “es” para español
  - Siempre utiliza minúscula
- País
  - Utiliza el código de país ISO 3166: Alfa-2 o el código de área numérico UN M.49
  - "US" para Estados Unidos, "ES" para España
  - Siempre utiliza mayúscula
- Consulte los tutoriales de Java para obtener más información sobre todos los estándares utilizados

CJAVA
siempre para apoyarte

# Grupo de recursos

- La clase ResourceBundle aísla los datos específicos de la configuración regional:
  – Devuelve pares clave/valor almacenados de forma independiente.
  – Puede ser una clase o un archivo .properties.
- Pasos que usar:
  – Crear archivos de grupo para cada configuración regional.
  – Llamar a una configuración regional específica desde la aplicación.

# Archivo de grupo de recursos

- El archivo de propiedades contiene un juego de pares clave/valor.
  - Cada clave identifica un componente de aplicación específico.
  - Los nombres de archivo especiales utilizan códigos de idioma y país.
- Valor por defecto para la aplicación de ejemplo:
  - Menú convertido en grupo de recursos

**MessageBundle.properties**

```
menu1 = Set to English
menu2 = Set to French
menu3 = Set to Chinese
menu4 = Set to Russian
menu5 = Show the Date
menu6 = Show me the money!
menuq = Enter q to quit
```

# Archivos del grupo de recursos de ejemplo

Ejemplos para francés y chino

**MessagesBundle_fr_FR.properties**

```
menu1 = Régler à l'anglais
menu2 = Régler au français
menu3 = Réglez chinoise
menu4 = Définir pour la Russie
menu5 = Afficher la date
menu6 = Montrez-moi l'argent!
menuq = Saisissez q pour quitter
```

**MessagesBundle_zh_CN.properties**

```
menu1 = 设置为英语
menu2 = 设置为法语
menu3 = 设置为中文
menu4 = 设置到俄罗斯
menu5 = 显示日期
menu6 = 显示我的钱!
menuq = 输入q退出
```

# Inicialización de la aplicación de ejemplo

```
PrintWriter pw = new PrintWriter(System.out, true);
    // More init code here

    Locale usLocale = Locale.US;
    Locale frLocale = Locale.FRANCE;
    Locale zhLocale = new Locale("zh", "CN");
    Locale ruLocale = new Locale("ru", "RU");
    Locale currentLocale = Locale.getDefault();

    ResourceBundle messages =
  ResourceBundle.getBundle("MessagesBundle", currentLocale);

   // more init code here

    public static void main(String[] args){
        SampleApp ui = new SampleApp();
        ui.run();
    }
```

# Aplicación de ejemplo: bucle principal

```
public void run(){
        String line = "";
        while (!(line.equals("q"))){
            this.printMenu();
            try { line = this.br.readLine(); }
            catch (Exception e){ e.printStackTrace(); }

            switch (line){
                case "1": setEnglish(); break;
                case "2": setFrench(); break;
                case "3": setChinese(); break;
                case "4": setRussian(); break;
                case "5": showDate(); break;
                case "6": showMoney(); break;
            }
        }
    }
```

# Método printMenu

En lugar de texto, se utiliza el grupo de recursos.

- messages es un grupo de recursos.
- Se utiliza una clave para recuperar cada opción de menú.
- El idioma se selecciona según la definición de Locale.

```
public void printMenu(){
   pw.println("=== Localization App ===");
   pw.println("1. " + messages.getString("menu1"));
   pw.println("2. " + messages.getString("menu2"));
   pw.println("3. " + messages.getString("menu3"));
   pw.println("4. " + messages.getString("menu4"));
   pw.println("5. " + messages.getString("menu5"));
   pw.println("6. " + messages.getString("menu6"));
   pw.println("q. " + messages.getString("menuq"));
   System.out.print(messages.getString("menucommand")+" ");
}
```

# Cambio de Locale

Para cambiar Locale:

- Defina currentLocale en el idioma deseado.
- Vuelva a cargar el grupo mediante la configuración regional actual.

```
public void setFrench(){
    currentLocale = frLocale;
    messages = ResourceBundle.getBundle("MessagesBundle",
  currentLocale);
}
```

# Interfaz de ejemplo con francés

Después de seleccionar la opción de francés, la interfaz de usuario actualizada es parecida a la siguiente:

```
=== Localization App ===
1. Régler à l'anglais
2. Régler au français
3. Réglez chinoise
4. Définir pour la Russie
5. Afficher la date
6. Montrez-moi l'argent!
q. Saisissez q pour quitter
Entrez une commande:
```

CJAVA
siempre para apoyarte

# Formato de fecha y moneda

- Los números se pueden localizar y mostrar en su formato local.
- Las clases de formato especial incluyen:
  - DateFormat
  - NumberFormat
- Cree objetos mediante Locale.

# Inicialización de fecha y moneda

La aplicación puede mostrar la moneda y la fecha con formato local. Las variables se inicializan de la siguiente forma:

```
// More init code precedes
    NumberFormat currency;
    Double money = new Double(1000000.00);

    Date today = new Date();
    DateFormat df;
```

# Visualización de fecha

- Aplicación de formato a una fecha:
  - Obtenga un objeto DateFormat basado en el objeto Locale.
  - Llame al método format transfiriendo la fecha al formato.

```
public void showDate(){

    df = DateFormat.getDateInstance(DateFormat.DEFAULT, currentLocale);
    pw.println(df.format(today) + " " + currentLocale.toString());
}
```

- Fechas de ejemplo:

20 juil. 2011 fr_FR

20.07.2011 ru_RU

# Personalización de fechas

- Las constantes DateFormat incluyen:
  - SHORT: es completamente numérica, como 12.13.52 o 3:30 p.m.
  - MEDIUM: es más larga, como 12 ene, 1952
  - LONG: es más larga, como 12 de enero,1952 o 3:30:32 p.m.
  - FULL: se especifica completamente, como martes, 12 de abril, 1952 DC o 3:30:42 p.m. PST
- SimpleDateFormat:
  - Una subclase de una clase DateFormat

| Letter | Date or Time | Presentation | Examples |
|---|---|---|---|
| G | Era | Text | AD |
| y | Year | Year | 1996; 96 |
| M | Month in Year | Month | July; Jul; 07 |

# Visualización de moneda

- Aplicación de formato a la moneda:
  - Obtenga una instancia de moneda de NumberFormat.
  - Transfiera Double al método format.

```
public void showMoney(){
   currency = NumberFormat.getCurrencyInstance(currentLocale);
   pw.println(currency.format(money) + " " +
  currentLocale.toString());
}
```

- Salida de moneda de ejemplo:

1 000 000 руб. ru_RU

1 000 000,00 fr_FR

￥1,000,000.00 zh_CN

CJAVA
siempre para apoyarte

# Resumen

En esta lección, debe haber aprendido a hacer lo siguiente:

- Describir las ventajas de localizar una aplicación
- Definir lo que representa una configuración regional
- Leer y definir la configuración regional mediante el objeto Locale
- Crear un grupo de recursos para cada configuración regional
- Llamar a un grupo de recursos desde una aplicación
- Cambiar la configuración regional para un grupo de recursos
- Aplicar formato a texto para la localización mediante NumberFormat y DateFormat


# Contáctenos

Av. Arenales 395 oficina 405
Santa Beatriz - Lima 01 - Perú

Teléfono: 433-6948

RPC / WhatsApp: 932 656 459

Email: info@cjavaperu.com

# Síguenos

/cjava.peru.1

/cjava_peru

/cjavaperu

CJAVA

siempre para apoyarte

#CJavaNoPara

Gracias