

UD 8

FRAMEWORKS

David Rizo Valero
Pedro J. Ponce de León

Versión 20131125





1. Introducción
2. Java Collection Framework
3. Ejemplo de un framework propio
4. Librerías y toolkits JDK
5. JDBC
6. Tratamiento de XML
7. Logging
8. Hibernate
9. Apache Commons
10. Otros frameworks



- **Librerías y frameworks:**
 - Implementan funcionalidades útiles que el propio lenguaje de programación no incorpora
- **Framework:**
 - 'Esqueletos' para fines específicos que debemos completar y personalizar mediante la implementación de interfaces, herencia de clases abstractas o ficheros de configuración.
 - El framework llama a nuestros métodos.
- **Librería:**
 - Conjunto de clases que realizan funciones más o menos concretas.
 - Nosotros llamamos a los métodos de las librerías.



- Utilizan los tres mecanismos principales de la POO:
 - *Encapsulación* → ocultar implementación
 - *Polimorfismo* → mismo código para diferentes tipos de objetos
 - *Herencia* → reuso de funcionalidad.
 - Dividimos el universo en categorías, conjuntos de objetos, o sea clases, y subclases. Éstas últimas reutilizan la implementación y/o el interfaz de sus clases base.
- Esto permite

**Programar para un interfaz,
no para una implementación**



■ Inversión de control

- El framework llama a nuestros métodos.
- El control de cuándo se llaman reside en el framework.
- El usuario del framework especifica métodos a invocar, normalmente proporcionando implementación para métodos declarados en interfaces o clases abstractas.
- Implementación del ***Principio de Hollywood***: *"no nos llames; nosotros te llamaremos"*.

Frameworks. Ejemplo de inversión de control (*)



- Pedir que el usuario introduzca un dato:
 - Por línea de comando

```
String usuario=null;
String pregunta = null;
BufferedReader br =
    new BufferedReader(new InputStreamReader(System.in));

System.out.print("Diga su nombre: ");
usuario = br.readLine();
procesarUsuario(usuario);

System.out.print("Diga su pregunta: ");
pregunta = br.readLine();
procesarPregunta(pregunta);
```

(*) inspirado en <http://martinfowler.com/bliki/InversionOfControl.html>

Frameworks. Ejemplo de inversión de control (*)



- Pedir que el usuario introduzca un dato:
 - Mediante un diálogo, usando un hipotético framework de gestión de ventanas:

```
CuadroDialogo d = new CuadroDialogo("Diga su nombre");  
// registra el método a invocar cuando el usuario  
// pulse tecla INTRO  
d.registra(procesarUsuario);  
// muestra el diálogo e interactúa con el usuario.  
// El contenido del cuadro de texto se pasa al método  
// procesarUsuario()  
d.mostrar();  
  
d = new CuadroDialogo("Diga su pregunta");  
d.registra(procesarPregunta);  
d.mostrar();
```

(*) inspirado en <http://martinfowler.com/bliki/InversionOfControl.html>



- En el ejemplo de línea de comando, tenemos control absoluto sobre el flujo del programa.
- En el del cuadro de diálogo, no. Parte del flujo está controlado por el framework de gestión de ventanas.
- Se ha ***invertido*** el control. El framework llama a nuestros métodos, en lugar de ser nosotros quienes llamamos al código del framework para pedirle la cadena de entrada.



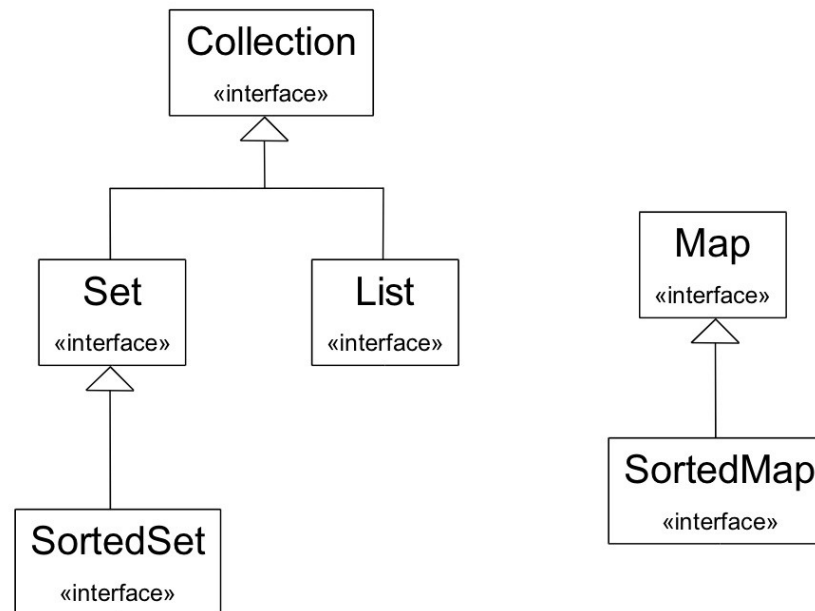
- Ejemplo: **JUnit** es un framework.
 - Funciona mediante inversión de control. Nosotros especificamos una serie de métodos que se tienen que ejecutar (metodos setUp(), testAlgo(),...). El framework se encarga de
 - Arrancar el proceso de prueba
 - Cargar las clases que contienen las pruebas
 - Invocar a los métodos @BeforeClass
 - Para cada test
 - Invocar a los métodos @Before antes de cada test
 - Invocar al método de test
 - Invocar a los métodos @After tras cada test
 - etc...
- JUnit controla el flujo de control, nosotros sólo indicamos qué métodos ejecutar en el momento de hacer las pruebas.

Java Collection Framework (JCF)

Java Collection Framework (JCF)



- Conjunto de clases incluido en el JDK representando tipos abstractos de datos básicos: pilas, colas, vectores, mapas, etc...
- Tiene formato de framework porque está diseñado para que cualquiera pueda usarlo como base para realizar su propia implementación de cada tipo de dato (extendiendo los interfaces)





```
public interface Collection {
    int size();
    boolean isEmpty();
    boolean contains(Object element);
    boolean add(Object element);
    boolean remove(Object element);
    Iterator iterator();
    boolean containsAll(Collection c);
    boolean addAll(Collection c);
    boolean removeAll(Collection c);
    boolean retainAll(Collection c);
    void clear();
    Object[] toArray();
    Object[] toArray(Object a[]);
}
```

```
public interface Map {
    Object put(Object key, Object value);
    Object get(Object key);
    Object remove(Object key);
    boolean containsKey(Object key);
    boolean containsValue(Object value);
    void putAll(Map t);
    public Set keySet();
    public Collection values();
    public Set entrySet();

    public interface Entry {
        Object getKey();
        Object getValue();
        Object setValue(Object value);
    }
}
```

```
public interface SortedSet extends Set {
    SortedSet subSet(Object fromElement, Object toElement);
    SortedSet headSet(Object toElement);
    SortedSet tailSet(Object fromElement);
    Object first();
    Object last();
    Comparator comparator();
}
```

```
public interface Set
    extends Collection {
    // intentionally empty.
}
```



```
public interface List extends Collection {  
    Object get(int index);  
    Object set(int index, Object element);  
    void add(int index, Object element);  
    Object remove(int index);  
    boolean addAll(int index, Collection c);  
    int indexOf(Object o);  
    int lastIndexOf(Object o);  
    ListIterator listIterator();  
    ListIterator listIterator(int index);  
    List subList(int from, int to);  
}
```

```
public interface SortedMap extends Map {  
    SortedMap subMap(Object fromKey, Object toKey);  
    SortedMap headMap(Object toKey);  
    SortedMap tailMap(Object fromKey);  
    Object first();  
    Object last();  
    Comparator comparator();  
}
```



■ Iteradores

```
public interface Iterator {  
    boolean hasNext();  
    Object next();  
    void remove();  
}
```

```
public interface ListIterator extends Iterator {  
    void add(Object o);  
    int nextIndex();  
    boolean hasPrevious();  
    Object previous();  
    int previousIndex();  
    void set(Object o);  
}
```



- JCF proporciona implementaciones de referencia para cada interface
 - Si sólo usamos esas implementaciones, usaremos JCF como **librería y no como framework**

Interfaces	Implementations			
	Hash Table	Resizable Array	Balanced Tree	Linked List
Set	HashSet		TreeSet	
List		ArrayList		LinkedList
Map	HashMap		TreeMap	



```
public class Collections {  
    public static int binarySearch(List list, Object key) { /*código*/}  
    public static void copy(List dest, List src) { /*código*/}  
    public static void fill(List list, Object o) { /*código*/}  
    public static Object max(Collection coll) { /*código*/}  
    public static Object min(Collection coll) { /*código*/}  
    public static void reverse(List list) { /*código*/}  
    public static void shuffle(List list) { /*código*/}  
    public static void shuffle(List list, Random rnd) { /*código*/}  
    public static void sort(List list) { /*código*/}  
    public static void sort(List list, Comparator c) { /*código*/}  
    // etc...  
}
```

- JCF no tiene una implementación de referencia de Comparator

```
public interface Comparator {  
    int compare(Object o1, Object o2);  
    void equals(Object obj);  
}
```




- Uso del framework en modo librería

(imprime: [2,7,9])

```
SortedSet<Integer> cjtoOrdenado = new TreeSet<Integer>();  
cjtoOrdenado.add(7);  
cjtoOrdenado.add(2);  
cjtoOrdenado.add(9);
```

```
System.out.println(cjtoOrdenado.toString());
```

- Uso como framework, nosotros completamos parte del código mediante implementación de un interfaz

```
ArrayList<MiClase> v = ..... // inicialización  
// implementación anónima del interfaz comparator  
Comparator c = new Comparator<MiClase>() {  
    public int compare(MiClase arg0, MiClase arg1) { /* código */}  
    public boolean equals(Object arg1) { /* código */}  
};  
// para ordenar, el framework JCF llama a nuestro método  
// de comparación entre objetos MiClase  
Collections.sort(v,c);
```

Ejemplo de un framework



Simulación de carreras de coches

- Framework propio (de juguete) para la realización de carreras de coches
 - No necesariamente coches
- El framework nos da hecho:
 - La definición de un circuito
 - La simulación del proceso de la carrera
- Cómo usarlo
 - Especificar qué coches van a intervenir y cuantas vueltas hay que dar.
 - Extendiendo e implementando las clases que se nos indica en la documentación (hipotética)

Vehiculo, ICorredor, ICocheAuxiliar



Interfaces

```
interface ICorredor {  
    void dar_vuelta ();  
}
```

```
interface ICocheAuxiliar {  
    boolean en_pista ();  
    void toggle ();  
}
```



Clase Vehiculo

```
abstract class Vehiculo {  
    public Vehiculo (String m) { marca = m; }  
    public String get_marca () { return marca; }  
  
    private String marca;  
}
```



Clase Circuito

(implementada para los interfaces ICorredor e ICocheAuxiliar)

```
class Circuito {  
    private int longitudkm;  
    private int aforo;  
    private String nombre;  
    private List<ICorredor> lv;  
    private ICocheAuxiliar sc;  
  
    public Circuito (String n) {  
        sc = null;  
        nombre = n;  
        lv = new ArrayList<ICorredor>();  
    }  
  
    ...  
}
```



Clase Circuito

...

```
public int get_nvehiculos ()
    { return lv.size(); }
public int get_longitudkm ()
    { return longitudkm; }
public int get_aforo ()
    { return aforo; }

public void add_vehiculo (ICorredor c)
    { lv.add(c); }

public void add_safetycar (ICocheAuxiliar ca)
    { sc = ca; }
```

...



Clase Circuito

```
public void simular_carrera (int nv) {
    System.out.println("Bienvenidos al circuito de " + nombre);
    if (sc != null) {
        System.out.println("Comienza la carrera:\n");
        while (nv > 0) {
            System.out.println("[");
            for (ICorredor v : lv) {
                if (!sc.en_pista ())
                    v.dar_vuelta ();
                else
                    System.out.println("SafetyCar en pista");
            }
            System.out.println("]\n");
            nv--;

            if((new Random()).next_int(100) > 50) {
                sc.toggle ();
            }
        }
    } else
        System.out.println(";No hay safety-car!");
}}
```




Uso del framework

```
class Coche extends Vehiculo {  
    public Coche (String marca) {  
        super (marca);  
    }  
}
```



Uso del framework

```
class SafetyCar extends Coche implements ICocheAuxiliar {  
  
    public SafetyCar (String marca) {  
        super (marca);  
        m_en_pista = false;  
    }  
    public boolean en_pista () { return m_en_pista;}  
    public void toggle () { m_en_pista = !m_en_pista;}  
    public void set_en_pista (boolean v)  
        { m_en_pista = v; }  
  
    private boolean m_en_pista;  
}
```



Uso del framework

```
class Formula1 extends Coche implements ICorredor {  
  
    public Formula1 (String marca) {  
        super (marca);  
        nvueltas = 0;  
    }  
  
    public void dar_vuelta () {  
        nvueltas++;  
        System.out.println(  
            "Formula1["+get_marca()+"], vuelta "+nvueltas);  
    }  
  
    protected int nvueltas;  
}
```



Uso del framework

```
class CamionFormula1 extends Formula1 {  
  
    public CamionFormula1 (String marca) {  
        super (marca);  
    }  
  
    public void dar_vuelta () {  
        nvueltas++;  
        System.out.println(  
            "CamionFormula1[" + get_marca() +  
            "], vuelta " + nvueltas);  
    }  
}
```



Uso del framework Programa principal

```
class CarreraF1 {  
    public static final void main (String[] args) {  
        int MAXCOCHES = 3;  
  
        Circuito c = new Circuito("Valencia");  
        SafetyCar sc = new SafetyCar ("BMW");  
  
        c.add_safetycar (sc);  
  
        System.out.println("Simulador de carreras");  
        ...  
    }  
}
```



Uso del framework Programa principal

```
...
for (int n = 0; n < MAXCOCHES; n++) {
    int rn = (new Random()).nextInt(100);
    if (rn < 50) { // Formula1
        String marca = "HRT"+n;
        c.add_vehiculo (new Formula1(marca));
    } else {
        String marca = "RENAULT"+n;
        c.add_vehiculo (new CamionFormula1 (marca));
    }
}
c.simular_carrera (7);
} // fin main
} // fin clase
```

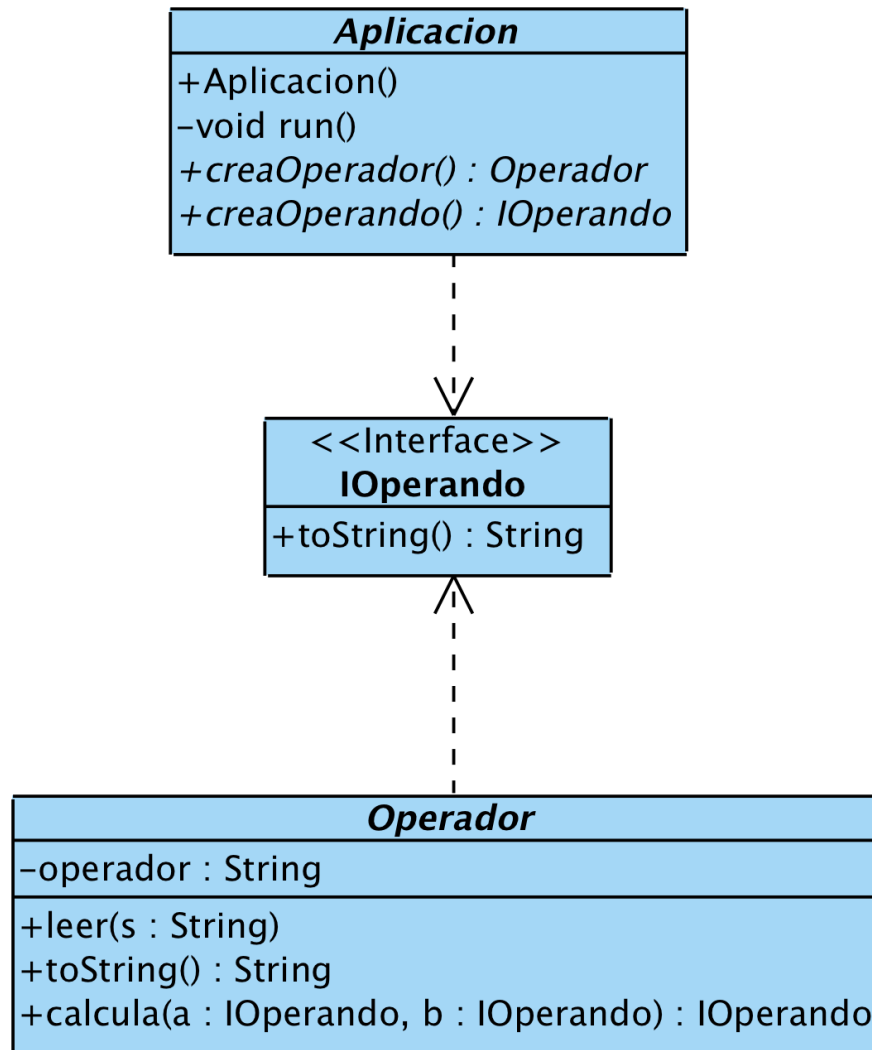


Operandos y operadores

- Framework propio (de juguete) para la realización de operaciones binarias entre dos operandos
 - No necesariamente números
- El framework nos da hecho:
 - Lectura de operandos
 - Lectura de operador
 - Escritura del resultado
 - Comprobaciones varias
- Cómo usarlo
 - Extendiendo e implementando las clases que se nos indica en la documentación (hipotética)

Aplicacion, IOperando, Operador

Ejemplo de un framework



Ejemplo de un framework



```
public interface IOperando {  
    String toString();  
    void lee(String cadena) throws ExcepcionFramworkOps;  
}
```

```
public abstract class Operador<TipoOperando extends IOperando> {  
    .....  
}
```

```
public abstract TipoOperando calcula(TipoOperando a, TipoOperando b) throws  
ExcepcionFramworkOps;  
}
```

```
public abstract class Aplicacion  
    <TipoOperando extends IOperando, TipoOperador extends Operador<TipoOperando>> {  
    public abstract TipoOperador creaOperador();  
    public abstract TipoOperando creaOperando();  
    .....  
}
```

Ejemplo de un framework



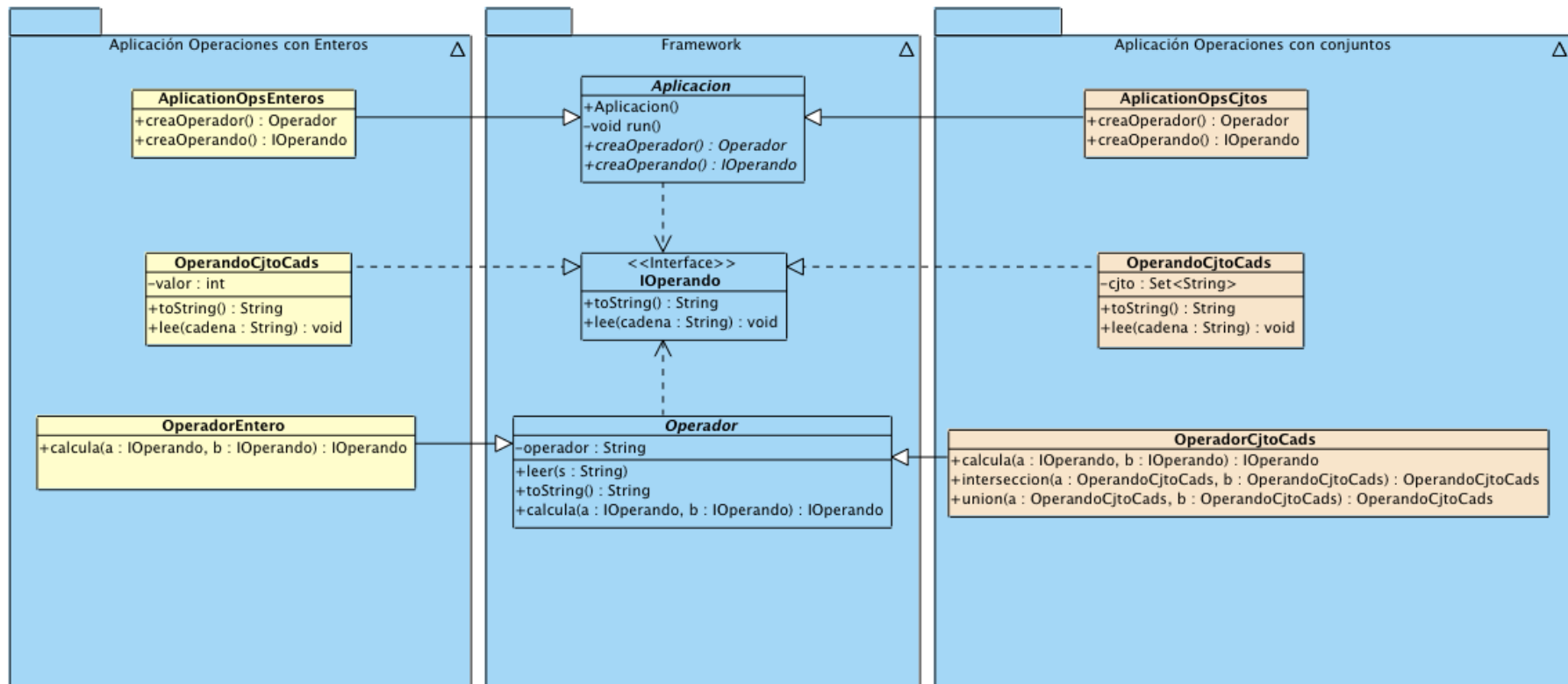
- El framework llama a nuestras clases (las que heredan de Aplicacion, Operador e implementan IOperando)

```
private void run() {  
    try {  
        Scanner scanner = new Scanner(System.in);  
        System.out.println("Introduce el primer operando:");  
        TipoOperando a = creaOperando();  
        a.lee(scanner.nextLine());  
        TipoOperador op = creaOperador();  
        System.out.println("Introduce la operación:");  
        op.lee(scanner.nextLine());  
        System.out.println("Introduce el segundo operador:");  
        TipoOperando b = creaOperando();  
        b.lee(scanner.nextLine());  
        TipoOperando resultado = op.calcula(a, b);  
        System.out.println("El resultado de la operación ");  
        System.out.print(a.toString()+op.toString()+b.toString());  
        + "=" + resultado.toString());  
    } catch (ExcepcionFrameworkOps e) {  
        System.err.println("Error: " + e.getMessage());  
    }  
}
```

Ejemplo de un framework



- Dos aplicaciones hechas con este framework: operaciones con enteros y operaciones con conjuntos





- Creación de una aplicación para operar con **enteros**:

```
public class AplicacionOpsEnteros extends Aplicacion<OperandoEntero,  
OperadorEntero> {  
    public static void main(String[] args) {  
        new AplicacionOpsEnteros();  
    }  
    @Override  
    public OperadorEntero creaOperador() {  
        return new OperadorEntero();  
    }  
    @Override  
    public OperandoEntero creaOperando() {  
        return new OperandoEntero();  
    }  
}
```



- Extendemos Operador

```
public class OperadorEntero extends Operador<OperandoEntero> {

    @Override
    public OperandoEntero calcula(OperandoEntero a, OperandoEntero b)
    throws ExcepcionFrameworkOps {
        if (toString().equals("+")) {
            return new OperandoEntero(a.getInt() + b.getInt());
        } else if (toString().equals("-")) {
            return new OperandoEntero(a.getInt() - b.getInt());
        } else {
            throw new ExcepcionFrameworkOps("Operando inválido: " +
                toString());
        }
    }
}
```



- Implementamos el interfaz IOperando

```
public class OperandoEntero implements IOperando {
    int valor;
    public OperandoEntero(int valor) {
        this.valor = valor;
    }
    public OperandoEntero() {
    }
    @Override
    public void lee(String cadena) throws ExcepcionFrameworkOps {
        try {
            valor = new Integer(cadena).intValue();
        } catch (NumberFormatException e) {
            throw new ExcepcionFrameworkOps("Error leyendo el operador
            entero: " + e.getMessage());
        }
    }
    public int getInt() {
        return valor;
    }
    public String toString() {
        return new Integer(valor).toString();
    }
}
```



- Ejemplo de ejecución:

Introduce el primer operando:

10

Introduce la operación:

-

Introduce el segundo operador:

4

El resultado de la operación

$10-4=6$

Ejemplo de un framework



- Creación de una aplicación para operar con **conjuntos de cadenas**:

```
public class AplicacionOpsCjtos extends Aplicacion<OperandoCjtoCads,  
    OperadorCjtoCads> {  
    public static void main(String[] args) {  
        new AplicacionOpsCjtos();  
    }  
    @Override  
    public OperadorCjtoCads creaOperador() {  
        return new OperadorCjtoCads();  
    }  
    @Override  
    public OperandoCjtoCads creaOperando() {  
        return new OperandoCjtoCads();  
    }  
}
```


Ejemplo de un framework



■ Extendemos Operador

```
public class OperadorCjtoCads extends Operador<OperandoCjtoCads> {
    @Override
    public OperandoCjtoCads calcula(OperandoCjtoCads a, OperandoCjtoCads b)
        throws ExcepcionFrameworkOps {
        if (toString().equals("U")) {
            return union(a,b);
        } else if (toString().equals("^")) {
            return interseccion(a,b);
        } else {
            throw new ExcepcionFrameworkOps("Operando inválido: " + toString());
        }
    }
    private OperandoCjtoCads interseccion(OperandoCjtoCads a,
        OperandoCjtoCads b) {
        Set<String> res = new TreeSet<String>(a.getCjto());
        res.retainAll(b.getCjto());
        return new OperandoCjtoCads(res);
    }
    private OperandoCjtoCads union(OperandoCjtoCads a, OperandoCjtoCads b) {
        Set<String> res = new TreeSet<String>(a.getCjto());
        res.addAll(b.getCjto());
        return new OperandoCjtoCads(res);
    }
}
```

Ejemplo de un framework



- Implementamos el interfaz IOperando (1/2)

```
public class OperandoCjtoCads implements IOperando {
    Set<String> cjto;

    public final Set<String> getCjto() {
        return cjto;
    }
    public OperandoCjtoCads(Set<String> valores) {
        this.cjto = valores;
    }
    public OperandoCjtoCads() {
        this.cjto = new TreeSet<String>();
    }
}
```

Ejemplo de un framework



■ Implementamos el interfaz IOperando (2/2)

```
/**
 * Lee la cadena de elementos separados por comas
 */
@Override
public void lee(String cadena) throws ExcepcionFrameworkOps {
    try {
        String [] cads = cadena.split(",");
        for (String string : cads) {
            cjto.add(string.trim());
        }
    } catch (NumberFormatException e) {
        throw new ExcepcionFrameworkOps("Error leyendo el operador: "
            + e.getMessage());
    }
}

public String toString() {
    return cjto.toString();
}
}
```

Ejemplo de un framework



- Ejemplo de ejecución:

Introduce el primer operando:

a,b,c

Introduce la operación:

U

Introduce el segundo operador:

b,d

El resultado de la operación

$[a, b, c] \cup [b, d] = [a, b, c, d]$

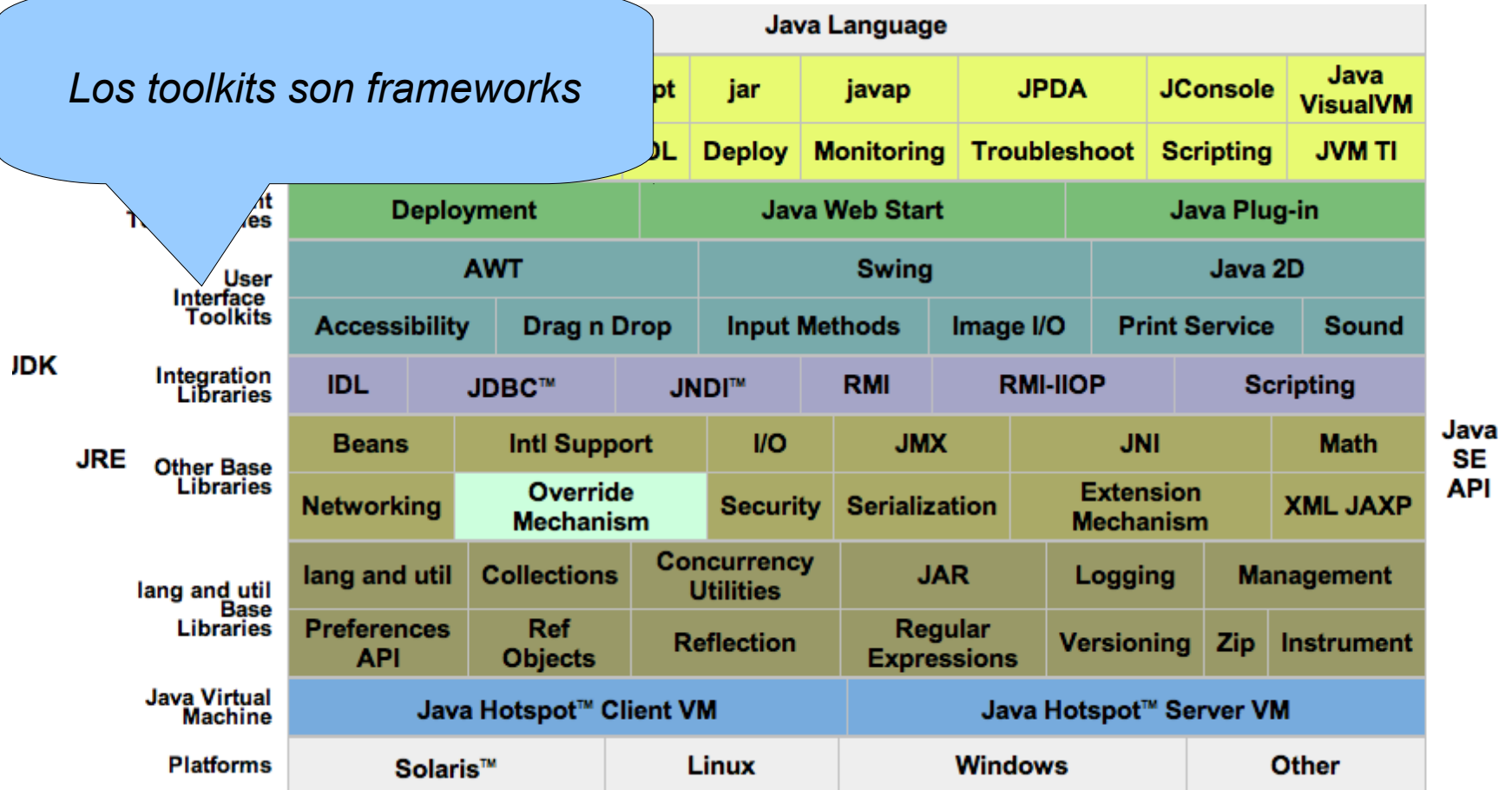
Librerías y toolkits JDK



- Conjunto de clases que incorporamos a nuestras aplicaciones
- No requieren que implementemos ni heredemos nada
- Son como cajas negras
- El fabricante nos proporciona una API (application program interface)
 - Conjunto de clases y sus métodos
- Podemos considerar librerías a la implementación de referencia del JCF incluido en el JDK:
 - `java.util.ArrayList`
 - `java.util.Stack`
 - `java.util.TreeSet`
 - ...
- En C++ disponemos de librerías similares: STD



Los toolkits son frameworks



JDBC



- Java DataBase Connection
- Realmente es un framework que utilizan los fabricantes de RDBMS para conectar Java con sus sistemas de bases de datos
- El JDK proporciona una serie de interfaces y clases abstractas que los fabricantes deben especializar
- Los desarrolladores utilizamos directamente las librerías de los fabricantes, a las que les denominamos *drivers* o conectores. P.ej.
 - MySQL: mysql-connector-java-5.1.18-bin
 - Oracle: ojdbc6.jar
 - SQLServer: sqljdbc4.jar
 - etc....
- Una lista completa de drivers se puede encontrar en <http://developers.sun.com/product/jdbc/drivers>



- Estructura básica:
 - 1º crear conexión
 - 2º usar la conexión para consultar / manipular la BBDD
 - Mediante la clase Statement
 - 3º cerrar la conexión

```
Class.forName("com.mysql.jdbc.Driver"); // cargamos driver  
// Cadena de conexión  
String dbURL = "jdbc:mysql://localhost/mibbdd";  
// Conectamos  
Connection con = DriverManager.getConnection(dbURL, "milogin", "mipassword");  
// CONSULTAMOS, INSERTAMOS, BORRAMOS usando con  
con.close(); // cerramos conexión al terminar
```



■ Inserción / actualización

```
Statement stmt = conexion.createStatement(); // usamos java.sql.Statement y no la de MySQL
String sqlInsercion = "INSERT INTO equipo (nombre, abreviatura) values ('Valencia', 'VAL'),
('Levante', 'LEV')";
int filasInsertadas = stmt.executeUpdate(sqlInsercion);
System.out.println("Se han insertado " + filasInsertadas + " registros");
```

■ Borrado

```
Statement stmt = conexion.createStatement();
String sqlBorrado = "delete from equipo where abreviatura = 'ALC' or abreviatura = 'HER'";
int filasBorradas = stmt.executeUpdate(sqlBorrado);
System.out.println("Se han borrado " + filasBorradas + " registros");
```

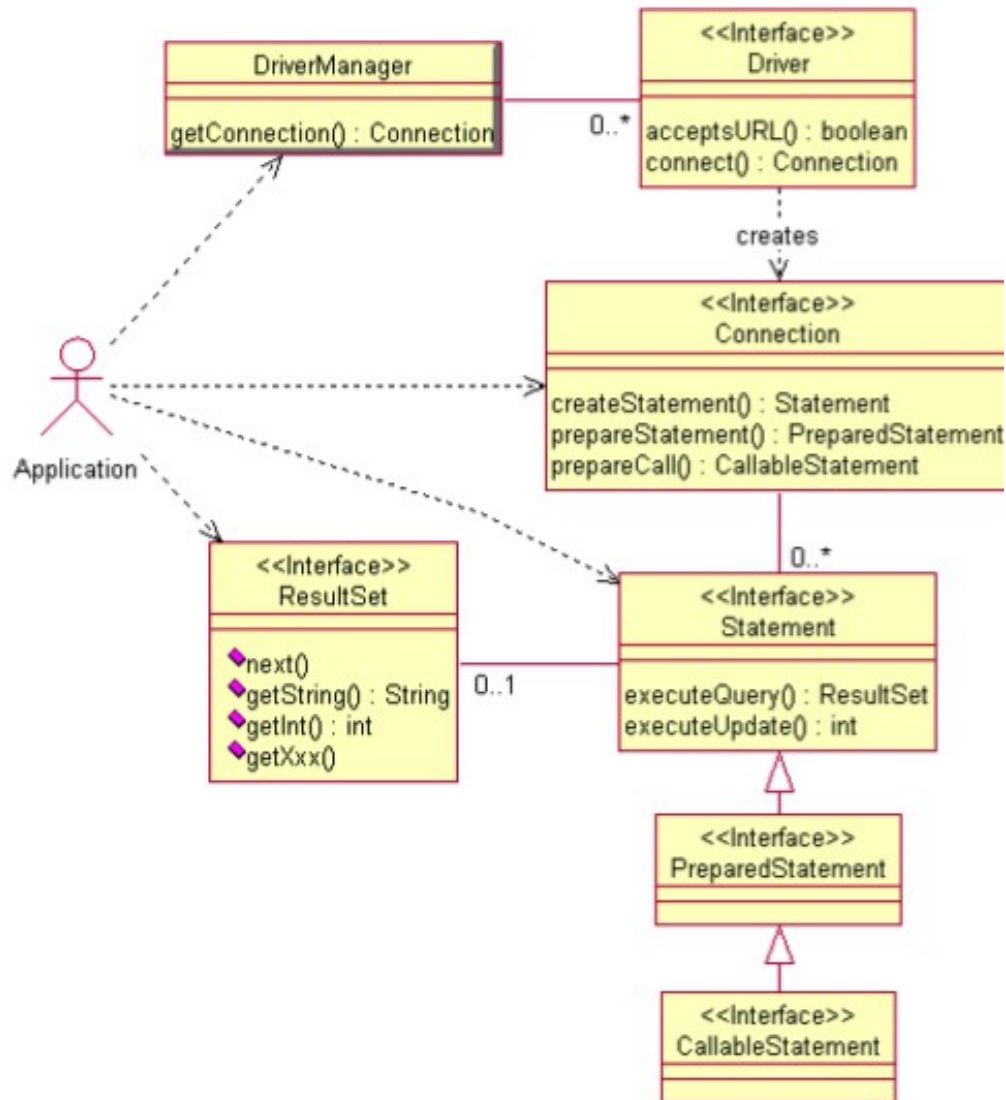
■ Consulta

```
String sqlConsulta = "SELECT abreviatura, nombre from equipo order by abreviatura";
ResultSet rstEquipos = stmt.executeQuery(sqlConsulta);
while( rstEquipos.next() ) {
    String abrv = rstEquipos.getString("abreviatura");
    String nombreCompleto = rstEquipos.getString("nombre");
    System.out.println(abrv + "\t" + nombreCompleto);
}
```



- JDBC tiene mecanismos para mejorar el rendimiento
- Por ejemplo: PreparedStatement
- La sentencia SQL es compilada por el SGBD previamente.
- Se crea con una conexión
- La sentencia puede contener variables marcadas con ?.

```
PreparedStatement preparedSQL =  
conexion.prepareStatement("select * from equipo where abreviatura = ?");  
preparedSQL.setString(1, "ALC");  
  
rstEquipos = preparedSQL.executeQuery();  
while( rstEquipos.next() ) {  
    String abrv = rstEquipos.getString("abreviatura");  
    String nombreCompleto = rstEquipos.getString("nombre");  
    System.out.println(abrv + "\t" + nombreCompleto);  
}
```



Tratamiento de XML



- El JDK incluye dos métodos para analizar XML
 - DOM Parser: lee todo el XML formando el árbol de elementos en memoria
 - Funciona como librería
 - SAX Parser: usado para grandes documentos
 - Funciona como framework. Cada vez que se abre y cierra un elemento se invoca a un *callback* (puntero a una función proporcionada por nosotros)

```
<svg width="1000.0" height="500.0">  
  <desc>Esto es una cadena</desc>  
  <circle cx="100.0" cy="300.0" fill="red">  
</svg>
```

- `svg`, `desc`, `circle` son elementos
- `width`, `height`, `cx`, `cy`, `fill` son atributos



```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
factory.setValidating(true);
factory.setIgnoringElementContentWhitespace(true);
try {
    DocumentBuilder builder = factory.newDocumentBuilder();
    File file = new File("test.xml");
    Document doc = builder.parse(file);

    // TRATAMOS AQUÍ EL DOCUMENTO USANDO LOS MÉTODOS DE LECTURA
    // DE ELEMENTOS DE Document

} catch (ParserConfigurationException e) {
    // TRATAR EXCEPCIÓN
} catch (SAXException e) {
    // TRATAR EXCEPCIÓN
} catch (IOException e) {
    // TRATAR EXCEPCIÓN
}
```




- MiElementHandler es una clase nuestra que hereda de DefaultHandler

```
SAXParserFactory factory = SAXParserFactory.newInstance();
factory.setValidating(true);
try {
    SAXParser saxParser = factory.newSAXParser();
    File file = new File("prueba.xml");
    saxParser.parse(file, new EMiElementHandler());
}
catch(ParserConfigurationException e1) {
    //TRATAR EXCEPCION
}
catch(SAXException e1) {
    //TRATAR EXCEPCION
}
catch(IOException e) {
    //TRATAR EXCEPCION
}
```

- Cada vez que se lee el inicio de un elemento XML se invoca al método

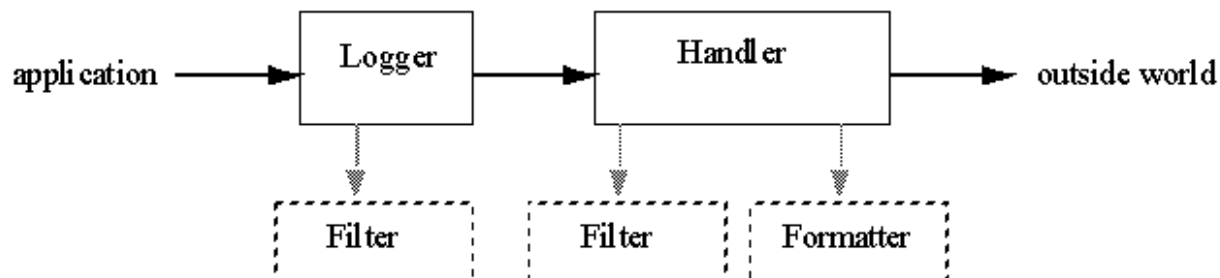
```
startElement(String uri, String localName, String qName, Attributes
attributes)
```

que tendremos definido en nuestra clase MiElementHandler

Logging



- Framework para la emisión eficiente de logs: configurable mediante código o con ficheros
 - Opción no eficiente ni parametrizable
 -
 - `System.out.println("paso por aquí y x=" + x);`
 -
 - Además, se mezclan en consola todos los "logs"
 - Opción eficiente y parametrizable
 - `logger.info("paso por aquí y x=" + x);`
- Actualmente el JDK incorpora el framework Logging





- Uso:

```
public class MiClase {  
    static final Logger logger = Logger.getLogger(MiClase.class.getName());  
  
    void F() {  
        logger.fine("Mensaje de depuración");  
        logger.info("Mensaje de monitorización de funcionamiento");  
        logger.warning("Mensaje de advertencia");  
        logger.severe("Mensaje de error grave");  
    }  
}
```

- Salida por consola:

```
18-nov-2011 19:24:09 MiClase F  
INFO: Mensaje de monitorización de funcionamiento  
18-nov-2011 19:24:09 MiClase F  
ADVERTENCIA: Mensaje de advertencia  
18-nov-2011 19:24:09 MiClase F  
GRAVE: Mensaje de error grave
```



- Si queremos que sólo nos salgan los logs a partir de un nivel, ponemos en el main, al principio:

```
Logger.getLogger("").getHandlers()[0].setLevel(Level.SEVERE);  
System.setProperty("java.util.logging.ConsoleHandler.level", "Level.OFF");
```

- O mejor, usamos un fichero de configuración (*properties* en la terminología Java)

```
java -Djava.util.logging.config.file=./logger.properties -cp  
./classes:./lib/* ClassMain
```

- -D pasa una propiedad definida a la máquina virtual
 - que también podríamos cargar así

```
LogManager.getLogManager().readConfiguration(new  
    FileInputStream("./logger.properties"));
```



```
# especificacion de detalle de log
# nivel de log global
.level = WARNING

# manejadores de salida de log
# se cargaron un manejador de archivos y
# manejador de consola
handlers = java.util.logging.FileHandler,
           java.util.logging.ConsoleHandler

# configuración de manejador de archivos
# nivel soportado para archivos
java.util.logging.FileHandler.level = ALL
# archivo de almacenamiento de las salidas de log
java.util.logging.FileHandler.pattern = ./log/prog3.log
# maximo tamaño de archivo en bytes
java.util.logging.FileHandler.limit = 10485760
# maximo numero de archivos de logs
java.util.logging.FileHandler.count = 3
# clase para formatear salida hacia el archivo de log
java.util.logging.FileHandler.formatter =
    java.util.logging.XMLFormatter
# añadir entrada al ultimo archivo (si es falso escribirá al
# inicio del archivo cuando la aplicación sea ejecutada)
java.util.logging.FileHandler.append = true
```



- Esta configuración genera un fichero prog3.log con este contenido:

```
<?xml version="1.0" encoding="MacRoman" standalone="no"?>
<!DOCTYPE log SYSTEM "logger.dtd">
<log>
  <record>
    <date>2011-11-18T19:37:08</date>
    <millis>1321641428407</millis>
    <sequence>0</sequence>
    <logger>MiClase</logger>
    <level>WARNING</level>
    <class>MiClase</class>
    <method>F</method>
    <thread>10</thread>
    <message>Mensaje de advertencia</message>
  </record>
  <record>
    <date>2011-11-18T19:37:09</date>
    <millis>1321641429282</millis>
    <sequence>1</sequence>
    <logger>MiClase</logger>
    <level>SEVERE</level>
    <class>MiClase</class>
    <method>F</method>
    <thread>10</thread>
    <message>Mensaje de error grave</message>
  </record>
</log>
```

Hibernate

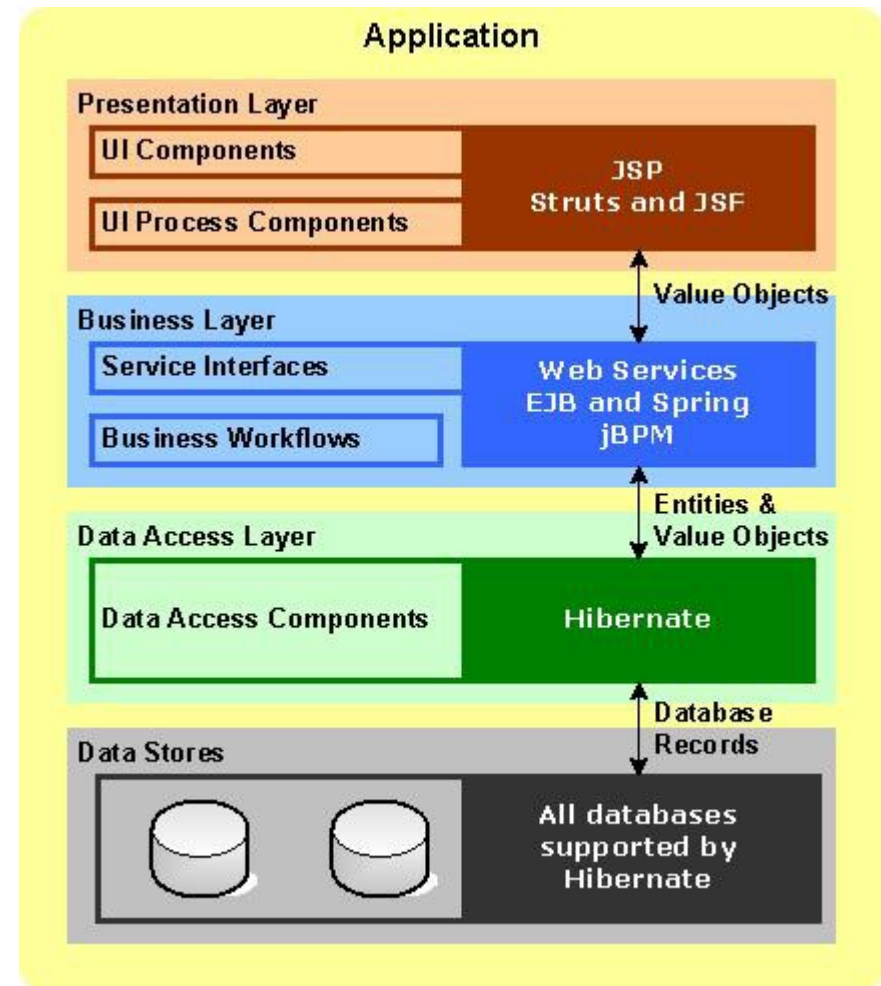


Mapeo Objeto / Relacional (O/R)

A partir de unos ficheros de configuración genera clases que gestionan las operaciones CRUD (create, retrieve, update, delete) que hacen persistentes los objetos en BBDD

Si usamos JDBC tenemos que escribirlas nosotros manualmente

- (insert into, delete from, update,, select ...)
- Con Hibernate nos liberamos de esa tarea





- En el fichero de configuración especificamos parámetros como el nombre de las tablas que asignamos a nuestras clases o cómo se traducen nuestros tipos de datos Java a SQL:

```
<class name="entidades.Temporada" table="temporada" catalog="mibbdd">  
<id name="temporadaId" type="java.lang.Integer">
```

- Guardar objetos es tan sencillo como:

```
Session session = HibernateUtil.getSessionFactory().getCurrentSession();  
session.beginTransaction();  
Temporada temporada = new Temporada(2011, 2012);  
session.save(temporada);  
session.getTransaction().commit();
```

- Que internamente genera la correspondiente sentencia SQL
insert into ...

Apache Commons



- Conjunto de librerías incluídas en el proyecto Apache
 - <http://commons.apache.org/>
- Especialmente útiles son:
 - CLI: parámetros para la utilidades en línea de comando
 - Collections: añade más tipos abstractos de datos
 - Configuration: ficheros de configuración
 - Email
 - FileUpload: subir ficheros a nuestro servidor
 - Math: para operaciones estadísticas

- Ejemplo: Multimap

```
MultiKeyMap mapa=new ....;
MultiKey key = new MultiKey(partido.getLocal(), partido.getVisitante());
// para añadir
this.mapa.put(key, partido);
.....
// para recuperar
MultiKey key = new MultiKey(local, visitante);
Partido p = (Partido) mapa.get(key);
```

Otros frameworks Java



Frameworks de Java en Wikipedia

http://es.wikipedia.org/wiki/Categor%C3%ADa:Frameworks_de_Java

- **GWT (Google Web Toolkit):** generación de aplicaciones ricas web mediante la generación de Javascript a partir de Java
- **Spring:** generación de aplicaciones web empresariales
 - HTML5, REST, AJAX, soporte móviles
 - Mapeo O/R
 - Integración con redes sociales
 - Integración con sistemas basados en mensajería asíncrona
- **Apache Struts:** framework para la generación de aplicaciones web
- **JUnit:** pruebas unitarias
- **JavaFX:** nueva estrategia de Oracle para GUI RIA (rich interface applications)
- **Apache Lucene:** motor de búsqueda para recuperación de información