




ORM – con java y hibernate - jpa


MCC Ramón Mora Márquez

IEU





Fundamentos de Java para desarrollo de bases de datos.



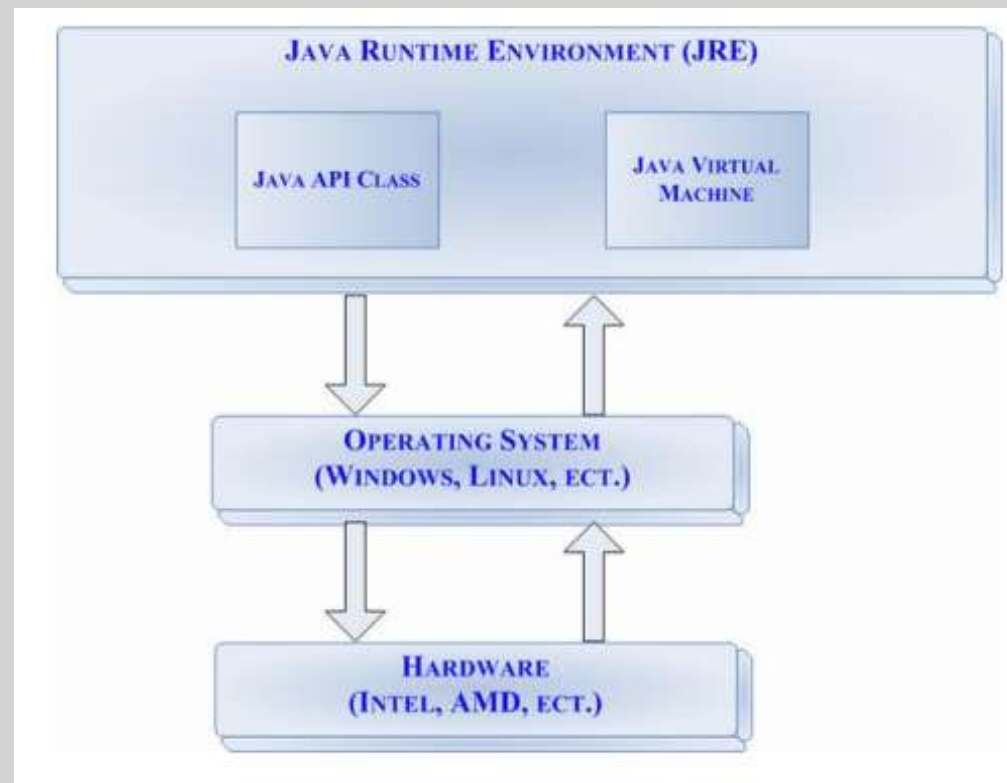
Agenda:

- Java virtual machine.
- JDK y JRE.
- Ambientes de desarrollo integrado.
- Tipos primitivos.
- Estructuras de control.
- Orientación a objetos: Clases, herencia, polimorfismo, interfaces
- Excepciones.
- Colecciones: Listas, y Mapas.
- Java y Bases de datos: JDBC

Java virtual machine

- El lenguaje de programación Java creado por Sun Microsystems liberado en 1995 y comprado por Oracle en 2009.
- Sus aplicaciones son compiladas a código binario (byte code) las cuales corren sobre la **maquina virtual java (JVM)**, las JVM corre sobre cualquier computadora, con la filosofía “write once, run anywhere (WORA)”.
- La JVM ejecuta el código java como un **interprete** de java byte code(archivos class).

Java virtual machine



JDK Y JRE

- Para poder trabajar con Java se requieren sus herramientas principales **Java Development Kit(JDK)** y el **Java Runtime Environment(JRE)**
- JDK provee herramientas para compilación y desarrollo convirtiendo los códigos fuentes (archivo.java) a código binario (.class).
- El compilador es el programa javac
- JRE ejecuta los programas binarios, el interprete de jvm es java

JDK Y JRE

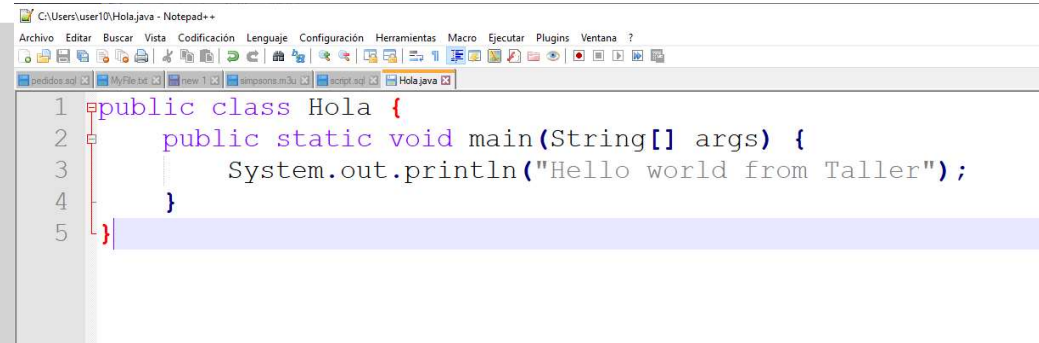


```
C:\Users\user10>java -version
java version "1.8.0_201"
Java(TM) SE Runtime Environment (build 1.8.0_201-b09)
Java HotSpot(TM) 64-Bit Server VM (build 25.201-b09, mixed mode)

C:\Users\user10>
```

JDK Y JRE

- Hola mundo en Java
- Abrir editor de texto, poner el nombre archivo
- C:\Program Files\Java\jdk1.8.0_ver\bin\javac
Hola.java opcional -verbose
- Java Hola



A screenshot of a Notepad++ window titled 'C:\Users\user10\Hola.java - Notepad++'. The window displays a Java program with the following code:

```
1 public class Hola {  
2     public static void main(String[] args) {  
3         System.out.println("Hello world from Taller");  
4     }  
5 }
```



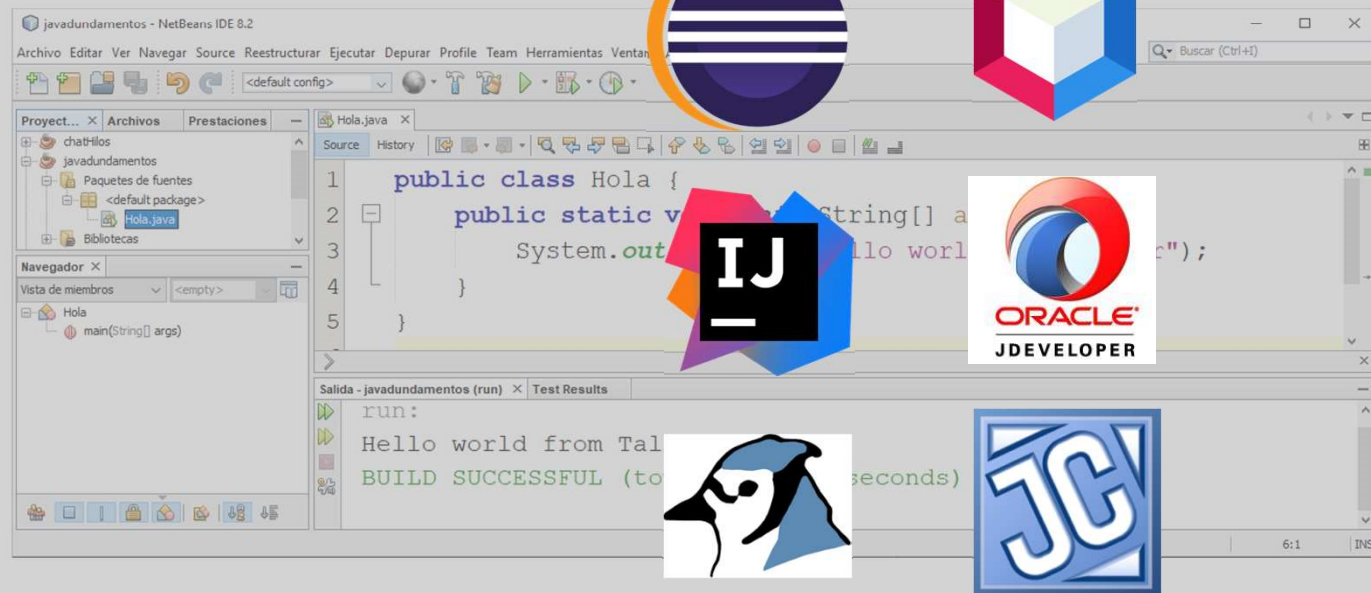
A screenshot of a Windows Command Prompt window titled 'Símbolo del sistema'. The window shows the following commands and output:

```
C:\Users\user10>"c:\Program Files\Java\jdk1.8.0_201\bin\javac.exe" Hola.java  
  
C:\Users\user10>java Hola  
Hello world from Taller  
  
C:\Users\user10>
```

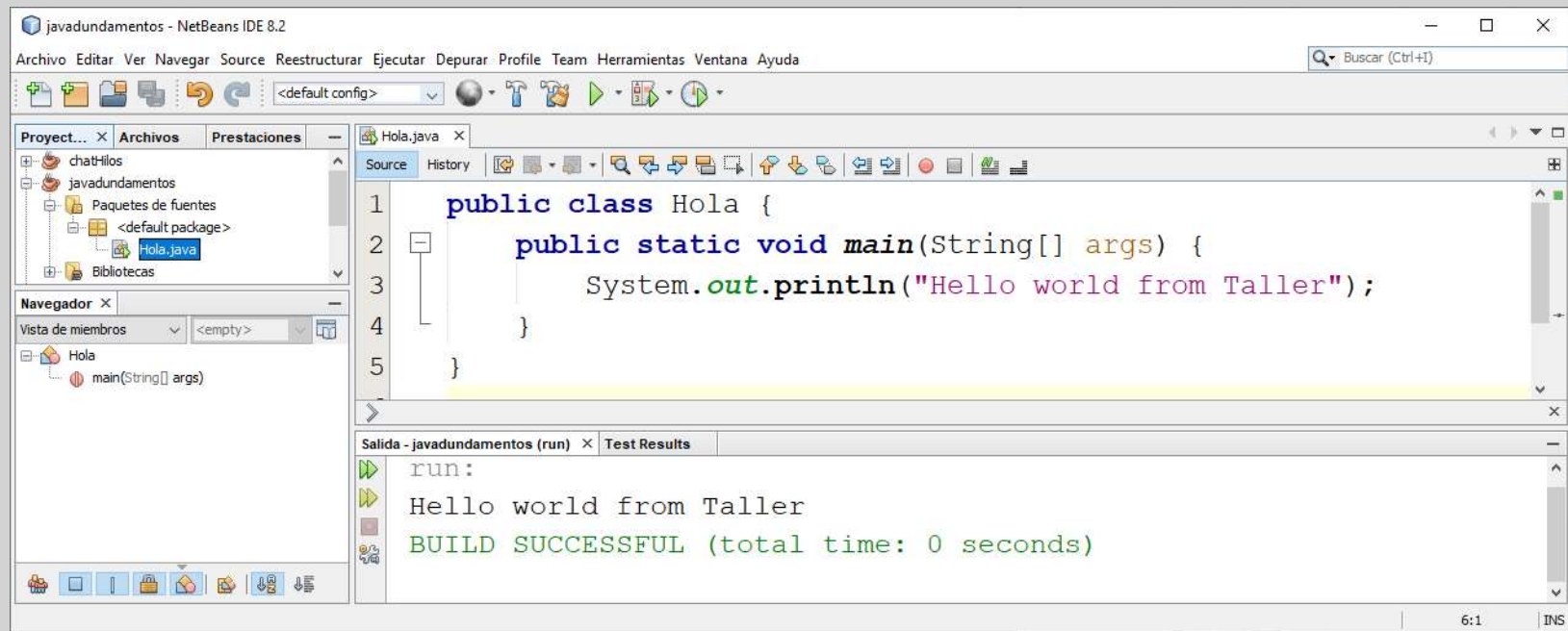

(IDE)

- **Integrated Development environment (IDE)** es una aplicación que provee un conjunto de herramientas para el desarrollar mas rápido y eficiente.

- Eclipse IDE.
- Netbeans IDE
- IntelliJ IDEA
- BlueJ
- JDeveloper
- JCreator



Netbeans



Mas teoría... Tipos primitivos

1. **byte**: The byte data type is an 8-bit signed integer.
2. **short**: The short data type is a 16-bit signed integer.
3. **int**: The int data type is a 32-bit signed integer. It has a maximum value of 2,147,483,647.
4. **long**: The long data type is a 64-bit signed integer.
5. **float**: The float data type is a single-precision 32-bit floating point.
6. **double**: The double data type is a double-precision 64-bit floating point.
7. **boolean**: The boolean data type has only two possible values: true and false.
8. **char**: The char data type is a single 16-bit Unicode character.

- ¿Y el String es tipo primitivo?
- No... es Clase

Arreglos

- Los arreglos Arrays son contenedores de un numero fijo de valores de un tipo determinado.
- `Int[] myArray = new int[10];`

```
public class Arreglos {  
    public static void main(String[] args) {  
        int[] unArreglo;  
        unArreglo = new int[5];  
  
        unArreglo[0] = 10;  
        unArreglo[1] = 20;  
        unArreglo[2] = 30;  
        unArreglo[3] = 40;  
        unArreglo[4] = 50;  
  
        System.out.println("Valor en la posicion 0: " + unArreglo[0]);  
        System.out.println("Valor en la posicion 0: " + unArreglo[1]);  
        System.out.println("Valor en la posicion 0: " + unArreglo[2]);  
        System.out.println("Valor en la posicion 0: " + unArreglo[3]);  
        System.out.println("Valor en la posicion 0: " + unArreglo[4]);  
    }  
}
```

Control de flujo

- Todos los códigos son ejecutados secuencialmente de arriba a bajo, Pero podemos controlar el flujo usando sentencias condicionales, ciclos y múltiples ramas de ejecución.

If/Else and Switch

- Sentencias If/else ejecuta una sección de código basado en una condicionante, si esta es verdadera.

```
if (someExpression)
    statement1
else
    statement2
```

```
int a = 10;
int b = 20;
int c = 30;

if( a + b == c ){
    System.out.println(a + " + " + b + " = " + c );
}else{
    System.out.println(a + " + " + b + " != " + c );
}
```

Control de flujo

Switch statement

- La sentencia switch permite tener varias rutas de ejecución.

```
int dayOfWeek = 1;
String dayString="";
switch (dayOfWeek) {
    case 1: dayString = "Monday";
            break;
    case 2: dayString = "Tuesday";
            break;
    case 3: dayString = "Wednesday";
            break;
    case 4: dayString = "Thursday";
            break;
    case 5: dayString = "Friday";
            break;
    case 6: dayString = "Saturday";
            break;
    case 7: dayString = "Sunday";
            break;
}
System.out.println(dayString);
```

Ciclos

While loop

- La sentencia while, ejecuta bloques de código continuamente, mientras la condición es verdadera.

```
while (expression) {  
    statement(s)  
}
```

```
int counter = 1;  
while (counter < 11) {  
    System.out.println("Count is: " + counter);  
    counter++;  
}
```

For Loop

- La sentencia for es una manera mas compacta de itera sobre rangos de valores.

```
for (initialization; termination-condition; increment) {  
    statement(s)  
}
```

```
for (int i = 1; i <= 10; i++) {  
  
    System.out.println("Value of i is: " + i);  
  
}
```

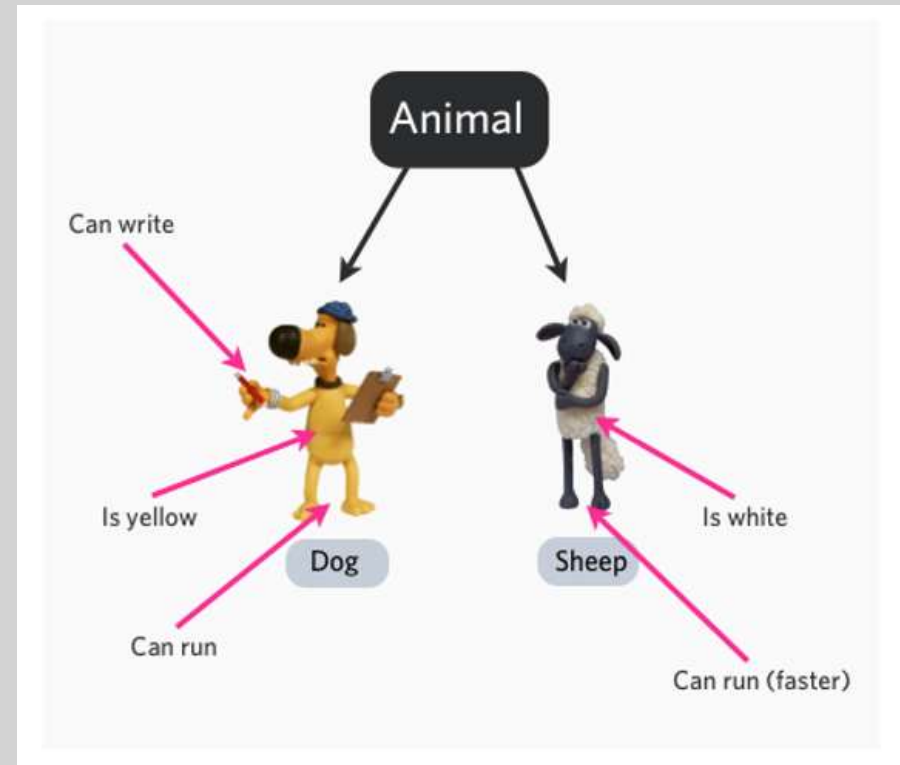
Orientación a objetos

- Java es un lenguaje que soporta **programación orientado a objetos**.
- **Objeto:**
 - Cosa material inanimada, generalmente de tamaño pequeño o mediano, que puede ser percibida por los sentidos.
 - Persona o cosa a la que va dirigida una acción o pensamiento.
 - Es alguien, algo en el mundo real.
 - Pueden ser concretos/físicos o abstractos/ideas



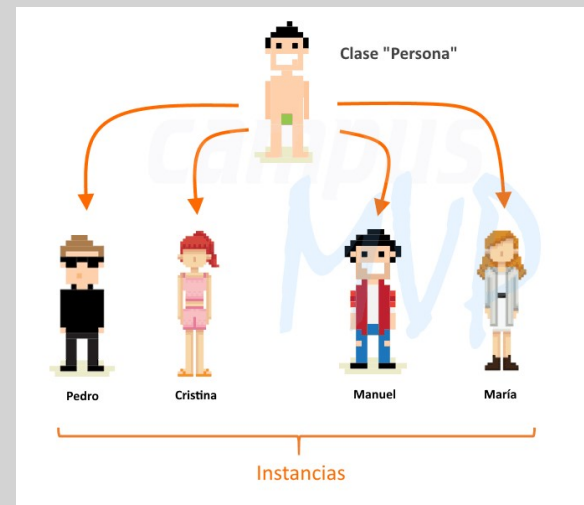
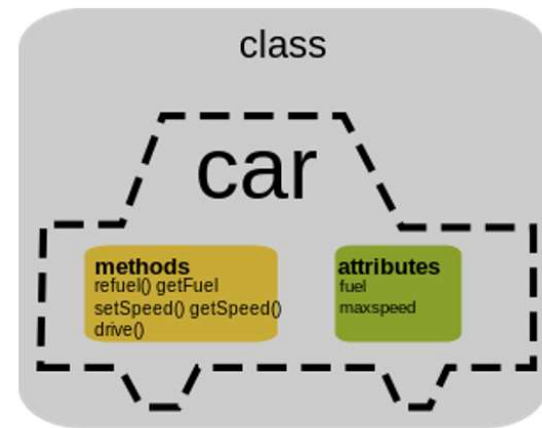
Orientación a objetos

- Objeto (en software) es un conjunto de características (estados) y comportamiento (operaciones).
- Estados son valores propios del objeto, también llamados **atributos**.
- Operaciones son capacidades del objeto, también llamados **métodos**.



Clases

- **Clase** es un prototipo (molde) por los cuales los objetos son creados.
- Modela los métodos y atributos de un objeto del mundo real.



Classes



Clases

Vehicle

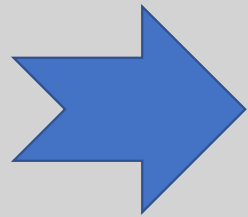
int speed

int gear = 1

void changeGear(int newGear)

void speedUp(int increment)

void printState()



```
class Vehicle {  
    int speed = 0;  
    int gear = 1;  
  
    void changeGear(int newGear) {  
        gear = newGear;  
    }  
  
    void speedUp(int increment) {  
        speed = speed + increment;  
    }  
  
    void printStates() {  
        System.out.println(" speed:" + speed + " gear:" + gear);  
    }  
}
```

Constructor: Permiten la creación de un objeto, similares a funciones pero:
Se tienen que llamar con la clase.

No tiene tipo de retorno, ya que devuelve el objeto

```
public Vehicle(int s, int g) {  
    speed = s;  
    gear = g;  
}
```

```
Vehicle vehicle = new Vehicle(4, 2);
```

```
public Vehicle() {  
}  
  
public Vehicle(int s, int g) {  
    gear = g;  
    speed = s;  
}
```

Classes

- Usando el vehículo

```
public class UseVehicle {  
    public static void main(String[] args) {  
        Vehicle auto1 = new Vehicle();  
  
        auto1.printState();  
  
        auto1.speedUp(5);  
        auto1.printState();  
  
        auto1.changeGear(2);  
        auto1.printState();  
    }  
}
```

Modificadores de acceso

- Modificadores de acceso

Access Levels				
Modifier	Class	Package	Subclass	All Other
<i>public</i>	Y	Y	Y	Y
<i>protected</i>	Y	Y	Y	N
<i>Default</i>	Y	Y	N	N
<i>private</i>	Y	N	N	N

```
public class Vehicle {  
    private int speed = 0;  
    private int gear = 1;  
  
    public void changeGear(int newGear) {  
        gear = newGear;  
    }  
    public void speedUp(int increment) {  
        speed = speed + increment;  
    }  
    public void printState() {  
        System.out.println(" speed: " + speed + " gear: " + gear);  
    }  
}
```

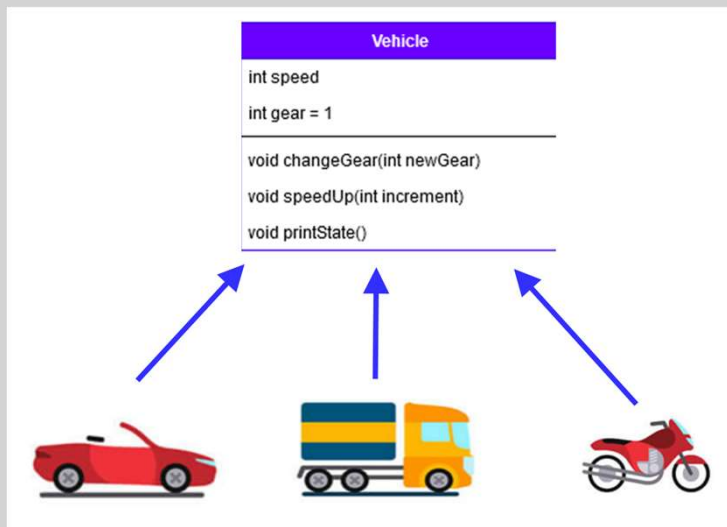
- Getter y Setter

```
public int getSpeed() {  
    return speed;  
}  
public void setSpeed(int speed) {  
    this.speed = speed;  
}
```

```
auto1.speed = 5;  
auto1.setSpeed(5);
```

Herencia

- Mecanismo para organizar y estructurar software. Define un relación de tipo parent-child entre dos objetos diferentes.
- POO permite a las clases heredar métodos y atributos de otra clase.
- Para heredar en java se usa la forma **clasehija extends clasepadre**



```
public class Car extends Vehicle {  
    int numOfSeats;  
    //Set of statements defining  
    //a car's state and behavior  
}  
  
public class Truck extends Vehicle {  
    public int loadWeight;  
    //Set of statements defining  
    //a truck's state and behavior  
}
```

Herencia

- Dos palabras reservadas al uso de herencia son **this** y **super**.
- **this** para referirse a miembros de la clase hija, y **super** para referirse a miembros de la clase padre.

```
public class Car extends Vehicle {  
    int numOfSeats;  
  
    void printStates() {  
        super.printStates();  
        System.out.println(" Number of Seats:" + numOfSeat);  
    }  
}
```


Polimorfismo: Sobrecarga y Anulación

- Polimorfismo: significa que diferentes clases del mismo padre tienen comportamiento diferente.
- Overload (sobrecarga): Es tener dos métodos con el mismo nombre pero diferente número o tipo de argumentos.
- Override: Ocurre cuando una clase hereda de una superclase pero genera su propia implementación de un método.

```
public int sumar(int x, int y){  
    return x + y;  
}  
public double sumar(double x, double y){  
    return x + y;  
}  
public int sumar(int x, int y, int z){  
    return x + y + z;  
}
```

```
public class Car extends Vehicle {  
    int numOfSeats;  
    public void speedUp(int increment) {  
        speed = speed + increment + 2;  
    }  
}
```

```
Car car = new Car();  
car.speedUp(2);
```

Polimorfismo: Sobrecarga y Anulación

- Clase Vehicle

```
public void showInfo() {  
    System.out.println("The vehicle has a speed of: " + this.speed  
        + " and at gear " + this.gear);  
}
```

- Clase Truck

```
public void showInfo() {  
    super.showInfo();  
    System.out.println("The truck has is carrying a load of: "  
        + this.loadWeight);  
}
```

- Clase Car

```
public void showInfo() {  
    super.showInfo();  
    System.out.println("The car has "  
        + this.numOfSeats + " seats.");  
}
```

Interface, combinar herencia y polimorfismo

- Interface: es un contrato entre una clase y el mundo exterior. Cuando una clase implementa una interface, debe proveer un comportamiento especificado en una interface.

```
public interface IVehicle {  
    void changeGear(int newValue);  
    void speedUp(int increment);  
}
```

```
* @author user10  
*/  
public class Vehicle implements IVehicle{  
      
}
```

Vehicle is not abstract and does not override abstract method speedUp(int) in IVehicle

(Alt-Enter shows hints)

```
public class Vehicle implements IVehicle{  
6  
7
```

Implement all abstract methods
Make class Vehicle abstract

```
public class Vehicle implements IVehicle{  
  
    @Override  
    public void changeGear(int newValue) {  
    }  
  
    @Override  
    public void speedUp(int increment) {  
    }  
}
```

Interface, combinar herencia y polimorfismo

```
class Vehicle implements IVehicle {  
    int speed = 0;  
    int gear = 1;  
  
    public void changeGear(int newValue) {  
        gear = newValue;  
    }  
  
    public void speedUp(int increment) {  
        speed = speed + increment;  
    }  
  
    void printStates() {  
        System.out.println(" speed:" + speed + " gear:" + gear);  
    }  
}
```

Practica de P00

```
IVehicle vehicle1, vehicle2, vehicle3;  
vehicle1 = new Vehicle(50,2);  
vehicle2 = new Car(50,2,4);  
vehicle3 = new Truck(40,2,500);  
System.out.println("Vehicle 1 info:");  
vehicle1.showInfo();  
System.out.println("\nVehicle 2 info:");  
vehicle2.showInfo();  
System.out.println("\nVehicle 3 info:");  
vehicle3.showInfo();
```

Excepciones

- Para el manejo de errores Java usa excepciones, una excepción es un Objeto de error generado cuando ocurre el mismo, este es heredado de Exception.
- Contiene métodos y atributos para el manejo de error.
- Este objeto puede ser atrapado (try catch) o lanzado (throw) para que sea manejado por el sistema o el usuario.
- Probemos.

```
public class UseExceptions {  
    public static void main(String[] args) {  
        System.out.println("Hola mundo desde curso!");  
        String nullString = null;  
        System.out.println("sentencia try ");  
        String partialString = nullString.substring(1);  
        // Execution will break before reaching this line  
        System.out.println("partial string is: " + partialString);  
    }  
}
```

Excepciones: try catch

- El bloque try{ }catch(Exception ex) { }, permite atrapar una excepción conocida si esta es lanzada por el JVM, en caso contrario el código funciona con normalidad.

```
public class UseExceptions {  
    public static void main(String[] args) {  
        System.out.println("Hola mundo desde curso!");  
        String nullString = null;  
        try {  
            System.out.println("sentencia try ");  
            String partialString = nullString.substring(1);  
            // Execution will break before reaching this line  
            System.out.println("cadena partial es: " + partialString);  
        } catch( Exception ex){  
            System.out.println("Error es: " + ex.getMessage());  
            ex.printStackTrace(); // <-- Mínimo de un try catch  
        }  
    }  
}
```

Colecciones: listas y mapas

- Las Java Collections son clases (e interfaces) localizados en java.útil.*.
- Permite contener dentro de ellos conjuntos de elementos, haciendo referencia las estructuras de datos como listas, pilas, colas, y diccionarios.
- Lista: contiene un grupo de objetos que pueden ser manipulados, pueden ser únicos o duplicados.
- Diccionarios, mapean llaves a valores y no pueden contener llaves duplicada.

Listas: List	Mapas: Map
ArrayList	HashMap
LinkedList	Hashtable
Stack	Properties
Vector	TreeMap

Colecciones: listas

- ArrayList: Es una implementación de lista basada en arreglos dinámicos.

```
ArrayList<String> androids = new ArrayList<String>();  
// Adding elements  
androids.add("Cupcake");  
androids.add("Donut");  
androids.add("Eclair");  
androids.add("Froyo");  
androids.add("Gingerbread");  
androids.add("Honeycomb");  
androids.add("Ice Cream Sandwich");  
androids.add("Jelly Bean");  
System.out.println("Size of ArrayList: " + androids.size());  
// Display the contents of the array list  
System.out.println("The ArrayList has the following elements: "  
    + androids);  
  
System.out.println("Deleting second element...");  
androids.remove(3);  
System.out.println("Size after deletions: " + androids.size());  
System.out.println("Contents after deletions: " + androids);
```

Colecciones: Mapas

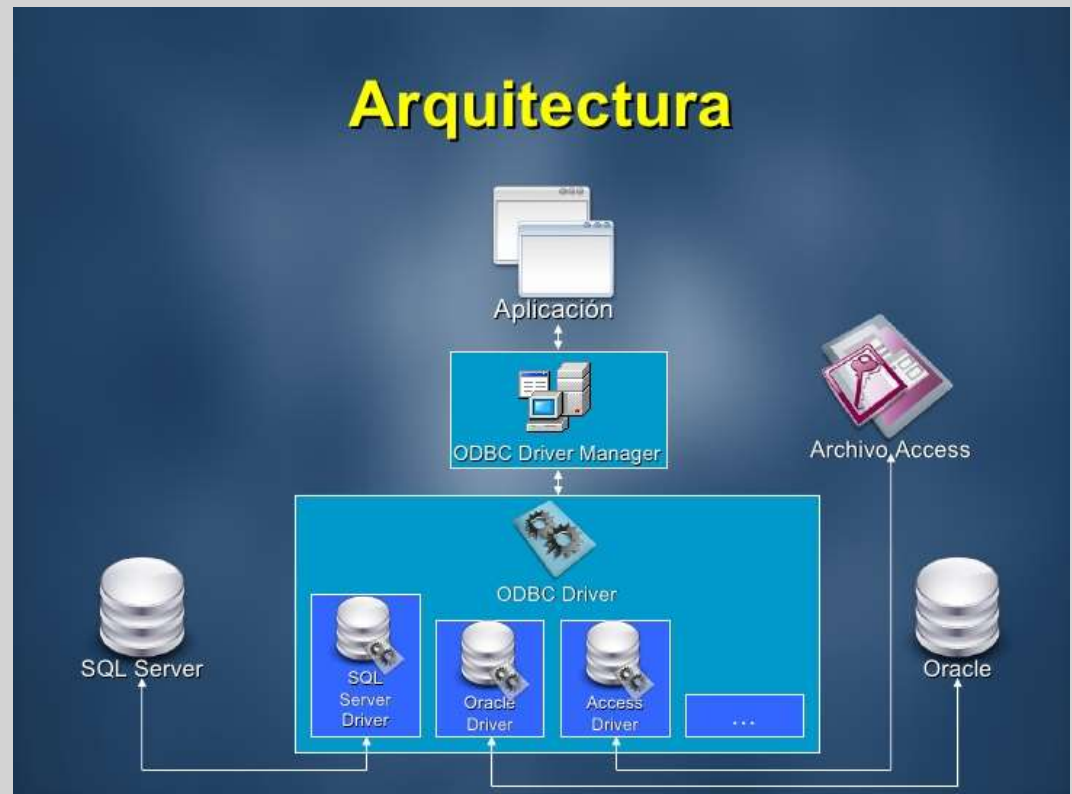
- HashMap. Es una implementación de un diccionario (Map), permite agregar elementos nulos.

```
// Creating a HashMap
Map<String,Double> androids = new HashMap<String,Double>();
// Adding elements
androids.put("Cupcake", new Double(1.5) );
androids.put("Donut",new Double(1.6));
androids.put("Eclair", new Double(2.1));
androids.put("Froyo", new Double(2.2));
//recover values and keys
List<String> listKeys = new ArrayList<>(androids.keySet());
List<Double> listValues = new ArrayList<>(androids.values());
for( String key : listKeys){
    System.out.print(key + ": ");
    System.out.println( androids.get(key) );

    int idx = listKeys.indexOf(key);
    System.out.println(" , " + listValues.get(idx));
}
```

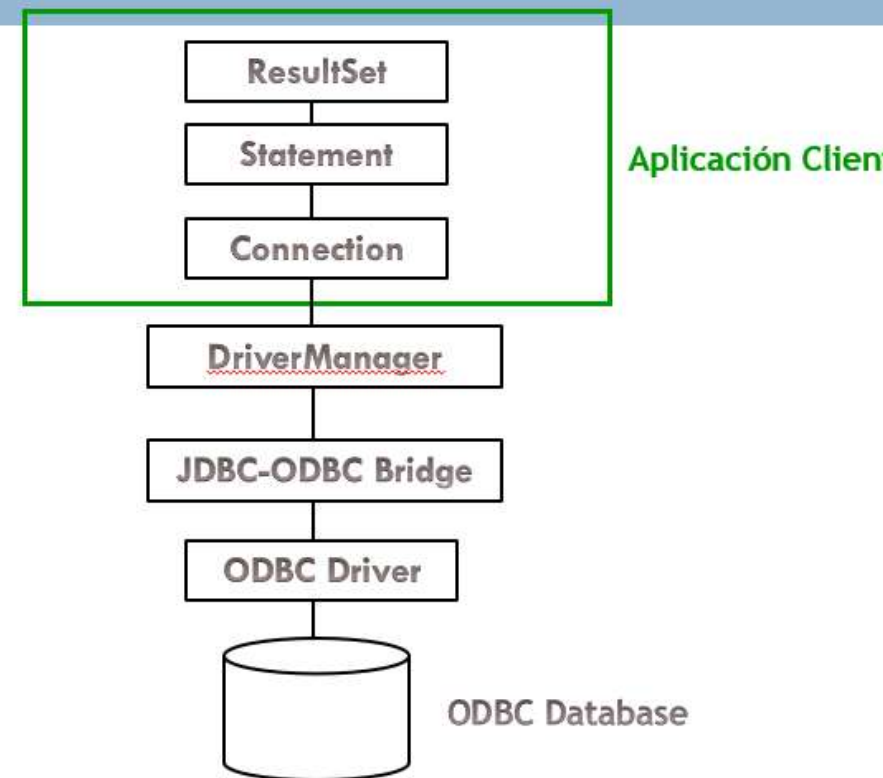
Java y Bases de datos: ODBC

- Open DataBase Connectivity (ODBC) es un estándar de acceso a las bases de datos desarrollado por SQL Access Group (SAG) en 1992.
- El objetivo de ODBC es hacer posible el acceder a cualquier dato desde cualquier aplicación, sin importar qué sistema de gestión de bases de datos (DBMS) almacene los datos.



Java y Bases de datos: JDBC

- API de Java para ejecutar sentencias SQL
- JDBC posibilita básicamente tres cosas:
 - Establecer una conexión con una base de datos desde Java
 - Enviar sentencias SQL a través de dicha conexión
 - Procesar los resultados
- La JDBC 3.0 API comprende 2 paquetes:
 - java.sql
 - javax.sql



JDBC: Connection

- **Connection:** es la clase que permite comunicar con el servidor de BD, mediante un configuración llamada cadena de conexión.
- **Statement/PreparedStatement:** es la clase que permite mandar una comando o instrucción de base datos como SQL, procedimientos almacenados.
- **ResultSet:** es la clase que almacena los resultados devueltos si los hay de la instrucción anterior.

```
try {
    Class.forName("com.mysql.jdbc.Driver");
    Connection con = DriverManager.getConnection(
        "jdbc:mysql://localhost:3306/spring", //URL
        "root", //USER
        "admin"); //PASS
    String sql = "SELECT * FROM Usuarios WHERE username = ?";
    PreparedStatement cmd = con.prepareStatement(sql);
    String useremail = "admin@email.com";
    cmd.setString(1, useremail);
    ResultSet rs = cmd.executeQuery();
    while(rs.next()){
        String userpass = rs.getString("password");
        String usernombre = rs.getString("nombre");
        System.out.println("username: " + useremail +
            ",password " + userpass + ",nombre real" + usernombre );
    }
    rs.close();
    cmd.close();
    con.close();
} catch (SQLException | ClassNotFoundException ex) {
    ex.printStackTrace(System.out);
}
```



¿Preguntas?

